MAUI and Blazor Frameworks Possibilities for Creating Cross-platform Applications

Aleksei Gaas

Bachelor's thesis
2023

Tomas Bata University in Zlín
Faculty of Applied Informatics

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

Akademický rok: 2022/2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Aleksei Gaas**
Osobní číslo:	**A20495**
Studijní program:	**B0613A140020 Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Možnosti frameworků MAUI a Blazor pro tvorbu multiplatformních aplikací**
Téma práce anglicky:	**MAUI and Blazor Frameworks Possibilities for Creating Cross-platform Applications**

Zásady pro vypracování

1. Popište současný stav technologií pro vývoj multiplatformních aplikací.
2. Zaměřte se na frameworky Microsoft MAUI a Blazor.
3. Navrhněte aplikaci demonstrující možnosti tvorby multiplatformních aplikací ve framworku MAUI s využitím Blazor.
4. Realizujte vývoj navržené aplikace a demonstrujte 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. Microsoft. .NET Multi-platform App UI documentation [online]. Dostupné z: https://learn.microsoft.com/dotnet/maui/.
2. Microsoft. Blazor Tutorial – Build your first Blazor app [online]. Dostupné z: https://dotnet.microsoft.com/learn/aspnet/blazor-tutorial/intro.
3. Microsoft. .NET documentation [online]. Dostupné z: https://learn.microsoft.com/dotnet/.
4. Michael STONIS. Enterprise Application Patterns Using .NET MAUI [online]. V1.0. Redmond, Washington 98052-6399: Microsoft, 2022. Dostupné také z: https://aka.ms/maui-ebook.
5. Microsoft. Visual Studio documentation [online]. Dostupné z: https://learn.microsoft.com/visualstudio/windows/?view=vs-2022.
6. PRICE, Mark J. C# 9 and .NET 5: Modern Cross-Platform Development: Build intelligent apps, websites, and services with Blazor, ASP.NET Core, and Entity Framework Core using Visual Studio Code. 5th Edition. Birmingham, UK: Packt Publishing, 2020. ISBN 978-1800568105.

Vedoucí bakalářské práce: **Ing. Erik Král, Ph.D.**
Ústav počítačových a komunikačních systémů

Datum zadání bakalářské práce: **2. prosince 2022**
Termín odevzdání bakalářské práce: **26. května 2023**

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

doc. Ing. Jiří Vojtěšek, Ph.D.
děkan

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

Ve Zlíně dne 7. prosince 2022

I hereby declare that:

- I understand that by submitting my Bachelor´s Thesis, I agree to the publication of my work according to Law No. 111/1998, Coll., On Universities and on changes and amendments to other acts (e.g. the Universities Act), as amended by subsequent legislation, without regard to the results of the defence of the thesis.
- I understand that my Bachelor´s Thesis will be stored electronically in the university information system and be made available for on-site inspection, and that a copy of the Bachelor´s Thesis will be stored in the Reference Library of the Faculty of Applied Informatics, Tomas Bata University in Zlín, and that a copy shall be deposited with my Supervisor.
- I am aware of the fact that my Bachelor´s Thesis is fully covered by Act No. 121/2000 Coll. On Copyright, and Rights Related to Copyright, as amended by some other laws (e.g. the Copyright Act), as amended by subsequent legislation; and especially, by §35, Para. 3.
- I understand that, according to §60, Para. 1 of the Copyright Act, TBU in Zlín has the right to conclude licensing agreements relating to the use of scholastic work within the full extent of §12, Para. 4, of the Copyright Act.
- I understand that, according to §60, Para. 2, and Para. 3, of the Copyright Act, I may use my work - Bachelor´s Thesis, or grant a license for its use, only if permitted by the licensing agreement concluded between myself and Tomas Bata University in Zlín with a view to the fact that Tomas Bata University in Zlín must be compensated for any reasonable contribution to covering such expenses/costs as invested by them in the creation of the thesis (up until the full actual amount) shall also be a subject of this licensing agreement.
- I understand that, should the elaboration of the Bachelor´s Thesis include the use of software provided by Tomas Bata University in Zlín or other such entities strictly for study and research purposes (i.e. only for non-commercial use), the results of my Bachelor´s Thesis cannot be used for commercial purposes.
- I understand that, if the output of my Bachelor´s Thesis is any software product(s), this/these shall equally be considered as part of the thesis, as well as any source codes, or files from which the project is composed. Not submitting any part of this/these component(s) may be a reason for the non-defence of my thesis.

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.

In Zlín; dated: May 15, 2023 Aleskei Gaas v. r.

.....

Student´s Signature

ABSTRAKT

Cílem práce je zhodnocení možností frameworků MAUI a Blazor pro tvorbu multiplatformních aplikací. V teoretické části se popisuje současný stav technologií pro vývoj multiplatformních aplikací a zaměřuje se na frameworky Microsoft MAUI a Blazor. V praktické části je navrhována aplikace, která demonstuje možnosti tvorby multiplatformních aplikací ve frameworku MAUI s využitím Blazor, provádí se vývoj navržené aplikace a předvádějí se klíčové části řešení. Na závěr jsou hodnoceny dosažené výsledky a formulují se možnosti dalšího rozvoje aplikace.

Klíčová slova: .NET, C#, MAUI, Blazor, WebAssembly, Multiplatformní vývoj, Progresivní Webová Aplikace

ABSTRACT

The main goal of this thesis is to evaluate the possibilities of MAUI and Blazor frameworks for creating multiplatform applications. The theoretical part describes the current state of the art for multiplatform application development and focuses on the Microsoft MAUI and Blazor frameworks. In the practical part, an application is designed that demonstrates the possibilities of creating multiplatform applications in the MAUI framework using Blazor, the development of the proposed application is carried out and the key parts of the solution are demonstrated. Finally, the achieved results are evaluated and possibilities for further development of the application are formulated.

Keywords: .NET, C#, MAUI, Blazor, WebAssembly, Cross-Platform development, Progressive Web Application

I would like to thank my thesis supervisor Ing. et Ing. Erik Král, Ph.D. for their advices and supervision of this thesis.

CONTENTS

INTRODUCTION

The demand for universal and practical applications has significantly increased in recent years. As a result, developers must choose the most relevant approach and technology to create an application that meets the needs of a wide range of users on different devices and platforms. This thesis examines the theoretical and practical aspects of various modern application development methodologies, including native, hybrid, and cross-platform approaches. It provides a detailed analysis of their advantages and disadvantages and considers the current state of the technologies and frameworks for cross-platform application development. Next, the thesis focuses on the popular cross-platform framework .NET MAUI, highlighting its work under the hood, advantages, and differences between MAUI and its predecessor Xamarin. Furthermore, the thesis examines Blazor, an advanced framework that can be used for developing web applications and cross-platform and hybrid applications. It presents WebAssembly as the key technology underlying it, analyzing its advantages and disadvantages.

In the Analysis part, it is proposed to create an application such as Budget Tracker using .NET MAUI and Blazor. The application will help users manage their finances by allowing them to track their expenses and income. The thesis will provide a complete analysis of the application development process, providing insight into the key steps and considerations associated with cross-platform application development. The Budget Tracker app exemplifies how cross-platform app development can be used to create apps that meet users' needs on different platforms.

The main goal of the thesis is to provide valuable insights into modern application development methodologies and technologies by comparing different frameworks, operating systems, and underlying technologies with focusing on .NET MAUI and Blazor.

I. THEORY

1 MODERN WAYS TO DEVELOP AN APPLICATION

Applications have become essential to our lives, with people relying on them for communication, entertainment, education, and more. As a result, companies are increasingly turning to app development to engage with their customers, improve brand presence and increase revenue. However, with multiple platforms and operating systems available, choosing the right approach to development can take time and effort. This is where the concepts of native, hybrid, and cross-platform development come to the rescue. Each scenario has unique advantages and disadvantages, and understanding the differences is crucial for businesses to make informed decisions about their mobile app development strategy.

1.1 Native

You can create native apps for desktops, smart TVs, and so on - but because the most popular target devices are smartphones, native app development is often used to refer to mobile app development.

Native apps are software applications created using a particular programming language for a specific device platform. For example, native apps for iOS are developed using Objective-C or Swift, and native apps for Android are developed using Java or Kotlin. [1]

1.1.1 Advantages

Native application development has several advantages over other approaches, such as hybrid and cross-platform application development. Here are some of the key benefits of developing native applications [2][3]:

- Performance

  Native apps are developed using platform-specific languages and APIs, allowing them to take full advantage of the device's hardware and software. This provides better performance, smoother animations, and a better user experience.

- User experience

  Native apps deliver a high-quality user experience with faster load times, smoother scrolling, and a responsive user interface. This results in a more engaged and satisfied user experience.

- Access to Device Features

  Native apps have access to a wide range of device features, such as camera, GPS, and sensors, allowing for advanced and sophisticated functionality.

- Security

  Native apps are built to meet the strict security standards of app stores and operating systems, giving users a safer environment to download and use apps.

- Ecosystem integration

  Native apps can be easily integrated with the operating system and other native apps, improving user experience and visibility.

1.1.2 Disadvantages

Although native development has a variety of advantages, it also has certain disadvantages that should be considered before deciding on this approach, such as [4][5]:

- Cost

  Native application development can be expensive because it requires separate development teams for each platform and specific skills in platform-specific programming languages and APIs.

- Development time

  Developing a native application for multiple platforms requires more time than other approaches, such as hybrid and cross-platform development.

- Maintenance

  Native apps require separate updates for each platform, which increases the cost and effort of maintenance.

- Limited audience reaches

  Native apps are developed for a specific platform, which limits the audience reach to users of that platform.

- Approval process

  Native apps go through a rigorous approval process with app stores, which can lead to delays and additional costs.

- Fragmentation

  Fragmentation of operating systems and devices can make native app development a challenge, as developers need to ensure compatibility with different gadgets and platform versions.

1.2 Hybrid

Hybrid application development involves creating applications that run on multiple platforms using a single code base. Hybrid apps are developed using web technologies such as HTML, CSS, and JavaScript and then wrapped in a native container that allows them to be installed and run on mobile devices [6].

1.2.1 Advantages

Here are some of the key advantages of developing applications in a hybrid way [6][7][8]:

- Cost-effective

  Hybrid application development is cost-effective because it allows developers to write code once and deploy it to multiple platforms, reducing development costs and time.

- Faster development time

  Hybrid application development provides a faster development cycle than native application development because a single code base can be used across multiple platforms.

- Ease of maintenance

  Hybrid applications can be updated with a single codebase, making it easy and cost-effective to maintain and update the application across multiple platforms.

- Easy integration with web technologies

  Hybrid apps are created using web technologies such as HTML, CSS, and JavaScript, making them easy to integrate with web technologies and third-party tools.

- Huge audience reaches

  Hybrid apps can be deployed across multiple platforms, providing wider audience reach and market penetration.

1.2.2 Disadvantages

Here are some of the important disadvantages of hybrid app development [6][7][8]:

- Performance

  Hybrid applications are developed using web technologies, which may result in lower performance than native applications, especially for complex and resource-intensive applications.

- User experience

  Hybrid apps may not provide the same level of user experience as native apps, especially for platform-specific functionality and user interface elements.

- Limited access to device features

  Hybrid apps have limited access to devise features and APIs compared to native apps, which can restrict the creation of advanced and complex functionality.

- Platform limitations

  Hybrid apps are limited by the web technologies' capabilities, which can lead to limitations for advanced functionality and user interface elements.

- Dependence on third-party tools

  Hybrid application development heavily depends on third-party tools, which can lead to compatibility issues and require additional time and effort for integration and maintenance.

- Approval process

  Hybrid apps are subject to the same rigorous app store approval process as native apps, which can cause delays and additional costs.

1.3 Cross-platform

Cross-platform application development involves creating applications that run on multiple platforms using a single codebase. Cross-platform applications are developed using cross-platform frameworks and libraries, which allow developers to write code once and deploy it to various platforms. Cross-platform applications provide a balance between performance, cost-effectiveness, and development speed, making them an attractive option for many companies [9].

1.3.1 Advantages

Cross-platform application development and hybrid application development are similar in that they both allow developers to create applications that run on multiple platforms. However, cross-platform application development has some advantages over hybrid development. Here are some of the key benefits of cross-platform application development over hybrid development [9][10][11]:

- Performance

  Cross-platform application development can offer better performance than hybrid application development because the code is compiled initially for each platform, while hybrid applications are run in a web preview.

- Access to platform-specific features

  Cross-platform applications can have full access to platform-specific functions and APIs, while hybrid applications may have limited access to those functions.

- Code reusability

  Cross-platform application development provides more code reuse across platforms than hybrid application development, resulting in cost savings and reduced development time.

- Native-like user experience

  Cross-platform app development allows for a native-like user experience with smooth animations, fast load times, and a responsive user interface. In contrast, hybrid apps cannot provide the same level of user experience.

- Easier maintenance

  Cross-platform apps are easier to maintain than hybrid apps because the code base is common to all platforms, making implementing updates and bug fixes easier.

- A larger developer community

  Cross-platform application development has a larger developer community and more tools and resources than hybrid application development, making it easier for enterprises to find qualified developers and resources.

1.3.2 Disadvantages

Here are the key disadvantages of cross-platform app development [10][11]:

- Development tools

  Cross-platform application development requires specialized development tools, which can be more complex and challenging to use than tools used for hybrid application development.

- Learning curve

  Cross-platform application development can be more challenging for developers than hybrid application development because of the complexity of development tools and the need to maintain compatibility across multiple platforms.

- Limited access to platform-specific features

  Although cross-platform application development can provide full access to platform-specific features, some cannot be accessed or integrated seamlessly.

- Performance

  Although cross-platform application development can offer better performance than hybrid application development, performance may still be lower than a fully native application, especially for complex and resource-intensive applications.

- Debugging and Troubleshooting

  Debugging and troubleshooting in cross-platform application development can be more complicated than in hybrid application development due to the complex nature of development tools and the need to maintain compatibility across multiple platforms.

- Customization

  Cross-platform application development may not allow the same level of customization as hybrid application development because the development tools are designed to create a consistent user experience across multiple platforms.

1.4 Summary

Following all the above should be noted that the development of native, hybrid, and cross-platform applications has its advantages and disadvantages, and the choice of the best approach depends on the specific needs of the business and the target audience. Native applications provide the best performance and usability, but their development can be expensive and time-consuming. Hybrid apps offer faster development time and cost savings but may not give the same user experience as native apps. Cross-platform application development balances these two options, providing access to platform-specific features, a native-like user experience, and cost savings through the ability to reuse code.

Although cross-platform application development has some drawbacks, it remains a popular and effective approach for companies looking to provide better audience reach and ease of maintenance. Moreover, thanks to the ongoing development of tools and frameworks, the gap between cross-platform and native development is getting smaller, and cross-platform app development is becoming more powerful, flexible, and customizable, allowing developers to create apps that work just as well as native apps.

2 CURRENT STATE OF TECHNOLOGIES FOR CROSS-PLATFORM APPLICATION DEVELOPMENT

The current state of technology for cross-platform app development is diverse and dynamic. Many tools and frameworks allow developers to create apps that run on different platforms using a single codebase. Some popular technologies for cross-platform app development are .NET MAUI, Kotlin Multiplatform Mobile, Flutter, React Native, and Ionic, as shown in Figure 1. Each tool has its advantages and disadvantages, depending on project requirements and developer preferences. [12]

2.1 Popular cross-platform frameworks

2.1.1 . NET MAUI

. NET Multiplatform App User Interface (.NET MAUI) is open source and is an evolution of Xamarin.Forms, which is a popular framework for creating cross-platform mobile applications on .NET. In fact, the .NET MAUI extends Xamarin by adding support for Windows, macOS, and Linux through the Uno Platform, improving performance, productivity, and toolset. .NET MAUI also integrates with other .NET technologies, such as Blazor for UI, ASP.NET Core for web services, and Orleans for distributed systems. [13]

2.1.2 Kotlin Multiplatform Mobile

Kotlin Multiplatform Mobile (KMM) is an SDK for iOS and Android app development that offers all the combined benefits of creating cross-platform and native apps. It allows you to share code for logic elements that often fall out of sync while keeping the advantages of native programming, including excellent app performance and full access to the Android and iOS SDKs. [14]

2.1.3 Flutter

Flutter is an open-source framework developed by Google for building beautiful, natively compiled mobile, web, desktop, and embedded mobile applications from a single codebase. It is used with Dart, an object-oriented programming language by Google. Flutter uses its own graphics engine named Skia, which bypasses platform UI libraries and communicates directly with Skia in the engine layer providing instructions for GPU. [15]

2.1.4 React Native

React Native is an open-source framework developed by Meta (Facebook), used to build natively rendered applications for mobile platforms from a single codebase using the React JavaScript library. React Native allows the creation of iOS and Android apps and other platforms like the web and desktop without writing platform-specific code. [16]

2.1.5 Ionic

Ionic is an open-source toolkit for building high-quality cross-platform mobile apps from a single codebase using JavaScript. It is designed to work quickly with hardware-accelerated transitions and touch-optimized gestures by default. It has built-in support for JavaScript frameworks such as React, Angular, or Vue. Ionic applications can run as Android, iOS, Electron, and PWA from the same codebase. [17]

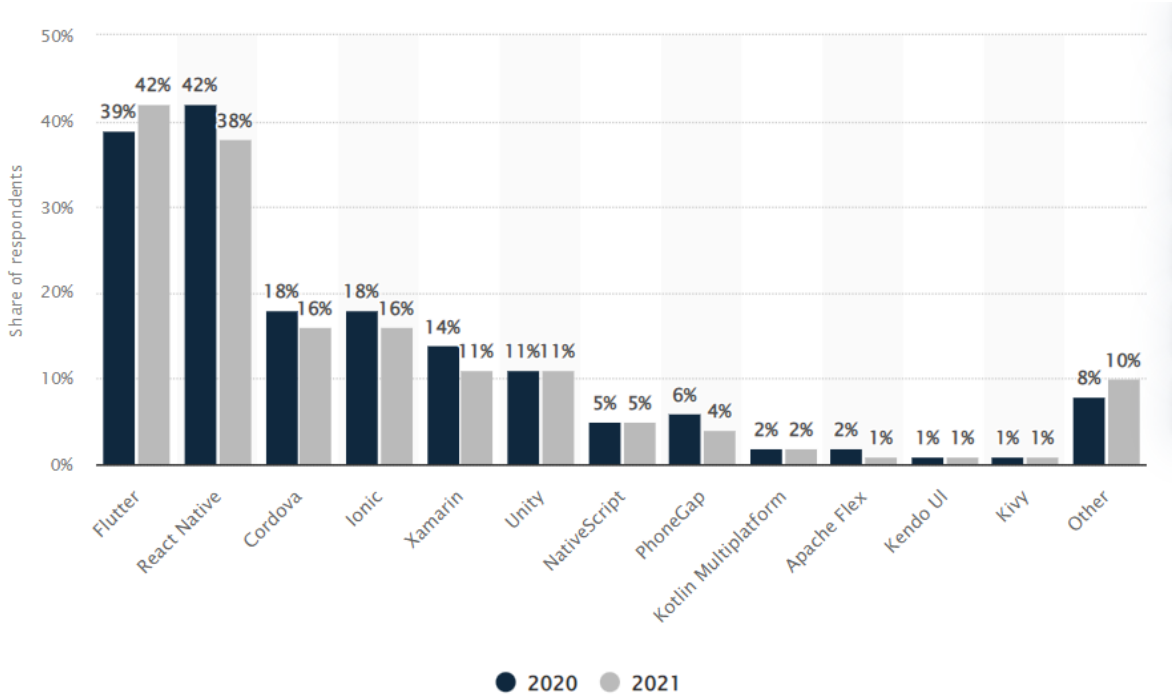

Figure 1 Cross-platform mobile frameworks used by software developers worldwide from 2020 to 2021 [18]

2.2 Popular operating systems for desktop and mobile devices

2.2.1 What is an operating system?

The operating system (OS) is the software that controls computer hardware and provides services to computer programs. It is the most important type of system software that runs on a computer or mobile device; without it, the device would be unable to perform any functional tasks.

The primary purpose of the operating system is to provide an interface between the user and the computer hardware, allowing the user to interact with the computer more intuitively and efficiently. It manages computer resources such as memory, storage, and processing power and ensures that every program running on the computer has fair and efficient access to these resources.

The operating system also provides several services, including file management, security, networking, and device drivers. These services are essential to the operation of most computer programs, and the operating system provides a consistent and reliable way for programs to access these services.

There are many popular operating systems, as shown in Figure 2, including desktop operating systems such as Windows, macOS, and Linux and mobile operating systems such as Android and iOS. Each operating system has features and characteristics, but all aim to provide a stable and reliable platform for running computer programs [19][20].

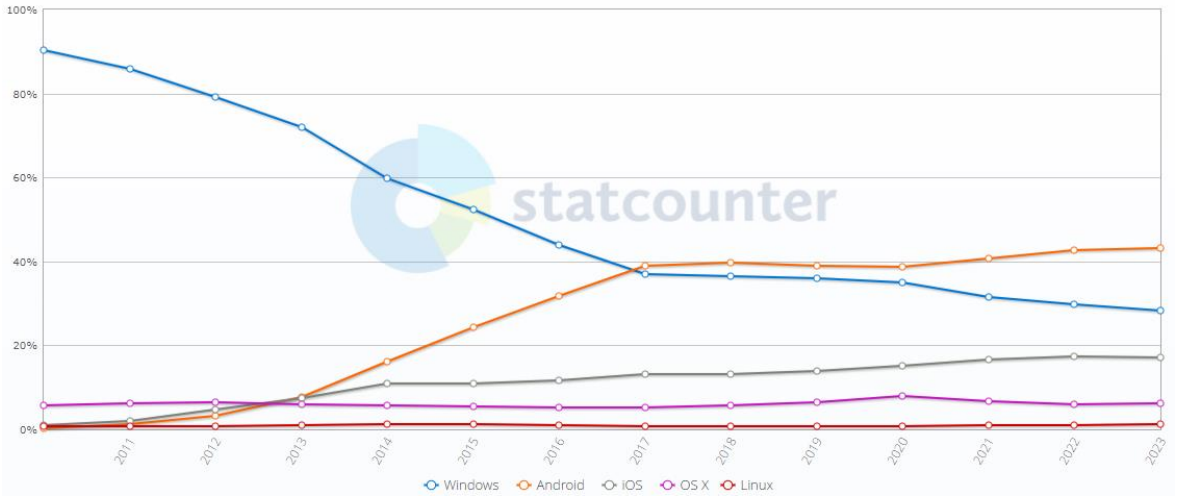

Figure 2 Operating System Market Share Worldwide [21]

2.2.2 Windows

Windows is a popular operating system for desktops and laptops. It has a user-friendly interface and supports a wide range of software applications. Windows also supports a variety of hardware configurations, making it easy to find the suitable device for your needs. It has a significant market share, which means that software developers are more likely to create applications that are compatible with the system. Windows is also known for its vulnerability to malware and viruses, so it is essential to ensure proper security criteria. [22]

2.2.3 macOS

macOS is the operating system used on Apple's Mac computers. It has an elegant and intuitive user interface known for its reliability and security. macOS also comes with a variety of built-in applications. One of the main advantages of macOS is its tight integration with other Apple devices, such as the iPhone and iPad. This makes it easy to share files and use apps across devices. However, macOS is less common than Windows, so some programs may not be compatible with it. [23] [24]

2.2.4 Linux

Linux is an open-source operating system popular with developers and technology enthusiasts. It is highly customizable and can be used for various applications, including servers, desktops, and embedded devices. One of the main advantages of Linux is its security and stability. It is also free to use, making it a cost-effective option for businesses and organizations. However, Linux can be more difficult to use than other operating systems, especially for people unfamiliar with the command-line interface. [25] [26]

2.2.5 Android

Android is the operating system used by most mobile devices that Apple does not manufacture. It is based on the Linux kernel and is known for its flexibility and customization options. One of the main advantages of Android is the wide range of applications that can be downloaded from the Google Play store. Android also supports a variety of hardware configurations, making it a popular choice for device manufacturers. However, like Windows, Android is also vulnerable to malware and viruses. [27] [28]

2.2.6 iOS

iOS is the operating system in Apple's mobile devices, including the iPhone, iPad, and iPod Touch. It comes with various default apps and is known for its ease of use and security features, including biometric authentication (Face ID or Touch ID) and app sandboxing. iOS devices are also tightly integrated with other Apple devices, making it easy to share files and use apps across devices. One of the main advantages of iOS is the app ecosystem, which is known for its high quality and security standards. However, iOS devices are more expensive than Android devices, which can be a barrier for some users. Also, iOS is not as customizable as Android, which can be a drawback for some users who prefer more control over their devices. [29] [30]

3 .NET MAUI

3.1 How does MAUI work?

MAUI provides a set of APIs and toolsets that allow developers to write user interface code in platform-agnostic mode using .NET and C# programming languages. Developers can use Visual Studio to create and edit MAUI applications, and the MAUI toolkit generates the platform-specific code needed to run the application on different operating systems.

Under the hood, MAUI uses several platform-specific renderers and controls to make the user interface look and feel native to each platform. MAUI also provides support for device-specific features such as biometric authentication, camera access, and notifications through a set of cross-platform APIs.

One of the key advantages of MAUI is that it allows developers to share much of their code base across multiple platforms, which can result in significant time and cost savings. In addition, MAUI applications can be deployed to various app stores and distribution channels, making it easier to reach a broad audience [31][32].

3.2 Benefits of using MAUI

Using the .NET ecosystem, MAUI offers developers a wide range of benefits, which are listed below [31][32][33]:

- Performance

  MAUI applications provide native-like performance because the framework uses native APIs of each platform, resulting in smoother animations, faster loading times, and a more responsive user interface.

- Single Codebase

  Like other cross-platform frameworks, MAUI allows developers to write a single codebase that can be compiled for multiple platforms, resulting in cost savings, reduced development time, and simplified maintenance.

- Unified development experience

  With MAUI, developers can create apps for all major platforms using a single set of tools and APIs, simplifying the development process and reducing training time.

- UI capabilities

  MAUI provides an advanced UI experience with improved support for complex layouts, better theming capabilities, and the ability to create custom controls. For UI, you can use XAML markup language or the Blazor hybrid to write in HTML instead. It allows developers to easily convert their web applications written in Blazor into native applications for various platforms.

- Flexibility

  With MAUI, developers gain greater flexibility and control over the user interface, allowing them to create more customizable and adaptable applications that meet the unique needs of businesses and audiences.

- .NET ecosystem

  MAUI is built on .NET 6, which provides improved performance, security, and compatibility with the wide range of libraries, tools, and resources available in the .NET, making integrating the existing code base into MAUI applications easier.

3.3 Differences between Xamarin and MAUI

As mentioned earlier, .NET MAUI is an evolution of Xamarin, so it adds new features and improvements to make development easier [34][35][36]:

- Scope

  Xamarin is a cross-platform development environment that allows developers to create native mobile apps for iOS and Android using a single codebase. Conversely, MAUI is a multiplatform framework that enables developers to create native apps for iOS, Android, macOS, and Windows using a single codebase.

- Project structure

  Xamarin uses a shared code architecture that allows developers to write a single codebase that can be used across platforms. MAUI builds on this architecture and introduces a new multitargeting capability to allow developers to use multiple platforms in a single project file.

- Toolkit

  MAUI introduces a new set of tools to simplify the development process, including a new Visual Studio extension that provides a unified application creation experience for all supported platforms. Xamarin, on the other hand, has its own set of tools and extensions specifically designed for mobile development.

- User interface

  Although Xamarin and MAUI use XAML to define user interfaces, MAUI introduces new features that improve the user interface, such as support for flexible layout containers, adaptive icons, and a new theme engine.

- Compatibility

  MAUI is designed to be fully compatible with existing Xamarin.Forms applications, allowing developers to migrate their existing codebase to the new framework without major changes. However, some features and APIs may need to be updated to take full advantage of the new features presented in MAUI.

- Hot Reload

  In Xamarin, Hot Reload is available for XAML-based UIs, allowing developers to make changes to the UI and instantly see those changes without having to rebuild the application. However, changes made to the code underlying the UI still require reassembly and redeployment of the application. MAUI takes this further by extending the Hot Reload feature to cover both the user interface and the underlying code. This means that developers can make changes to both the UI and the code and see an instant reflection of those changes without rebuilding or redeploying the application.

- Renderer and Handler Architectures

  Xamarin uses renderers to create controls, as shown in Figure 3, and developers must use custom renderers to customize the user interface of native controls. However, these renderers can affect application performance and size. On the other hand, .NET MAUI uses a handler architecture that is loosely related to native assembly, as shown in Figure 4. The result is a lightweight application with improved performance on the native platform.

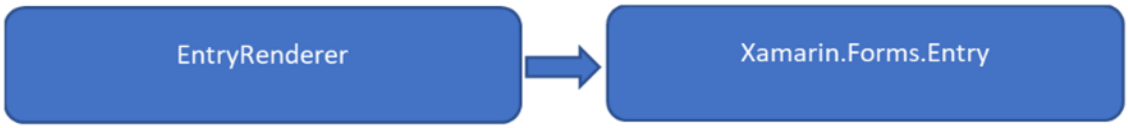

Figure 3 Xamarin Renderer Architecture [36]

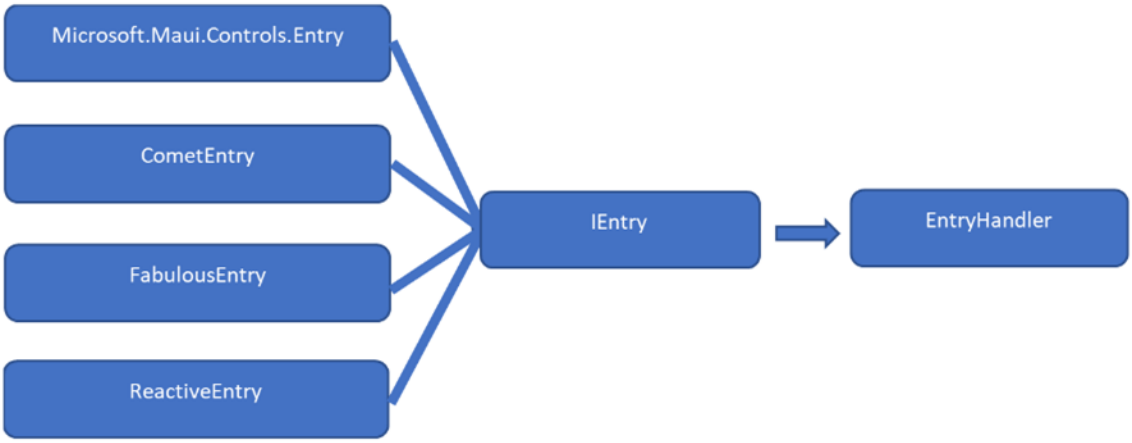

Figure 4 .NET MAUI Handler Architecture [36]

4 BLAZOR

4.1 What is Blazor?

Blazor is a web interface framework for creating interactive client-side web applications using C# instead of JavaScript. Blazor and MAUI are part of the .NET ecosystem and can be used together to create hybrid applications that share code and functionality between web and native platforms.

Blazor allows developers to write Web UI code using Razor and C# syntax and then compile that code into WebAssembly, which can be run in a browser according to the principle shown in Figure 5. MAUI allows developers to write native UI code using Blazor, .NET, and C# and then deploy that code to different platforms.

To connect Blazor and MAUI, developers can use the Blazor WebView component, a hybrid solution that allows developers to host a Blazor web application inside their MAUI application. The Blazor WebView component bridges the web and native worlds, allowing developers to share code and functionality between their web and native applications.

With the Blazor WebView component, developers can write business logic and UI code once in Blazor and reuse that code in their web and native applications. This saves time and effort and makes it easier to create and maintain hybrid applications running on different platforms [37][38][39][40].

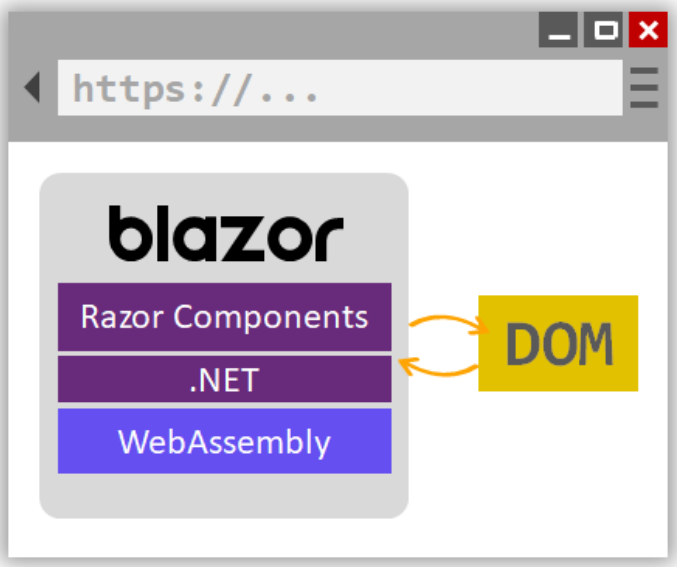

Figure 5 The working principle of Blazor WebAssembly [39]

4.2 WebAssembly

4.2.1 What is WebAssembly?

WebAssembly is a low-level binary format that allows developers to run compiled code in the browser. It is designed to be fast, efficient, and secure and can be used to build complex applications running near-native speeds. WebAssembly provides a virtual machine that can execute code compiled from a wide range of programming languages, including C/C++, C#, Rust, and others. When a developer compiles their code into WebAssembly format, the resulting binary file can be loaded into the browser and executed by the WebAssembly runtime [40].

4.2.2 Advantages of using WebAssembly

There are several advantages of using WebAssembly for web development [40][41][42][43]:

- Performance

  WebAssembly is designed to run at near-native speed, making it a powerful tool for building complex web applications that require high performance.

- Security

  WebAssembly provides a sandbox-like execution environment that helps improve security by preventing malicious code from accessing the user's computer or data.

- Portability

  WebAssembly is designed to be platform-independent, which means that code compiled in the WebAssembly format can run on different platforms, including desktop and mobile devices.

- Language flexibility

  WebAssembly can be used with a variety of programming languages, including C/C++, Rust, and C#, allowing developers to use whichever language best suits their needs.

- Code Reuse

  WebAssembly allows developers to reuse existing code written in other programming languages, saving time and resources.

- Compatibility

  WebAssembly is supported by all major web browsers, meaning that code written in WebAssembly can run in any modern browser without plugins or extensions.

4.2.3 Disadvantages of using WebAssembly

Although WebAssembly offers many advantages, it may not be the best choice for every web development project, and developers should carefully consider the potential drawbacks before deciding to use it because [40][44][45]:

- Limited browser support

  Although all major browsers now support WebAssembly, older browsers may not have it, which may limit the number of users who can access your application.

- Debugging

  Debugging WebAssembly code can be more difficult than traditional web development languages such as JavaScript, as debugging tools are not as well developed.

- Code size

  Because WebAssembly is designed to run at near-native speeds, the code can be larger than in traditional web development languages, affecting page load times and requiring more bandwidth.

- Learning curve

  Using WebAssembly requires learning new tools and concepts, making it more challenging for developers used to working with traditional web development languages.

- Limited ecosystem

  Because WebAssembly is a relatively new technology, the ecosystem of libraries and tools is not as well developed as traditional web development languages, which may limit what developers can do.

II. ANALYSIS

5 OVERVIEW

As we have already found out, the development of cross-platform applications has become increasingly popular in recent years as the demand for applications running on different platforms continues to grow. In this part, we look at how these frameworks can be used to develop a cross-platform Budget Tracker application.

Budget Tracker is an important tool for managing personal finances. It allows users to track their income and expenses and monitor their financial progress.

In this part, we will discuss the process of developing the Budget Tracker application using the .NET MAUI and Blazor frameworks. We will look at the development process of each framework, including the tools and resources required.

Overall, this analysis will provide valuable insights into the potential of MAUI and Blazor frameworks for creating cross-platform applications and their suitability for Budget Tracker application development.

6 SOLUTION STRUCTURE

Solution has listed projects, as shown in Figure 6:

- BudgetTracker.Shared

  The project provides a shared user interface written in Blazor and service definitions.

- BudgetTracker.Native

  The project with .NET MAUI for native communication with different platforms.

- BudgetTracker.Web

  The project allows us to run this application not only on native platforms but also on web pages and use it as PWA.

This approach enables the application to be accessed from a wide range of devices, providing flexibility and accessibility to users.

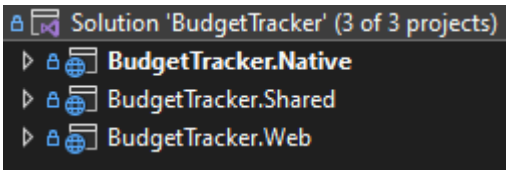

Figure 6 Solution structure

6.1 BudgetTracker.Shared

6.1.1 Overview

This project contains the user interface, service definitions, and domain models for the Budget Tracker application written in C# using Blazor. It provides a shared code base for the application functionality and is used in native and web projects. UI components define the structure and appearance of the application's user interface, service definitions represent communication with business logic, and domain models represent application data objects. The project acts as a shared library between other parts of the application. You can see the project structure in Figure 7.

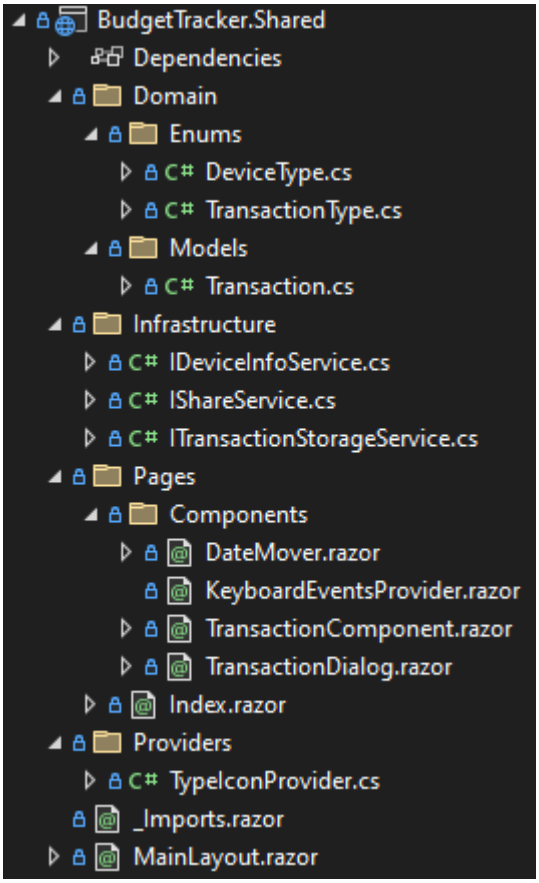

Figure 7 BudgetTracker.Shared project structure

6.1.2 Stack

The project uses the following technologies:

- .NET 6
- Blazor
- MudBlazor as NuGet package

  MudBlazor is an open-source library of ready-to-use UI components for creating web applications using Blazor. It offers many customizable components, such as buttons, forms, and data grids, providing a consistent user experience across devices and platforms. MudBlazor prioritizes accessibility and performance and can be easily integrated into a Blazor project using the NuGet package. It is widely used in the .NET community to create responsive and visually appealing applications [46][47].

6.1.3 Domain

The folder contains all the necessary domain entities used in the application.

- Transaction.cs

  The main entity, which represents any transaction that the user can make.

```
public record Transaction
{
    public int Id { get; set; } = -1;

    public string Description { get; set; } = string.Empty;

    public TransactionType Type { get; set; }

    public double? Amount { get; set; }

    public DateTime UpdatedAt { get; set; }
}
```

- TransactionType.cs

  Enumerator, which represents the possible types of the user's transactions.

```
public enum TransactionType
{
    Income,
    Housing,
    Transportation,
    Food,
    Personal,
    Health,
    Entertainment,
    Education,
    Miscellaneous,
    Savings,
    ChildCare,
    Technology,
    Insurance,
    Taxes,
    Pets,
    Travel
}
```

- DeviceType.cs

  Enumerator, which represents the possible types of the user's device.

```
public enum DeviceType
{
    Unknown,
    NativeMobile,
    NativeDesktop,
    Web
}
```

6.1.4 Infrastructure

The folder contains all the interfaces that define the necessary functionality that must be implemented by other projects to provide a shared codebase for the application. These interfaces act as a contract between the different parts of the solution and ensure consistency and compatibility between them.

- IDeviceInfoService.cs

  This interface defines a method for getting the type of device the application runs on.

```
public interface IDeviceInfoService
{
    Task<DeviceType> GetDeviceTypeAsync();
}
```

  The **GetDeviceTypeAsync** method returns a task that contains a DeviceType enum value, which can be used to determine the type of the user's device.

- IShareService.cs

  This interface defines several methods for sharing, importing, and downloading data in the application.

```
public interface IShareService
{
    Task ShareDataAsync();

    Task ImportDataAsync();

    Task ImportDataAsync(IBrowserFile browserFile);

    Task DownloadDataAsync();
}
```

  The **ShareDataAsync** method allows users to share data on native platforms.

  The **ImportDataAsync** method allows users to import data into the application.

  The **ImportDataAsync** method accepting the **IBrowserFile** argument, enables the user to import data from a file selected through the browser.

  The **DownloadDataAsync** method allows the user to download data from the application on web.

- ITransactionStorageService.cs

  This interface defines methods to access, add, update, and delete transaction data from the repository.

```
public interface ITransactionStorageService
{
    Task<Dictionary<TransactionType, List<Transaction>>>
GetTransactionsByYearMonthAsync(DateTime date);

    Task<Dictionary<TransactionType, List<Transaction>>>
GetTransactionsByYearAsync(DateTime date);

    Task<Transaction> GetTransactionByIdAsync(int id);

    Task AddNewTransactionAsync(Transaction transaction);

    Task UpdateTransactionAsync(int id, Transaction transaction);

    Task RemoveTransactionAsync(int id);

    string GetDirectory();

    Task SaveTransactionsAsync(IEnumerable<Transaction> transactions);
}
```

  The **GetTransactionsByYearMonthAsync** method returns a dictionary containing lists of transactions grouped by transaction type for a given year and month.

  The **GetTransactionsByYearAsync** method returns a dictionary containing lists of transactions grouped by transaction type for a given year.

  The **GetTransactionByIdAsync** method returns a single transaction with the specified ID.

  The **AddNewTransactionAsync** method adds a new transaction to the storage.

  The **UpdateTransactionAsync** method updates an existing transaction in the repository.

  The **RemoveTransactionAsync** method removes the transaction from the storage.

  The **GetDirectory** method returns the path to the directory where transaction data is stored.

  The **SaveTransactionsAsync** method saves transaction collection to the storage.

6.1.5 Providers

Providers can be considered as specific static classes that provide the application with a particular simple service or functionality. In the context of this application, we have only one **TypeIconProvider**, which provides the functionality of mapping a **TransactionType** to the corresponding icon. It can be easily accessed from other parts of the application that need this functionality. It is a centralized and reusable way of providing this functionality rather than duplicating code in different application parts.

TypeIconProvider.cs

```
public static class TypeIconProvider
{
    public static string GetIcon(TransactionType type)
        => type switch
        {
            TransactionType.Income
                => Icons.Material.Rounded.AttachMoney,

            TransactionType.Housing
                => Icons.Material.Rounded.Home,

            TransactionType.Transportation
                => Icons.Material.Rounded.DirectionsBus,

            TransactionType.Food
                => Icons.Material.Rounded.Fastfood,

            TransactionType.Personal
                => Icons.Material.Rounded.AccountCircle,

            TransactionType.Health
                => Icons.Material.Rounded.LocalHospital,

            TransactionType.Entertainment
                => Icons.Material.Rounded.SportsEsports,

            TransactionType.Education
                => Icons.Material.Rounded.School,

            TransactionType.Miscellaneous
                => Icons.Material.Rounded.Interests,

             ...

            TransactionType.Travel
                => Icons.Material.Rounded.FlightTakeoff,

            _ => string.Empty,
        };
}
```

All icons are taken from the MudBlazor Icons [48].

6.1.6 Pages and components

This folder contains the Blazor pages and components for the application. These elements are responsible for defining the user interface and the behavior of the various views and components in the application. In the context of a Blazor application, a page is a module that represents a route in the application. The corresponding page component is rendered and displayed when a user navigates to a particular route.

The Budget tracker application has only one page, **Index.razor**.

Each page and component can be divided into a visual part (.razor) and a logical part (.razor.cs). Writing all these parts in one .razor file is also possible, but it is more convenient and readable when separated.

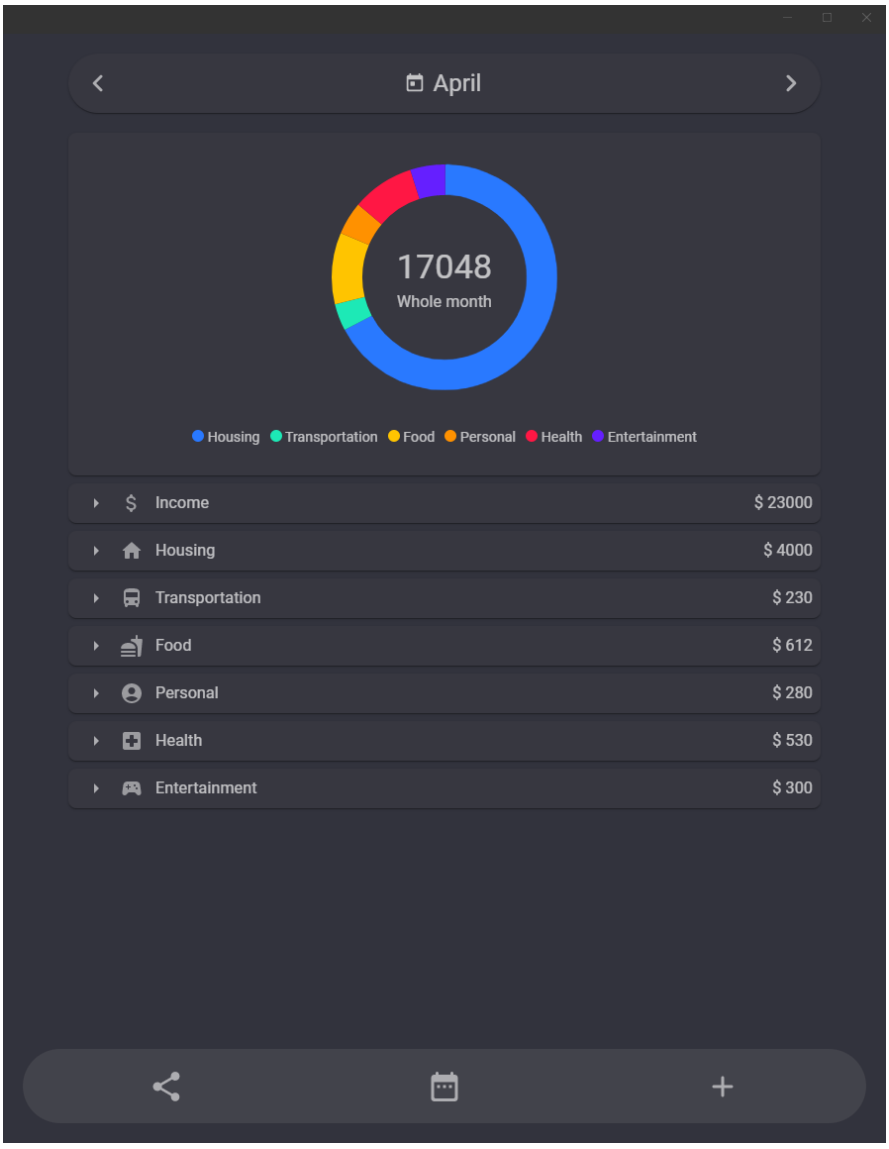

Figure 8 Main window for month transactions displaying

- Index.razor and Index.razor.cs

The file represents the main application window. Let's have a look at some interesting parts of the file.

Index.razor

```
<MudSwipeArea Style="min-height: 100%; max-height: fit-content; overflow:
visible"
              OnSwipe="@(async (d) => await OnSwipe(d))">
...
</MudSwipeArea>
```

Index.razor.cs

```
private async Task OnSwipe(SwipeDirection d)
{
    if (TransactionDialogOpened || PickerOpened)
        return;

    switch (d)
    {
        case SwipeDirection.RightToLeft:
            await MoveMonth(SelectedMenu == 0 ? 1 : 12);
            break;

        case SwipeDirection.LeftToRight:
            await MoveMonth(SelectedMenu == 0 ? -1 : -12);
            break;
    }
}
```

The **MudSwipeArea** element is from the MudBlazor library and provides easy tracking of the user's swipes on any device with a touchscreen. It simply invokes the function **OnSwipe** from **Index.razor.cs** with the direction of the swipe. In the Budget tracker application's context, it is used for the faster change of the month the user is currently looking at. So when the user swipes from right to left, it changes the current month to the next and otherwise to the previous month.

Index.razor

```
@if (SelectedMenu == 0)
{
    ...
}
else
{
    ...
}
```

Here you can see how easy it is to display different parts of the user interface code by adding a simple **if-else** statement directly to the .razor file. In the context of this file, this statement renders different components, the graph for the monthly display and all transactions for this month will be rendered when the **SelectedMenu** property is set to **0**, and the graph for the yearly display will be rendered otherwise, as you can see in Figures 8 and 9.

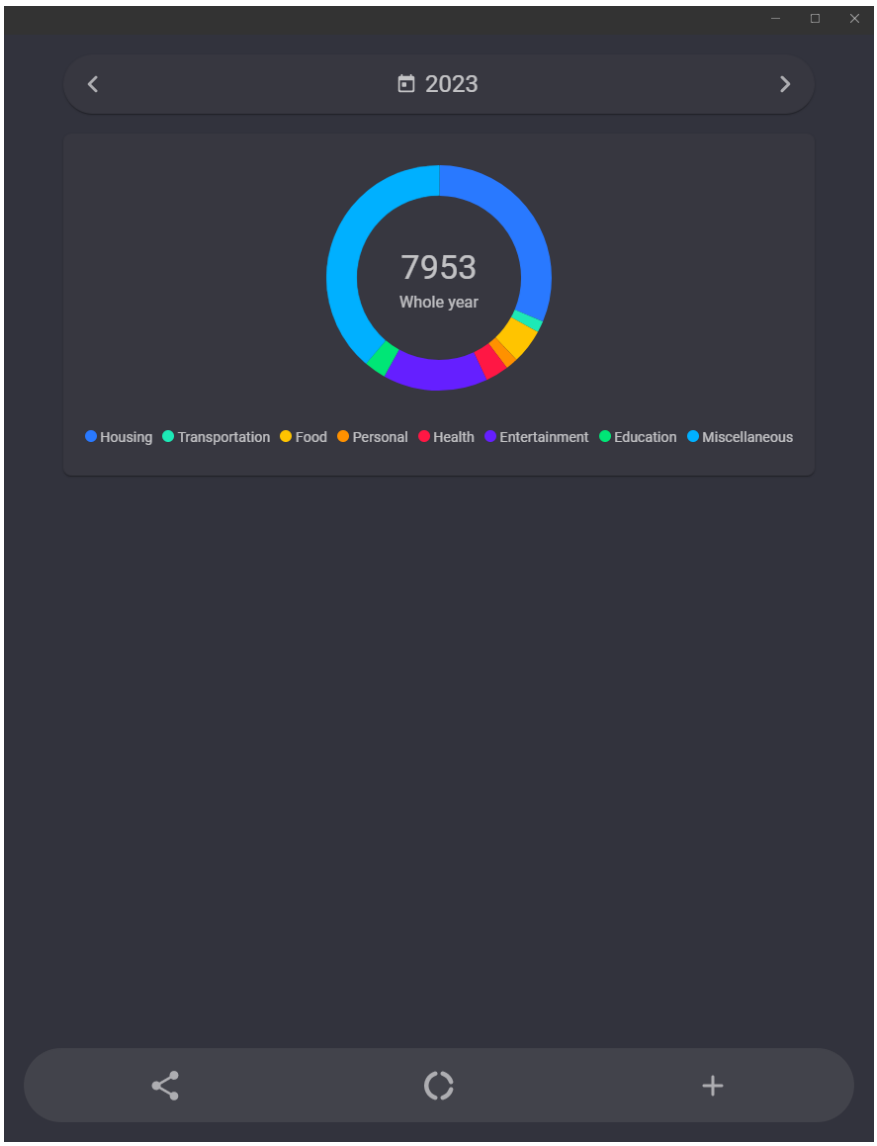

Figure 9 Main window for year transactions displaying

Index.razor

```
@foreach (var item in MonthTransactionsDictionary)
{
	<TransactionComponent Type="item.Key"
		Values="item.Value"
		OnButtonClickEvent="OpenDialogForEditTransactionAsync" />
}
```

Here you can see how several identical parts can be rendered using the **foreach** statement. The **TransactionComponent** is the application's own component, which will be described a little later, but you can already see them in Figures 8 and 10.

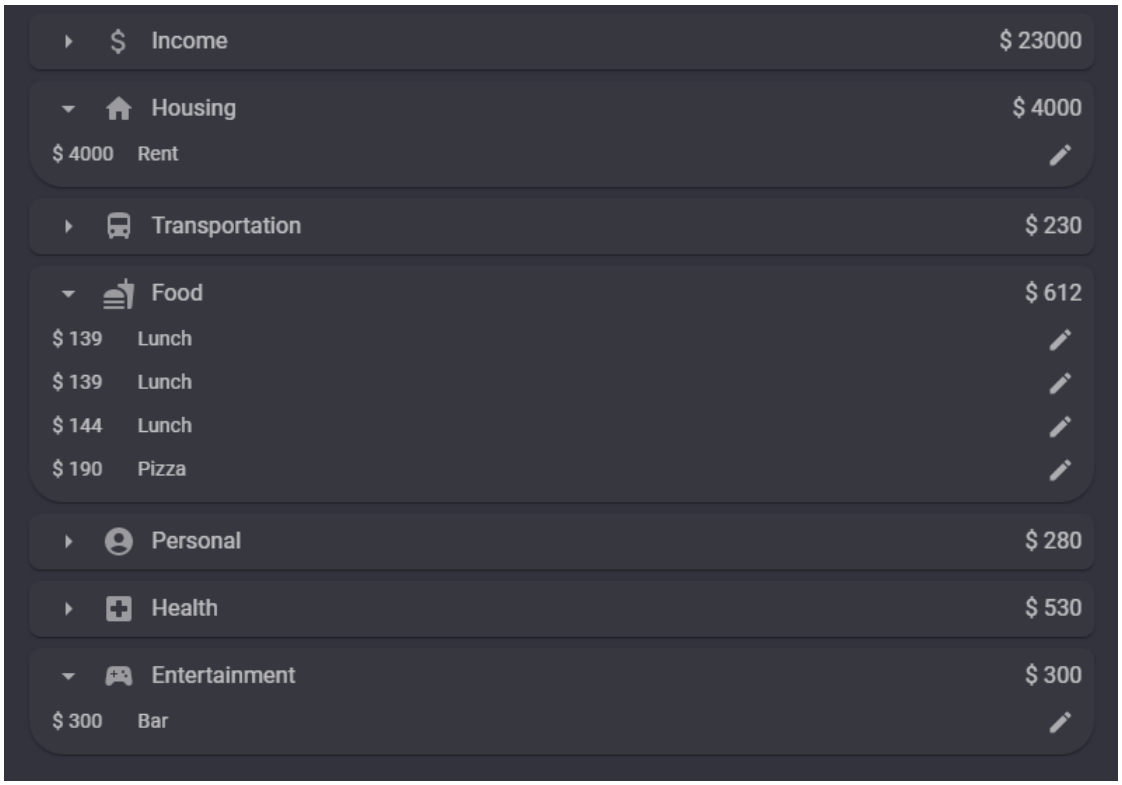

Figure 10 TransactionComponent components inside of the main window

Index.razor

```
<div class="Footer rounded-pill">
    <MudMenu Icon="@Icons.Material.Rounded.Share"
             Size="Size.Large"
             ActivationEvent="@(MouseEvent.LeftClick)"
             TransformOrigin="@Origin.TopCenter"
             AnchorOrigin="@Origin.CenterCenter">
        ...
    </MudMenu>

    <MudIconButton Class="FooterButton"
                   Icon="@(SelectedMenu == 1 ?
Icons.Material.Rounded.DonutLarge : Icons.Material.Rounded.DateRange)"
                   Size="Size.Large"
                   OnClick="SwitchWindow" />

    <MudIconButton Class="FooterButton"
                   Icon="@Icons.Material.Rounded.Add"
                   Size="Size.Large"
                   OnClick="OpenDialogForNewTransactionAsync" />
</div>
```

Here is the footer, with three elements inside it, as shown in Figures 8 and 9.

MudIconButton is a component of the MudBlazor library, which is a simple button with no text but a specified icon.

The second element calls the **SwitchWindow** method, when the user clicks on it, which is responsible for correctly selecting the window menu and providing the necessary data for the graphs.

Index.razor.cs

```
private async Task SwitchWindow()
{
    SelectedChartIndex = -1;

    if (SelectedMenu == 0)
    {
        SelectedMenu = 1;
        await RefreshYearChartData();
    }
    else if (SelectedMenu == 1)
    {
        SelectedMenu = 0;
        await RefreshMonthDictionary();
    }
}
```

The third element calls the **OpenDialogForNewTransactionAsync** method when the user clicks on it. It is responsible for opening the **TransactionDialog** component, where the user can add new transactions, which you can see in Figure 11.

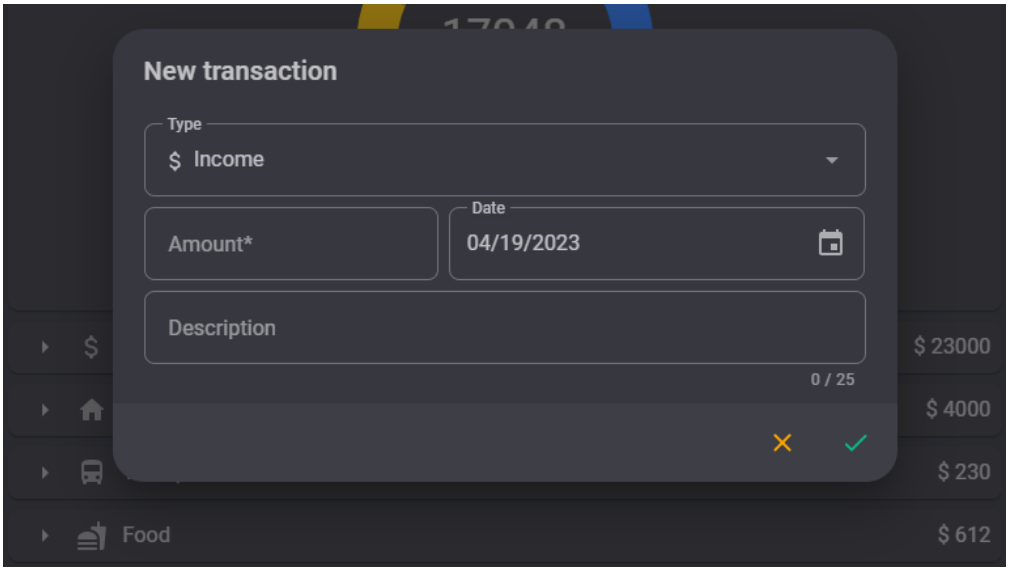

Figure 11 TransactionDialog component for creating a new transaction

The first element is the most difficult of these. Let's look at it a little deeper.

Index.razor

```
<MudMenu Icon="@Icons.Material.Rounded.Share"
            Size="Size.Large"
            ActivationEvent="@(MouseEvent.LeftClick)"
            TransformOrigin="@Origin.TopCenter"
            AnchorOrigin="@Origin.CenterCenter">

    <ActivatorContent>
        <MudIconButton Class="FooterButton"
                       Icon="@Icons.Material.Rounded.Share"
                       Size="Size.Large" />
    </ActivatorContent>

    <ChildContent>
        @if (CurrentDevice == DeviceType.NativeMobile || CurrentDevice ==
DeviceType.NativeDesktop)
        {
            <MudMenuItem OnClick="ShareData">
                    ... Share
            </MudMenuItem>
            <MudMenuItem OnClick="ImportData">
                    ... Import
            </MudMenuItem>
        }
        else if (CurrentDevice == DeviceType.Web)
        {
            <MudMenuItem OnClick="DownloadData">
                <MudIcon Icon="@Icons.Material.Rounded.Download" /> Download
            </MudMenuItem>
            <MudFileUpload T="IBrowserFile" FilesChanged="ImportDataWeb"
Accept="@(".bdtr")">
                    ... Import
            </MudFileUpload>
        }
    </ChildContent>
</MudMenu>
```

This is the MudBlazor menu component that creates a drop-down menu with options for sharing, importing, and downloading data.

The component has two main sections, defined by the **ActivatorContent** and **ChildContent** tags. The **ActivatorContent** section contains a button that, when clicked, activates the menu. The **ChildContent** section contains menu items displayed when the menu is opened.

The menu items are provisionally displayed depending on the type of device in use. The menu items for sharing and importing data are displayed if the device is native mobile or native desktop. The menu items for downloading and importing data are displayed if the device is a web device.

To find out which user device is currently in use, the **GetDeviceTypeAsync** method from the **IDeviceInfoService** interface is called on the page initialization:

Index.razor.cs

```
[Inject]
private IDeviceInfoService DeviceInfoService { get; set; } = default!;

private DeviceType CurrentDevice { get; set; } = DeviceType.Unknown;

protected async override Task OnInitializedAsync()
{
    ...
    CurrentDevice = await DeviceInfoService.GetDeviceTypeAsync();
    ...
    await base.OnInitializedAsync();
}
```

- DateMover.razor and DateMover.razor.cs

DateMover is a application component that allows the user to select a date or year from a calendar. You can see it on the top of the main application window in Figures 8 and 9. It consists of **MudDatePicker** and **MudPaper** library components. **MudDatePicker** is hidden and used to choose a date, and **MudPaper** displays the selected month or year along with two buttons to jump to the previous or next month or year.

Index.razor

```
<DateMover Date="SelectedDate"
           OnClickLeft="@(() => MoveMonth(SelectedMenu == 0 ? -1 : -12))"
           OnClickRight="@(() => MoveMonth(SelectedMenu == 0 ? 1 : 12))"
           OnDatePicked="SetDateFromPicker"
           IsForYear="@(SelectedMenu == 1)"
           OnPickerClosed="@(() => PickerOpened = false)"
           OnPickerOpened="@(() => PickerOpened = true)" />
```

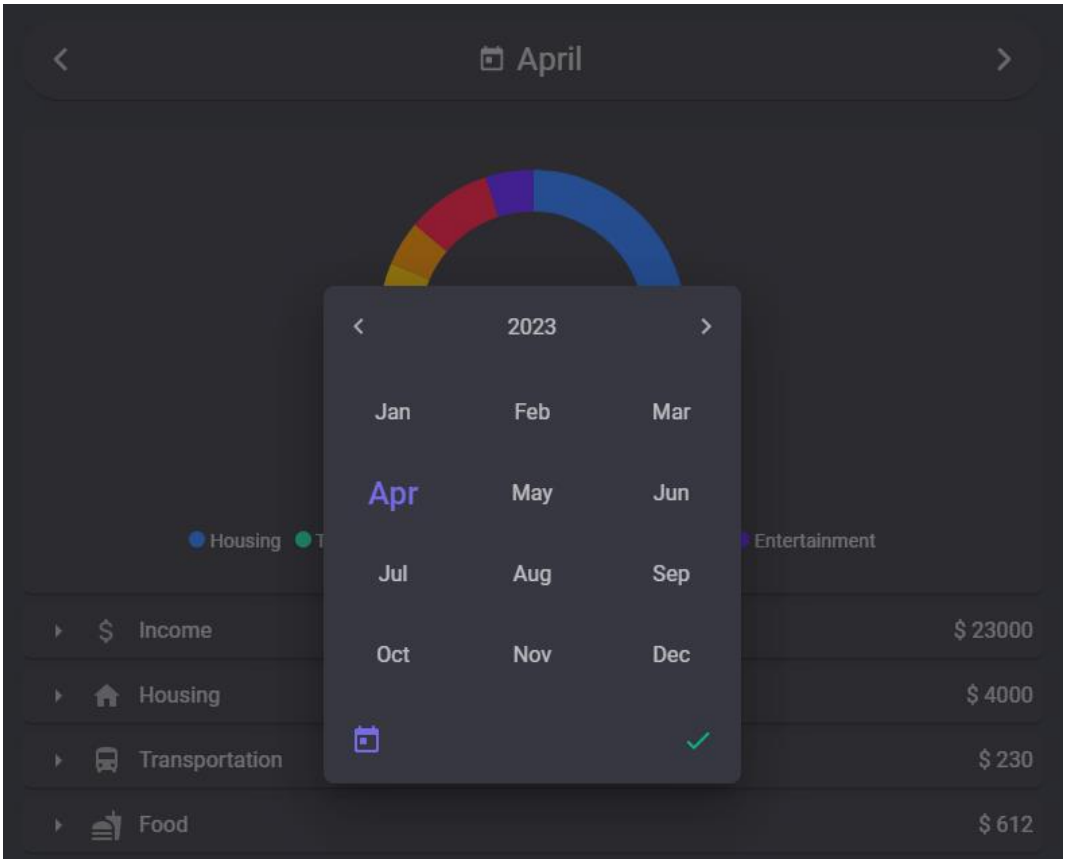

Figure 12 Opened date picker on DateMover component for month transactions displaying

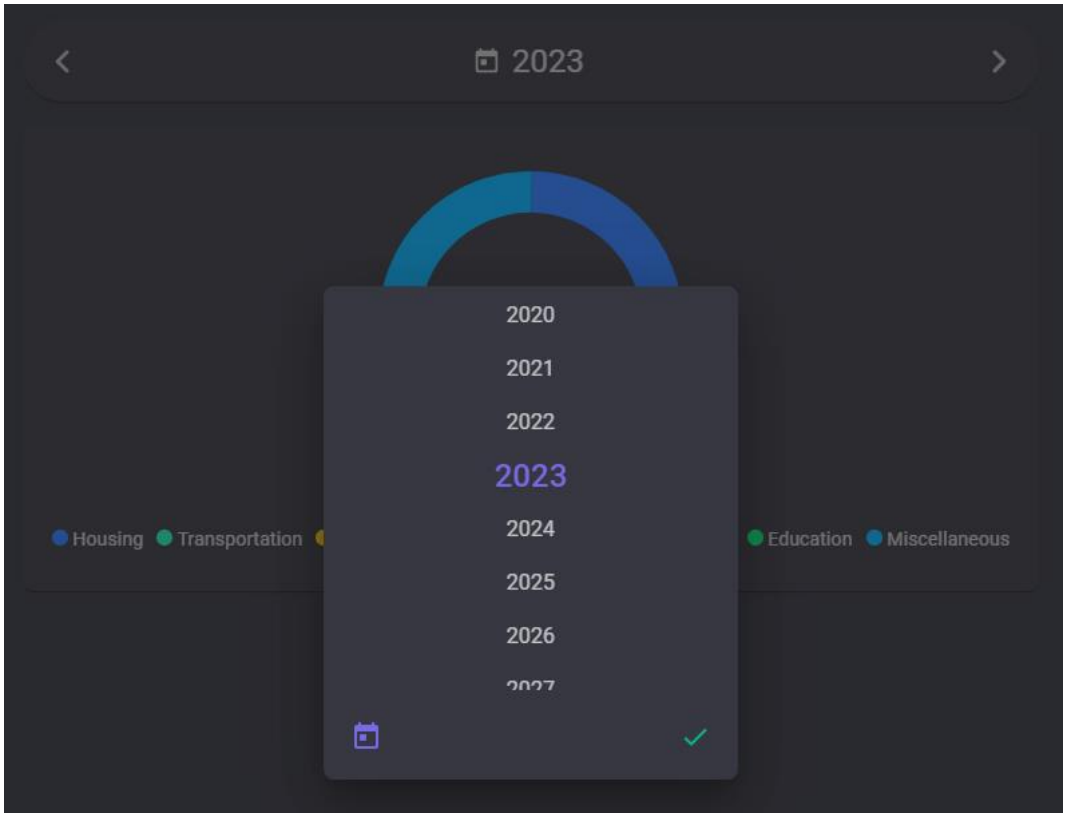

Figure 13 Opened date picker on DateMover component for year transactions displaying

- TransactionComponent.razor and TransactionComponent.razor.cs

TransactionComponent displays a card containing **MudTreeView** library component, which is used to display the transaction list in the tree structure. The component accepts three parameters: transaction type, transaction value list, and an event callback for edit button click. You can look at the component in Figures 8, 10 and 14.

TransactionComponent.razor

```
<MudCard ...
       Class="@("my-2 RadiusTransition " + (Expanded ? "rounded-b-xl" : ""))">
    <MudTreeView ...>
        <MudTreeViewItem ...
             ExpandedChanged="@((x) => Expanded = x)" >
            <div class="ValuesFrame">
                @foreach (var item in Values)
                {
                    <MudGrid ...>
                        <MudItem xs="3" sm="1" Class="align-self-center">
                            $ @item.Amount
                        </MudItem>
                        <MudItem xs="8" s sm="10" Class="align-self-center">
                            @item.Description
                        </MudItem>
                        <MudItem xs="1" Style="text-align:end">
                            <MudIconButton Size="Size.Small"
                                           Icon="@Icons.Material.Rounded.Edit"
                                           OnClick="@(() =>
OnButtonClick(item.Id))" />
                        </MudItem>
                    </MudGrid>
                }
            </div>
        </MudTreeViewItem>
    </MudTreeView>
</MudCard>
```

Thanks to the in-line **if else** statement with the **Expanded** variable, rendering the same element with different classes and styles is possible. You can see that the border radius is bigger for expanded elements.

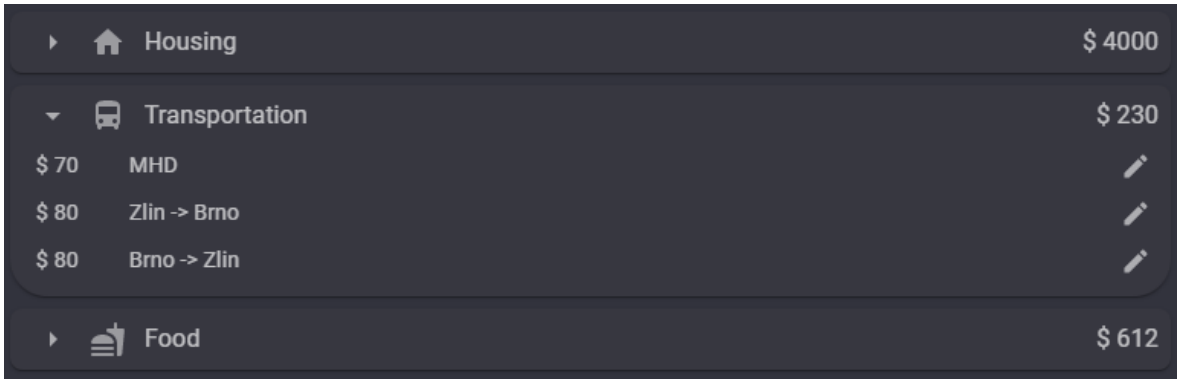

Figure 14 TransactionComponent

- TransactionDialog.razor and TransactionDialog.razor.cs

The **TransactionDialog** is a dialog component that allows users to enter transaction data, such as transaction type, amount (price), date, and description. It includes several child input library components, such as **MudSelect**, **MudNumericField**, **MudDatePicker**, and **MudTextField**, responsible for displaying and validating a particular input field. You can see this component in Figure 11.

The component takes several parameters, including **ForEdit**, **Transaction**, and **SelectedDate**. **ForEdit** is a boolean value that determines whether the dialog is used to edit an existing transaction or to create a new one. A **Transaction** is an object representing transaction data for editing. **SelectedDate** is a **DateTime** object representing the currently selected date.

TransactionDialog.razor

```
<MudSelect @bind-Value="Transaction.Type"
           T="TransactionType"
           Label="Type"
           Variant="Variant.Outlined">
    @foreach (var item in Enum.GetValues<TransactionType>())
    {
        <MudSelectItem Value="@item">
            <MudIcon Icon="@TypeIconProvider.GetIcon(item)" Size="Size.Small"/>
            @(" " + item)
        </MudSelectItem>
    }
</MudSelect>
```

Here you can see how all enumeration values can be shown so that the user can select one, as in Figure 15.

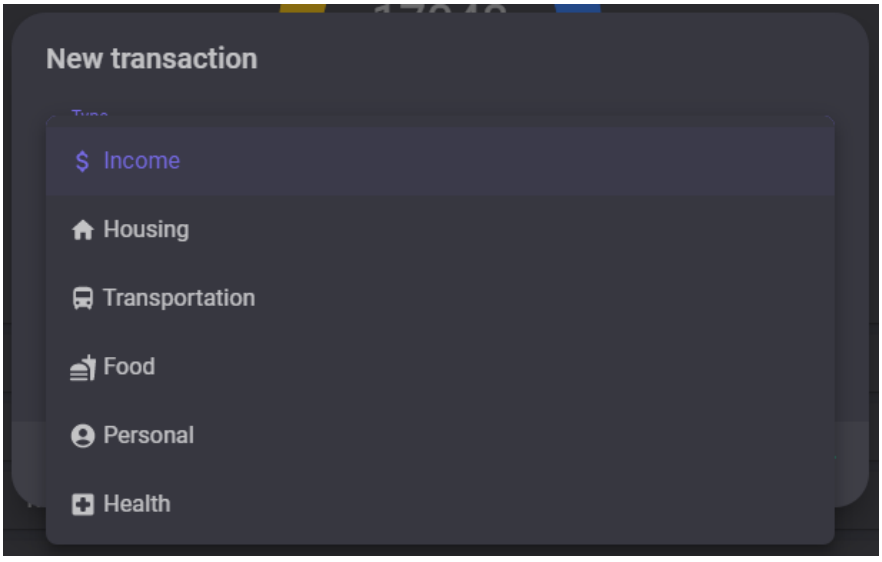

Figure 15 Selecting a transaction type in the TransactionDialog component

TransactionDialog.razor

```
<MudNumericField @ref="MudNumericPrice"
                 @bind-Value="Transaction.Amount"
                 T="double?"
                 Label="Amount"
                 Min="1"
                 Max="99999999"
                 Format="F2"
                 HideSpinButtons="true"
                 Required
                 RequiredError=""
                 Variant="Variant.Outlined">
</MudNumericField>
```

Here you can see that the Amount field is required and has its minimum and maximum possible values, so the user cannot input the incorrect data, as in Figure 16.

It is also not allowed to enter a description longer than 25 characters, but the field can be completely empty.

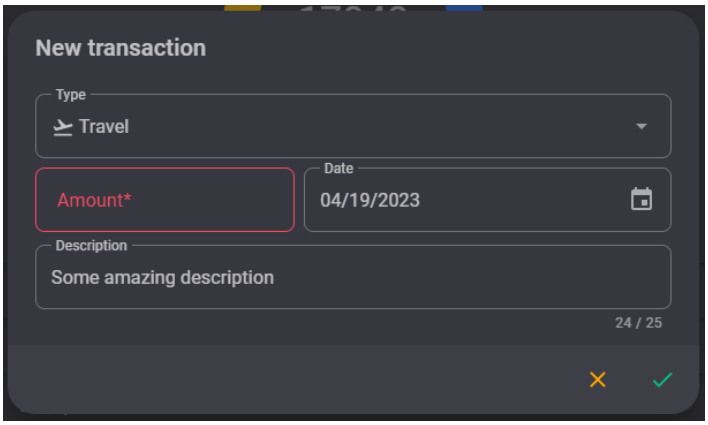

Figure 16 Invalid input in the TransactionDialog component

As already mentioned, this component is also used to edit a transaction by clicking on the **Pen** button in the **TransactionComponent** from Figure 14, as shown in Figure 17.

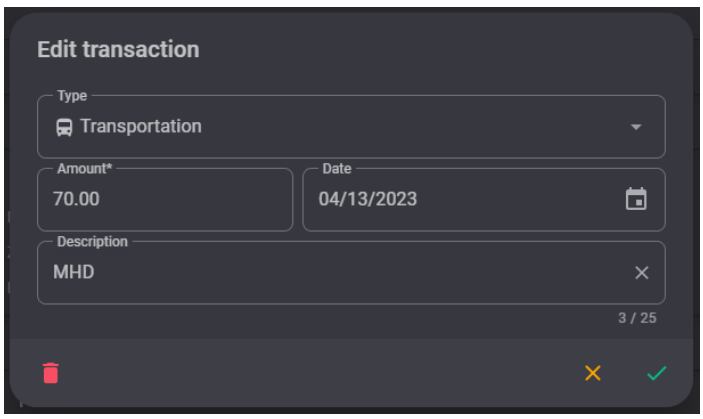

Figure 17 TransactionDialog component for transaction editing

- KeyboardEventsProvider.razor

The **KeyboardEventsProvider** is a custom component that enables registering and invoking actions when a keyboard event occurs. It uses a **CascadingValue** component to provide the current instance of the **KeyboardEventsProvider** to all the child components.

The component is a simple wrapper around a **div** element with a **tabindex** attribute to enable focus and an **@onkeyup** event handler to capture keyboard events. The **BindOnKeyUp** method is used to register an action to be executed when a keyboard event occurs. When a keyboard event is triggered, the **OnKeyUpInvoke** method is called and iterates over all registered actions, invoking them with the event argument.

KeyboardEventsProvider.razor

```
<CascadingValue Value="this">
    <div tabindex="0" class="FullScreen" @onkeyup="OnKeyUpInvoke">
        @ChildContent
    </div>
</CascadingValue>

@code {
    [Parameter]
    [EditorRequired]
    public RenderFragment ChildContent { get; init; } = default!;

    private List<Action<KeyboardEventArgs>> OnKeyUps { get; set; } = new();

    public void BindOnKeyUp(Action<KeyboardEventArgs> target)
    {
        ...
        OnKeyUps.Add(new Action<KeyboardEventArgs>(target));
    }

    private void OnKeyUpInvoke(KeyboardEventArgs e)
    {
        foreach (var item in OnKeyUps)
        {
            item.Invoke(e);
        }
    }
}
```

For this component to work, it must be placed on top of all other elements in **MainLayout.razor**, which is the component used as the layout template for application pages. It defines the layout structure and any common components that should be included on all pages:

MainLayout.razor

```
@inherits LayoutComponentBase

<KeyboardEventsProvider>
    <MudThemeProvider IsDarkMode="true" Theme="CustomTheme" />
    <MudDialogProvider />
    <MudSnackbarProvider />

    <div class="page">
        <main>
            <div class="d-flex justify-content-center main">
                <div class="content">
                    @Body
                </div>
            </div>
        </main>
    </div>
</KeyboardEventsProvider>
```

Then it is possible to get the instance of the **KeyboardEventsProvider** in all childs like this:

Index.razor.cs

```
[CascadingParameter]
public KeyboardEventsProvider KeyboardEvents { get; set; } = default!;
```

And bind the necessary method on the needed event:

Index.razor.cs

```
protected async override Task OnInitializedAsync()
{
    KeyboardEvents.BindOnKeyUp(async (e) => await OnKeyUp(e));
    ...
    await base.OnInitializedAsync();
}
```

As a result, it is possible to handle all events from the keyboard in a custom method:

Index.razor.cs

```
private async Task OnKeyUp(KeyboardEventArgs e)
{
    ...
    switch (e.Key)
    {
        case "ArrowRight":
            await MoveMonth(SelectedMenu == 0 ? 1 : 12);
            break;

        case "ArrowLeft":
            await MoveMonth(SelectedMenu == 0 ? -1 : -12);
            break;

        case "ArrowUp":
            await MoveMonth(12);
            break;

        case "ArrowDown":
            await MoveMonth(-12);
            break;

        default:
            return;
    }
}
```

In the context of this application, if the user presses the arrow keys (up, down, left, or right), the method calls the **MoveMonth** method with a parameter specifying the direction and number of months to move the calendar. Otherwise, the method returns and does not perform any further actions.

Also this component is used in the **TransactionDialog**, so if the **Enter** key was pressed, it calls the **Submit** method which is responsible for submitting the data and closing the dialog box. If the **Escape** key was pressed, it calls the **Cancel** method which is responsible for cancelling the dialog box without submitting any data:

TransactionDialog.razor.cs

```
[CascadingParameter]
public KeyboardEventsProvider KeyboardEvents { get; set; } = default!;

protected async override Task OnInitializedAsync()
{
    KeyboardEvents.BindOnKeyUp(async (e) => await OnKeyUpDialog(e));
    ...
    await base.OnInitializedAsync();
}
```

```
public async Task OnKeyUpDialog(KeyboardEventArgs e)
{
    switch (e.Key)
    {
        case "Enter":
            await Submit();
            break;

        case "Escape":
            Cancel();
            break;

        default:
            return;
    }
}
```

6.2 BudgetTracker.Native

6.2.1 Overview

This project is built using the .NET MAUI framework and provides a cross-platform implementation for both Windows and Android operating systems. It allows for direct communication with the underlying hardware and operating system APIs, providing access to device-specific features and capabilities. The project handles interactions with hardware components, such as storage, sensors, etc. To communicate with the UI, it implements the service interfaces defined in **BudgetTracker.Shared**. You can see the project structure in Figure 18.

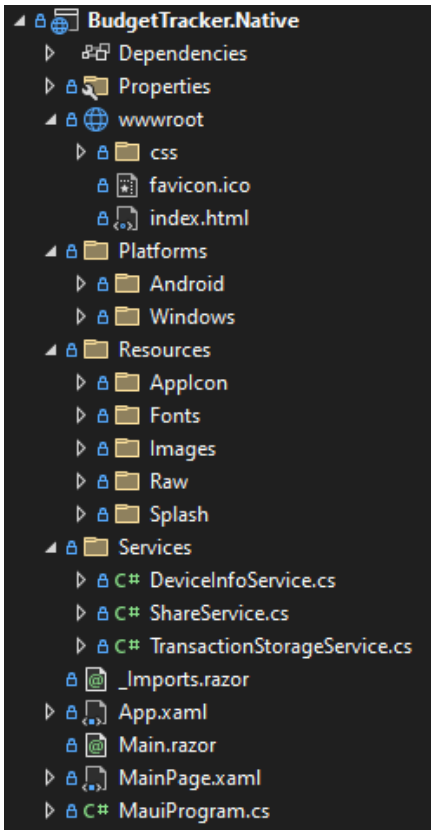

Figure 18 BudgetTracker.Native project structure

6.2.2 Stack

The project uses the following technologies:

- .NET 6
- .NET MAUI
- BudgetTracker.Shared as project reference

6.2.3 Entry point

- MauiProgram.cs

```
public static class MauiProgram
{
    public static MauiApp CreateMauiApp()
    {
        var builder = MauiApp.CreateBuilder();
        builder
            .UseMauiApp<App>()
            .ConfigureFonts(fonts =>
            {
                fonts.AddFont("OpenSans-Regular.ttf", "OpenSansRegular");
            });

        builder.Services.AddMauiBlazorWebView();

#if DEBUG
        builder.Services.AddBlazorWebViewDeveloperTools();
        builder.Logging.AddDebug();
#endif

        builder.Services.AddMudServices();

        builder.Services.AddScoped<ITransactionStorageService,
TransactionStorageService>();
        builder.Services.AddScoped<IShareService, ShareService>();
        builder.Services.AddScoped<IDeviceInfoService, DeviceInfoService>();

        return builder.Build();
    }
}
```

MauiProgram.cs is the file responsible for configuring the **MauiApp** instance, which is the starting point for the application. It configures various parameters and adds services and dependencies to the application's dependency injection container.

The code configures the app's fonts by adding a custom font file to the app's font collection. It then adds the **MauiBlazorWebView** service to the application's services collection, which ensures that Blazor web components are integrated into the Maui application.

The code adds developer tools for the Blazor web representation in debug mode, which can be very useful in the development and testing phases of the application life cycle. These tools give developers additional insight and visibility into the application's inner workings, allowing them to identify and diagnose problems and bugs.

Finally, the code adds services to the application's collection of injections, including the **ITransactionStorageService**, **IShareService**, and **IDeviceInfoService**, which are defined in the **BudgetTracker.Shared** project.

- MainPage.xaml

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:BudgetTracker"
             x:Class="BudgetTracker.MainPage"
             BackgroundColor="{DynamicResource PageBackgroundColor}">

    <BlazorWebView x:Name="blazorWebView" HostPage="wwwroot/index.html">
      <BlazorWebView.RootComponents>
        <RootComponent Selector="#app" ComponentType="{x:Type local:Main}" />
      </BlazorWebView.RootComponents>
    </BlazorWebView>

</ContentPage>
```

The file **MainPage.xaml** is a XAML file defining the main page of the MAUI application, which is the content page hosting the **BlazorWebView** component. This component allows the Blazor application to run inside the MAUI application, as discussed earlier in the theoretical part of the thesis.

BlazorWebView is configured with the **HostPage** property, which defines the location of the HTML file of the starting point of the Blazor application. The **RootComponents** property represents the main Blazor component displayed in the web view.

- Index.html

```
<body>
    ...

    <div id="app">
    </div>

    <div id="blazor-error-ui">
        An unhandled error has occurred.
        <a href="" class="reload">Reload</a>
        <a class="dismiss">✕</a>
    </div>

    ...
</body>
```

This code represents the body section of the **index.html** file, which is the starting point of the Blazor WebAssembly application.

The div with the id "app" is the root element where the Blazor WebAssembly application is displayed. It is used to hold all of the application's user interface components.

The div with the id "blazor-error-ui" is used to display an error message if raw errors occur during the execution of the application. The message will contain an option to reload the application or skip the message.

- Main.razor

```
<Router AppAssembly="@typeof(Main).Assembly" AdditionalAssemblies="new[]
{typeof(BudgetTracker.Shared.MainLayout).Assembly}">
    <Found Context="routeData">
        <RouteView RouteData="@routeData"
DefaultLayout="@typeof(MainLayout)" />
        <FocusOnNavigate RouteData="@routeData" Selector="h1" />
    </Found>
    <NotFound>
        <LayoutView Layout="@typeof(BudgetTracker.Shared.MainLayout)">
            <p role="alert">Sorry, there's nothing at this address.</p>
        </LayoutView>
    </NotFound>
</Router>
```

This file is responsible for defining the top-level routing and layout of the application. It sets up the **Router** component, which is responsible for handling URL-based navigation to different pages on the client side.

The **MainLayout** component is defined in the **BudgetTracker.Shared** project, which is a separate assembly from the Main component defined in the **BudgetTracker.Native** project. By specifying **typeof(MainLayout).Assembly** in the **AdditionalAssemblies** parameter, the **Router** component can find and display the **MainLayout** component in the **BudgetTracker.Shared** assembly.

The **Found** block represents the content that should be displayed when the requested URL matches a particular route and uses the **RouteView** component to display relevant content based on the route data. It also includes a **FocusOnNavigate** component that sets the focus to the first h1 element when navigating the page.

The **NotFound** block represents content displayed when the requested URL does not match any defined route and uses the **LayoutView** component with a simple error message.

6.2.4 Platforms

The Platforms folder has subfolders for each specific platform that contain different resources and manifests. In the context of Budget tracker, I only use Android and Windows platforms because I don't have a machine with iOS installed, which is necessary for iOS development, so I just removed the folders for the unused platforms. Still, you can see the default Platforms folder structure in Figure 19.

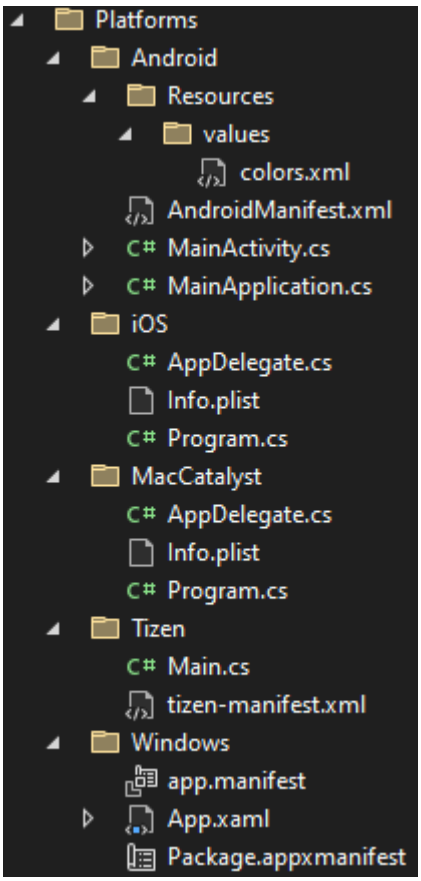

Figure 19 default Platforms folder structure

- Android

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
   <application android:allowBackup="true" android:icon="@mipmap/appicon"
android:roundIcon="@mipmap/appicon_round"
android:supportsRtl="true"></application>
   <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
   <uses-permission android:name="android.permission.INTERNET" />
   <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
</manifest>
```

AndroidManifest.xml is a file that contains essential information about an Android app. It acts as a blueprint for the app's entire structure and functionality and is used by the Android operating system to determine how to launch, interact with, and manage the app.

colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
      <color name="colorPrimary">#757575</color>
      <color name="colorPrimaryDark">#212121</color>
      <color name="colorAccent">#757575</color>
</resources>
```

In the **colors.xml** file, it is possible to define colors for some native values, such as the color of the notification panel at the top of the screen of an Android device.

- Windows

For Windows, Visual Studio 2022 has a menu interface, so you can access all the settings through this window, as shown in Figure 20.

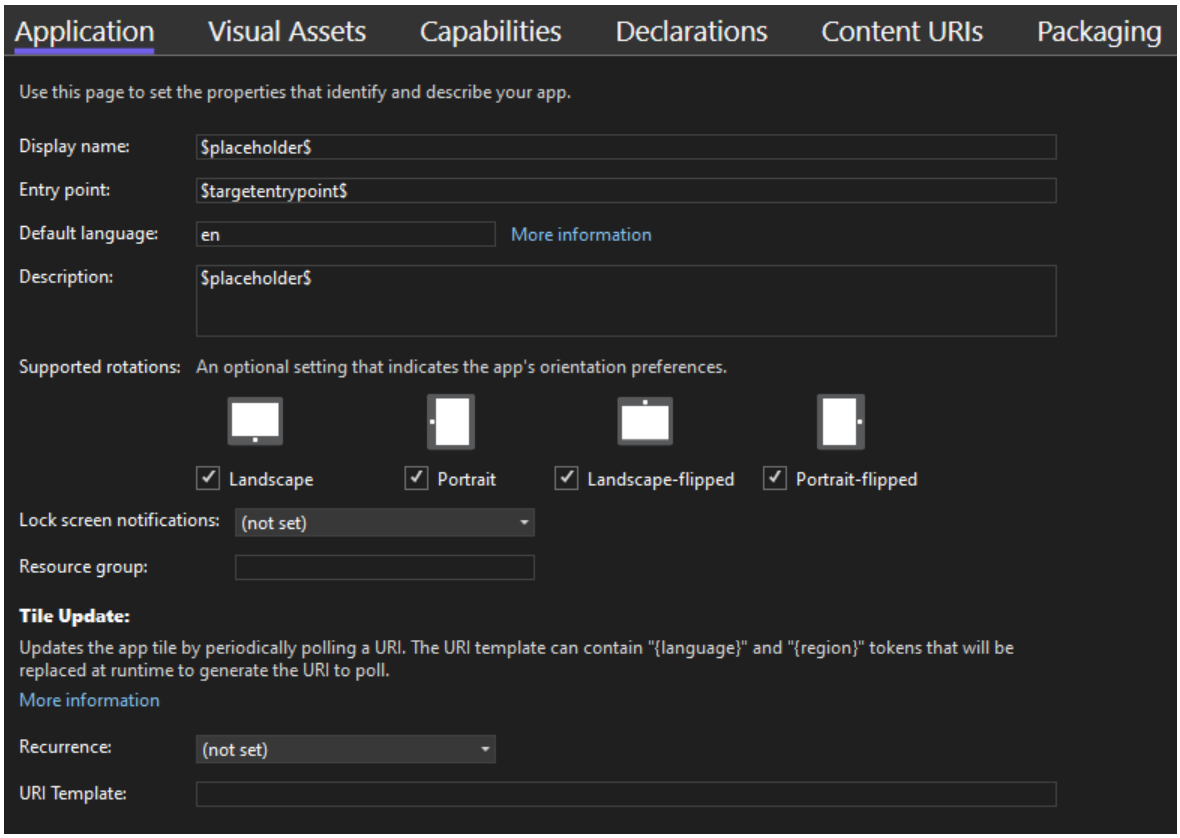

Figure 20 Package.appxmanifest window

6.2.5 Services

- ShareService.cs

The **ShareService.cs** is an implementation of the **IShareService** interface from the **BudgetTracker.Shared** project.

ShareService.cs

```
public async Task ShareDataAsync()
{
    var file = await GetTransactionsFile();

    await Share.Default.RequestAsync(new ShareFileRequest
    {
        File = file
    });
}
```

Here you can see, that MAUI provides a **Share** class, whis allows to share any file by simply calling the **RequestAsync** method. It is also possible to share text and multiple files. In Figures 21 and 22 you can see the share menu on Windows and Android.

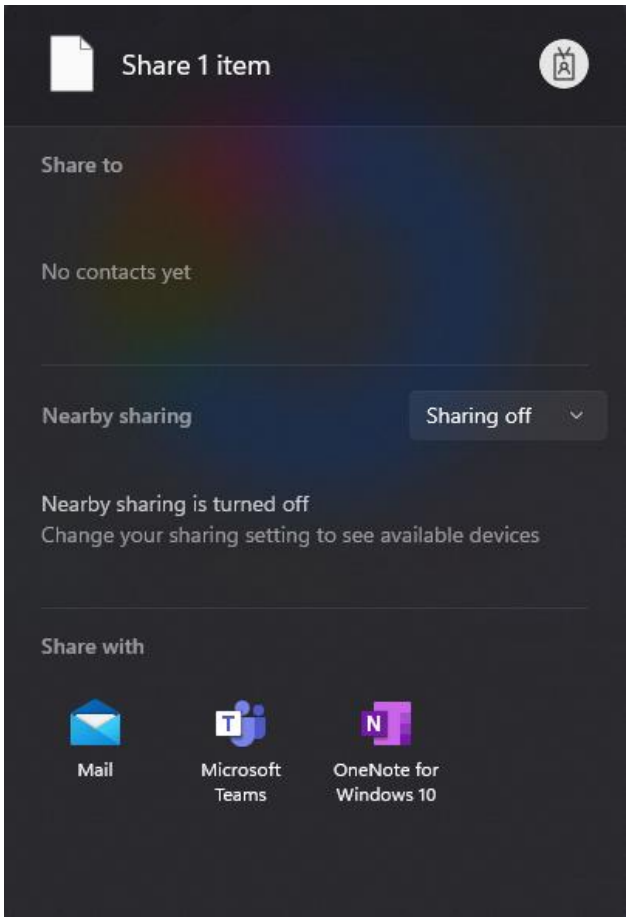

Figure 21 share menu on Windows

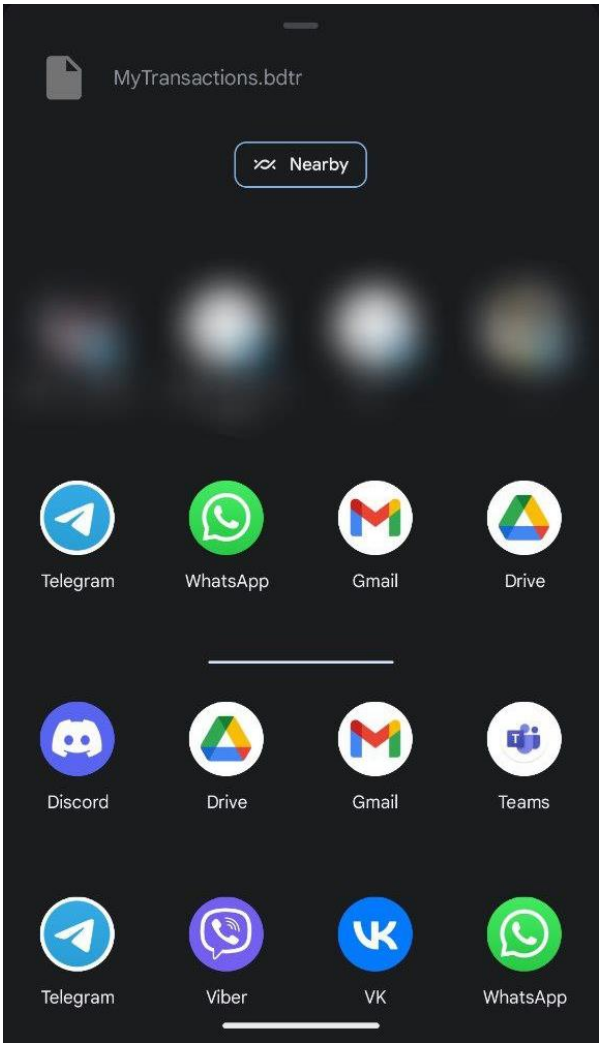

Figure 22 share menu on Android

ShareService.cs

```
private async Task<ShareFile> GetTransactionsFile()
{
    var directory = StorageService.GetDirectory();

    if (!File.Exists(directory))
        File.Create(directory).Close();

    var fileText = await File.ReadAllTextAsync(directory);

    string cacheFilePath = Path.Combine(FileSystem.Current.CacheDirectory,
"MyTransactions.bdtr");

    await File.WriteAllTextAsync(cacheFilePath, fileText);

    return new ShareFile(cacheFilePath, "application/bdtr");
}
```

Here is an example of using the **FileSystem** class, which provides easy access to specific directories without worrying about different systems with different storage architectures.

ShareService.cs

```
public async Task ImportDataAsync()
{
    PermissionStatus status = await
Permissions.RequestAsync<Permissions.StorageRead>();

    if (status == PermissionStatus.Denied)
        return;

    var result = await FilePicker.Default.PickAsync(new PickOptions()
    {
        FileTypes = CustomFileType
    });

    if (result == null || result.FileName.Split('.').LastOrDefault() !=
"bdtr")
        return;

    using var streamReader = new StreamReader(await result.OpenReadAsync());

    var data = await streamReader.ReadToEndAsync();

    var transactions = ParseData(data);

    if (transactions == null)
        return;

    await StorageService.SaveTransactionsAsync(transactions);
}
```

Here you can see an example of using the **Permissions** class. It allows a developer to request permissions from the user, such as storage reading, and validate them.

The next MAUI class here is the **FilePicker**, which is used to select a file with any custom options, such as a custom file type, which can be declared as follows:

ShareService.cs

```
private readonly FilePickerFileType CustomFileType =
      new(new Dictionary<DevicePlatform, IEnumerable<string>>
{
    { DevicePlatform.Android, new[] { "application/bdtr" } }, // MIME type
    { DevicePlatform.WinUI, new[] { ".bdtr" } }, // file extension
});
```

- TransactionStorageService.cs

This class is an implementation of the **ITransactionStorageService** interface from the **BudgetTracker.Shared** project. For **BudgetTracker.Native** project it uses the external storage of the user's device. All transactions are saved in the **MyTransactions.bdtr** file, which represents a JSON object.

The only thing here specifically from MAUI is the **FileSystem** class, which has already been described before. In this service, it is used to get the direction of the application data folder.

TransactionStorageService.cs

```
private string TransactionsDirection { get; } =
    Path.Combine(FileSystem.Current.AppDataDirectory, "Transactions.bdtr");
```

Other methods just implement the usual CRUD operations (Create, Read, Update, Delete), and there is nothing special about it.

- DeviceInfoService.cs

This class implements the **IDeviceInfoService** interface from the **BudgetTracker.Shared** project.

DeviceInfoService.cs

```
public Task<DeviceType> GetDeviceTypeAsync()
{
    var deviceType = DeviceInfo.Current.Idiom;

    if (deviceType == DeviceIdiom.Phone || deviceType == DeviceIdiom.Tablet)
        return Task.FromResult(DeviceType.NativeMobile);

    else if (deviceType == DeviceIdiom.Desktop)
        return Task.FromResult(DeviceType.NativeDesktop);

    else
        return Task.FromResult(DeviceType.Unknown);
}
```

Here is an example of using the **DeviceInfo** class. This MAUI class allows you to read information about the device on which the application is running. It can provide information such as device platform (Android, Windows, iOS etc.), device idiom (phone, tablet, desktop etc.), and device type (physical or virtual).

6.2.6 Resources

In this folder developers can store application icon, splash screen image, fonts and all application images, as shown in Figure 23.

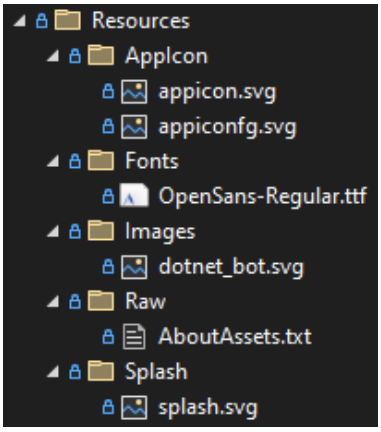

Figure 23 Resources folder structure

Then it is possible to bind the resources in the **.csproj** file:

BudgetTracker.Native.csproj

```
<ItemGroup>
    <!-- App Icon -->
    <MauiIcon Include="Resources\AppIcon\appicon.svg"
ForegroundFile="Resources\AppIcon\appiconfg.svg" Color="#FFFFFF" />

    <!-- Splash Screen -->
    <MauiSplashScreen Include="Resources\Splash\splash.svg" Color="#32333d"
BaseSize="1024,1024" />

    <!-- Images -->
    <MauiImage Include="Resources\Images\*" />
    <MauiImage Update="Resources\Images\dotnet_bot.svg" BaseSize="168,208" />

    <!-- Custom Fonts -->
    <MauiFont Include="Resources\Fonts\*" />

    <!-- Raw Assets (also remove the "Resources\Raw" prefix) -->
    <MauiAsset Include="Resources\Raw\**"
LogicalName="%(RecursiveDir)%(Filename)%(Extension)" />
</ItemGroup>
```

You can see the App Icon on different platforms and the Splash Screen in Figures 25, 26 and 27.

Also in this file you can specify the application name, identifier and version:

```
<!-- Display name -->
<ApplicationTitle>Budget tracker</ApplicationTitle>

<!-- App Identifier -->
<ApplicationId>com.Gaas.budgettracker</ApplicationId>
<ApplicationIdGuid>E3DF3561-60A9-48DB-B558-1402A656330B</ApplicationIdGuid>

<!-- Versions -->
<ApplicationDisplayVersion>1.0</ApplicationDisplayVersion>
<ApplicationVersion>1</ApplicationVersion>
```

The Budget Tracker application icon is shown in Figure 24 and was taken from **svgrepo.com**, where you can find a lot of free SVG images.

Link to the used icon: https://www.svgrepo.com/svg/223080/dollar-money

Figure 24 Budget tracker application icon

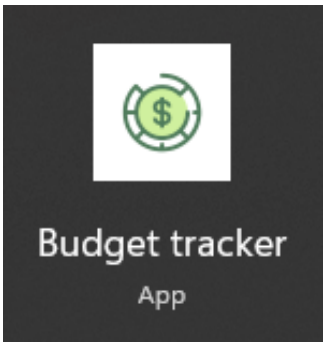

Figure 25 Budget tracker application icon on Windows

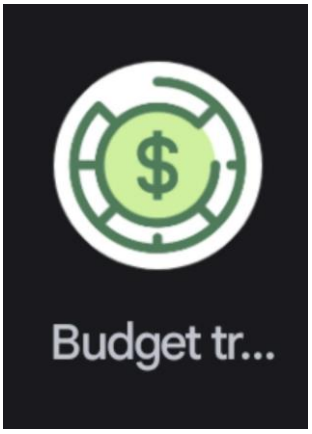

Figure 26 Budget tracker application icon on Android

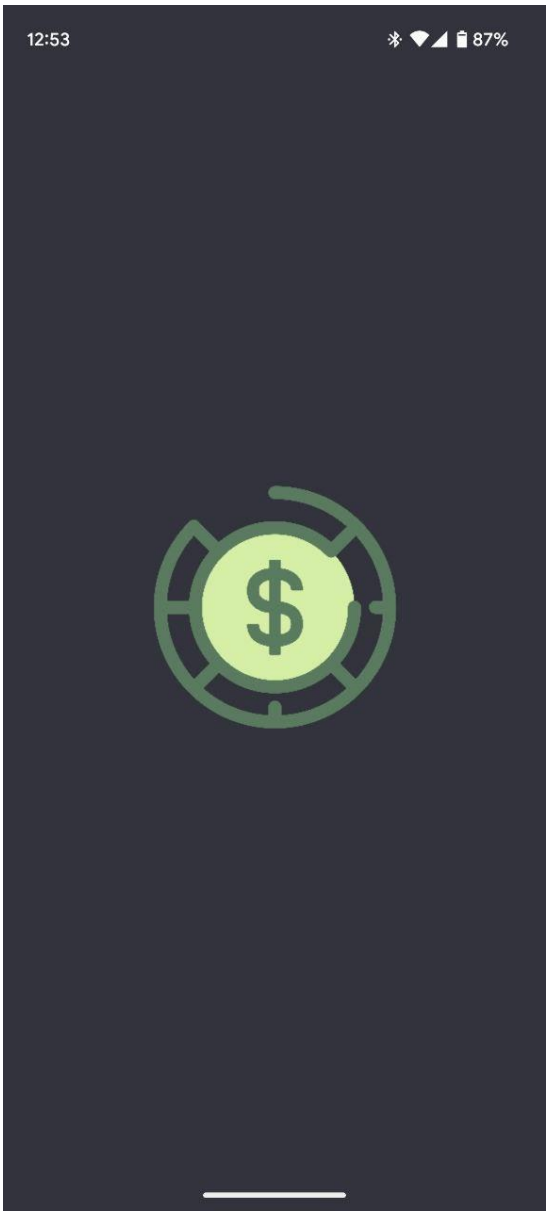

Figure 27 Budget tracker splash screen on Android

6.3 BudgetTracker.Web

6.3.1 Overview

This project is a web version of the Budget Tracker application created using Blazor. It allows you to host and access the application on a web server through a web browser, providing cross-platform support for desktop and mobile devices. The project can also be used to create a PWA, which allows you to install and use the app as a standalone application on mobile devices. To communicate with the UI, it implements the service interfaces defined in **BudgetTracker.Shared.**

The main goal of this project is to show that if you already have some UI written in Blazor, you can quickly build two types of applications without difficulty. For example, you have a web application written in Blazor, and one day you want it to be natively installed on your phone, and this is where MAUI comes into play. It also works in reverse, and you can easily create a web application from an existing MAUI Blazor project. You can see the project structure in Figure 28.

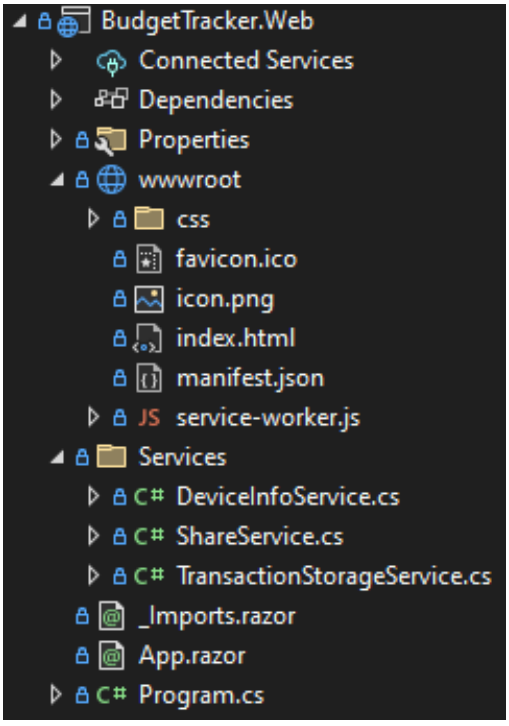

Figure 28 BudgetTracker.Web project structure

6.3.2 Stack

The project uses the following technologies:

- .NET 6
- Blazor WebAssembly
- BudgetTracker.Shared as project reference
- BlazorDownloadFile as NuGet package

  That library allows to download files from the browser without any JavaScript library or dependency [49].

- Blazored.LocalStorage as NuGet package

  That library provides access to the browsers local storage APIs for Blazor applications. An additional benefit of using this library is that it will handle serializing and deserializing values when saving or retrieving them [50].

6.3.3 Entry point

- Program.cs

```
var builder = WebAssemblyHostBuilder.CreateDefault(args);

builder.RootComponents.Add<App>("#app");
builder.RootComponents.Add<HeadOutlet>("head::after");

builder.Services.AddMudServices();

builder.Services.AddScoped(sp => new HttpClient { BaseAddress = new
Uri(builder.HostEnvironment.BaseAddress) });

builder.Services.AddBlazoredLocalStorage();
builder.Services.AddBlazorDownloadFile();

builder.Services.AddScoped<ITransactionStorageService,
TransactionStorageService>();
builder.Services.AddScoped<IShareService, ShareService>();
builder.Services.AddScoped<IDeviceInfoService, DeviceInfoService>();

await builder.Build().RunAsync();
```

This file is responsible for setting up the WebAssembly host and configuring services for the application.

It first creates the default **WebAssemblyHostBuilder** and adds an App component as the root component for the application that will be rendered in the **index.html** page with the identifier "app". It also adds a HeadOutlet component to render header elements.

It then adds all the services the application needs, such as MudServices to use the MudBlazor user interface library, BlazoredLocalStorage to handle the browser's local storage, and BlazorDownloadFile to download files. It also registers custom services for the application, such as **ITransactionStorageService**, **IShareService**, and **IDeviceInfoService**, defined in the BudgetTracker.Shared project.

Finally, it builds the WebAssembly host and runs the application asynchronously.

- App.razor

This file is the same as the **Main.razor** file from the **BudgetTracker.Native** project, so here is nothing new to discuss.

6.3.4 Services

- ShareService.cs

Since it is impossible to use the same share logic on the web, I have decided to allow users to download the application data from the browser.

```
public async Task DownloadDataAsync()
{
    var keyName = StorageService.GetDirectory();

    if (!await LocalStorageService.ContainKeyAsync(keyName))
        return;

    var fileText = await LocalStorageService.GetItemAsStringAsync(keyName);

    await FileDownloadService.DownloadFileFromText("MyTransactions.bdtr",
    fileText, System.Text.Encoding.Unicode, "application/bdtr");
}
```

Here you can see how the **BlazorDownloadFile** and **Blazored.LocalStorage** packages work. The developer can easily retrieve and install data in the browser's local storage and then download any file using the needed method.

- DeviceInfoService.cs

The implementation just returns that the device type is Web.

```
public class DeviceInfoService : IDeviceInfoService
{
    public Task<DeviceType> GetDeviceTypeAsync()
        => Task.FromResult(DeviceType.Web);
}
```

- TransactionStorageService.cs

The only difference between the implementation here and the implementation in the **BudgetTracker.Native** project is that here it is not possible to use external storage of the user's device, so here we can only use the local storage of the browser like this:

```
var fileText = await LocalStorageService.GetItemAsStringAsync(KeyName);
```

for reading the data from storage.

```
await LocalStorageService.SetItemAsStringAsync(KeyName, serializedData);
```

for writing the data to storage.

6.3.5 Views

The following figures show how the application looks on the web and as a PWA.

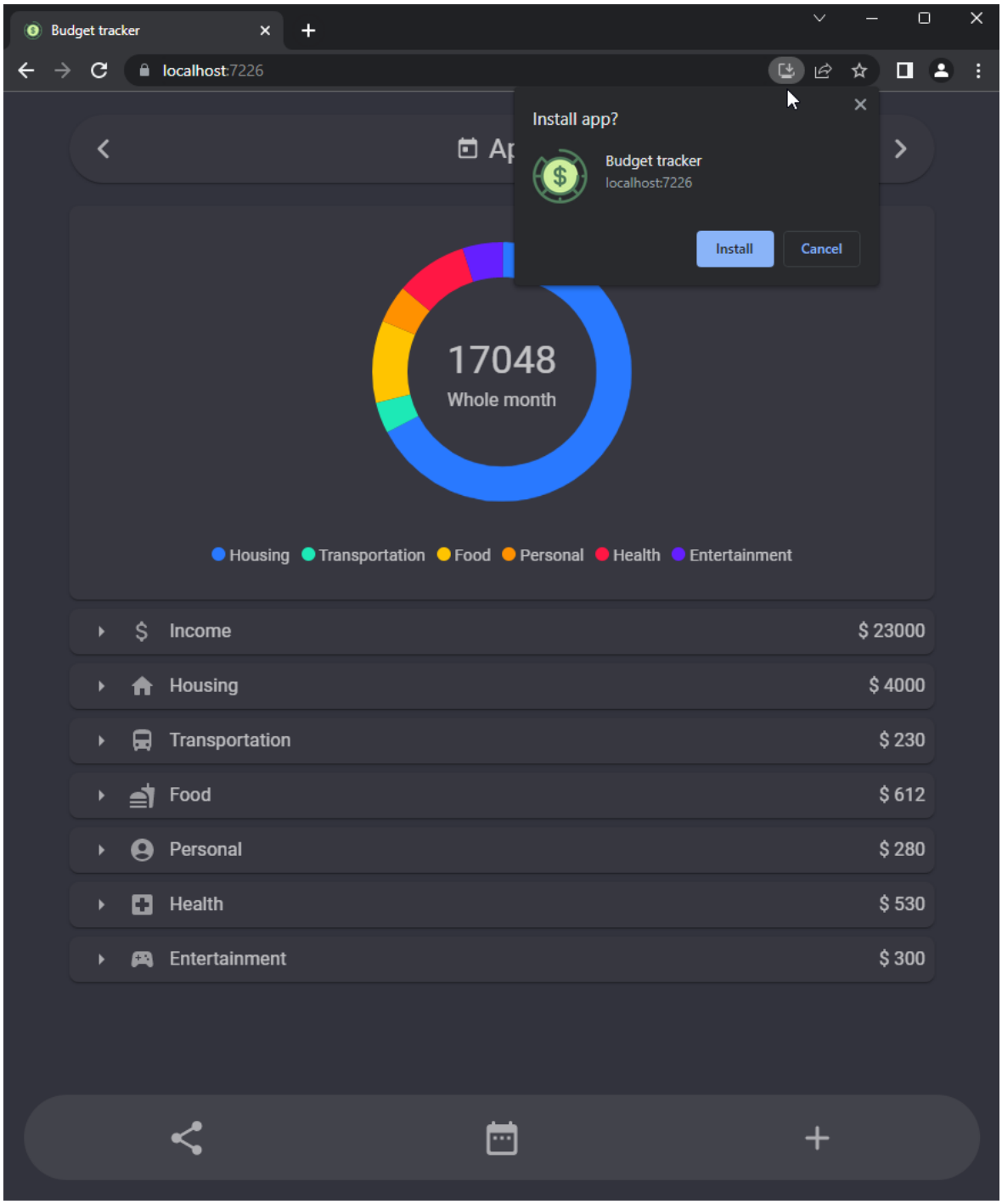

Figure 29 Main window on web

As shown in Figure 29, you can install the application as a PWA, which you can see in Figure 30. And this application will have a normal app icon, as shown in Figure 31.

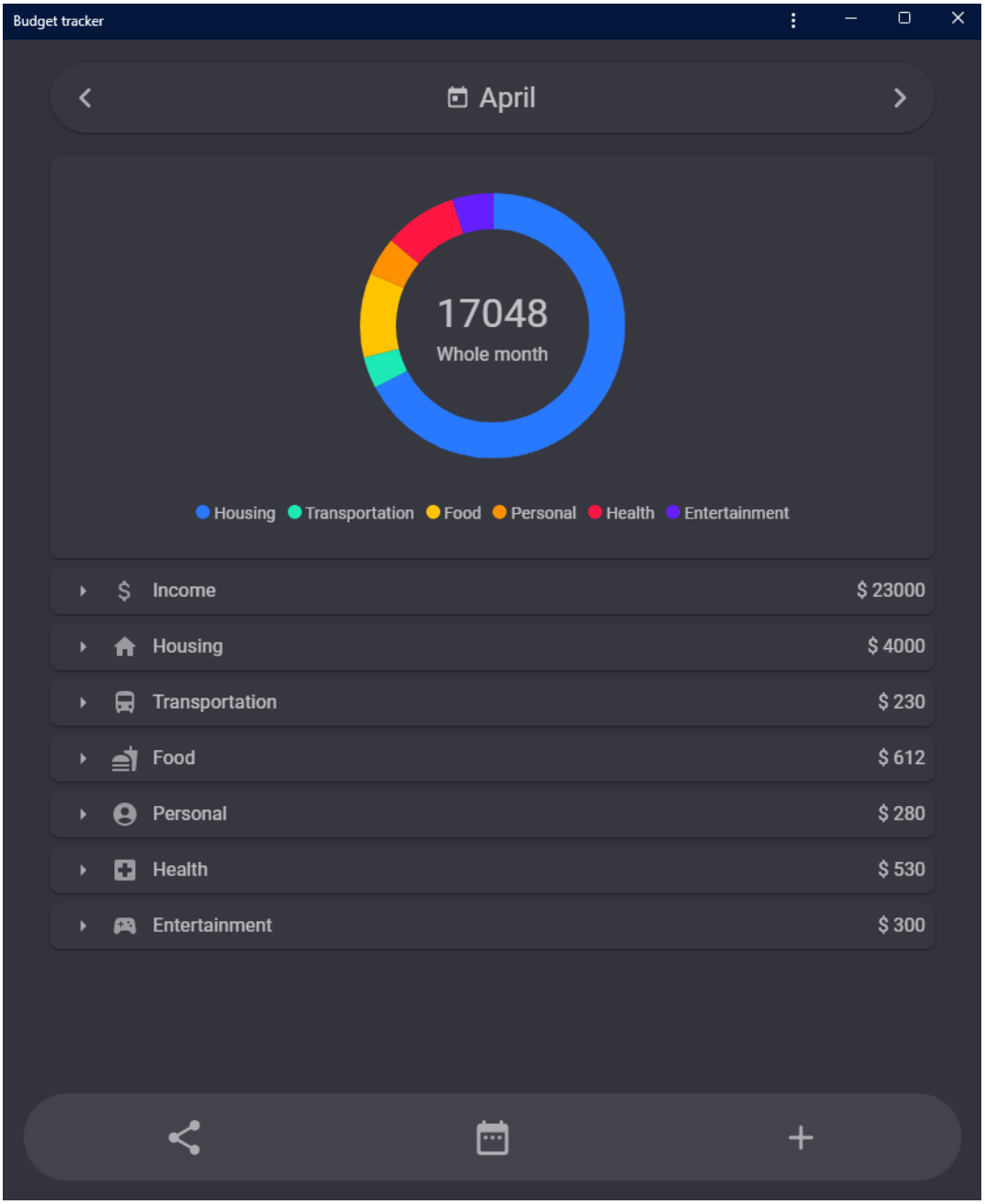

Figure 30 Main window as PWA application

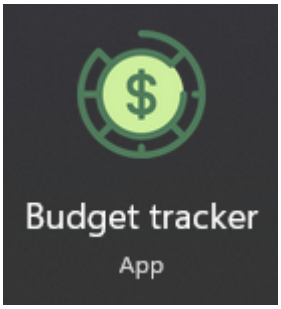

Figure 31 Budget tracker PWA icon on Windows

7 SUMMARIZING

7.1 Achieved results

The project has been successfully completed with the delivery of a fully functional and standalone application capable of performing all its tasks. It has been developed using modern development methodologies and best practices to ensure its stability, efficiency, and ease of maintenance, ensuring that any future changes or upgrades can be easily implemented without disrupting the existing system's functionality. The application is now ready to be deployed and used by its target users, providing a reliable and effective tool for tracking their budget.

7.2 Further development possibilities

Although the application is functional and appropriate, there is always room for improvement, such as:

- Additional features

  The current version of the app includes the essential features needed to track and manage budgets, but there may be additional features that users would find helpful. For example, providing multiple account support or the ability to create custom categories or generate reports based on spending patterns may increase the app's usefulness.

- Integration with external services

  The application can be enhanced by integrating with external services, such as banks or financial institutions, to track transactions and update the budget automatically. This would require secure data transfer and API integration.

- Localization

  The application can be translated into several languages to suit users' needs in different regions to increase its user base. This would require additional language resources and adjustment of the user interface.

Thanks to the use of .NET MAUI and Blazor, adding these features and updating the application will not be a problem. The modular architecture and shared code base of the solution structure allow for easy implementation of new functionality through all platforms.

CONCLUSION

In this thesis, various modern application development methodologies have been reviewed. A detailed analysis of their advantages and disadvantages was presented, and the current state of technologies and frameworks for cross-platform application development was reviewed. The focus was on the popular cross-platform framework .NET MAUI, highlighting its work under the hood and the advantages and differences between MAUI and its predecessor Xamarin. Furthermore, we looked at Blazor, an advanced framework that can be used to develop web, cross-platform, and hybrid applications, and analyzed WebAssembly as the key technology behind it.

In the analytical part, the Budget Tracker application using .NET MAUI and Blazor was created to help users manage their finances by allowing them to track their expenses and income. The thesis presented a complete analysis of the application development process, providing insight into the key steps and considerations involved in developing cross-platform applications. The Budget Tracker application demonstrated how cross-platform application development can be used to create applications that meet users' needs on different platforms.

Overall, this thesis has achieved its goal of providing valuable insight into current application development methodologies and technologies, as well as demonstrating how cross-platform application development can be used to create applications that meet the users' needs on different platforms. The achieved results have provided an excellent foundation for further development opportunities in this area.

BIBLIOGRAPHY

[1] What Is a Native App and How It Is Different from Hybrid and Web Apps? [online]. [cit. 2023-03-22]. Available from: https://www.mobileapps.com/blog/what-is-a-native-app

[2] Seven Reasons Why Native App Development is a Better Solution [online]. [cit. 2023-03-22]. Available from: https://appinventiv.com/blog/seven-reasons-native-app-development-better-solution/

[3] Native App Development vs Hybrid and Web App Building [online]. [cit. 2023-03-22]. Available from: https://mlsdev.com/blog/native-app-development-vs-web-and-hybrid-app-development

[4] Explore Advantages and Disadvantages of Native App Development [online]. [cit. 2023-03-22]. Available from: https://www.tigren.com/blog/advantages-and-disadvantages-of-native-app-development/

[5] The Pros and Cons of Native Apps [online]. [cit. 2023-03-22]. Available from: https://clutch.co/app-developers/resources/pros-cons-native-apps

[6] What are the Pros and Cons of Hybrid App Development? [online]. [cit. 2023-03-22]. Available from: https://www.rswebsols.com/tutorials/software-tutorials/pros-cons-hybrid-app-development

[7] Advantages & Disadvantages of Hybrid App Development [online]. [cit. 2023-03-22]. Available from: https://www.appoly.co.uk/2021/01/10/advantages-disadvantages-of-hybrid-app-development/

[8] The biggest advantages and disadvantages of hybrid apps [online]. [cit. 2023-03-22]. Available from: https://zudu.co.uk/blog/hybrid-apps-pros-and-cons/

[9] 11 Advantages of Cross-Platform App Development [online]. [cit. 2023-03-22]. Available from: https://blog.felgo.com/cross-platform-app-development/advantages

[10] Q&A on Cross-Platform App Development: Pros & Cons Revealed [online]. [cit. 2023-03-22]. Available from: https://www.rishabhsoft.com/blog/pros-cons-cross-platform-mobile-app-development

[11] Advantages & Disadvantages Of Developing Cross-Platform Apps [online]. [cit. 2023-03-22]. Available from: https://www.exeideas.com/2022/01/advantages-disadvantages-of-cross-platform-apps.html

[12] The Six Most Popular Cross-Platform App Development Frameworks [online]. [cit. 2023-03-22]. Available from: https://kotlinlang.org/docs/cross-platform-frameworks.html#react-native

[13] What is .NET MAUI? [online]. [cit. 2023-03-22]. Available from: https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui?view=net-maui-7.0

[14] What is Kotlin Multiplatform Mobile (KMM)? [online]. [cit. 2023-03-22]. Available from: https://kotlinlang.org/lp/mobile/

[15] Flutter [online]. [cit. 2023-03-22]. Available from: https://flutter.dev/

[16] React Native [online]. [cit. 2023-03-22]. Available from: https://reactnative.dev/

[17] Ionic [online]. [cit. 2023-03-22]. Available from: https://ionic.io/framework

[18] Cross-platform mobile frameworks used by software developers worldwide from 2020 to 2021 [online]. [cit. 2023-03-22]. Available from: https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/

[19] What is an Operating System? [online]. [cit. 2023-03-22]. Available from: https://www.howtogeek.com/361572/what-is-an-operating-system/

[20] What is Operating System? Explain Types of OS, Features and Examples [online]. [cit. 2023-03-22]. Available from: https://www.guru99.com/operating-system-tutorial.html

[21] Operating System Market Share Worldwide [online]. [cit. 2023-03-22]. Available from: https://gs.statcounter.com/os-market-share#yearly-2010-2023

[22] What is Windows? [online]. [cit. 2023-03-22]. Available from: https://www.computerhope.com/jargon/w/windows.htm

[23] macOS [online]. [cit. 2023-03-22]. Available from: https://en.wikipedia.org/wiki/MacOS

[24] What is macOS? [online]. [cit. 2023-03-22]. Available from: https://www.techtarget.com/whatis/definition/Mac-OS

[25] What is Linux? [online]. [cit. 2023-03-22]. Available from: https://www.linux.com/what-is-linux/

[26] What is Linux? [online]. [cit. 2023-03-22]. Available from: https://www.redhat.com/en/topics/linux/what-is-linux

[27] What is Android? [online]. [cit. 2023-03-22]. Available from: https://www.android.com/what-is-android/

[28] Android [online]. [cit. 2023-03-22]. Available from: https://en.wikipedia.org/wiki/Android_(operating_system)

[29] iOS [online]. [cit. 2023-03-22]. Available from: https://www.techopedia.com/definition/25206/ios

[30] iOS [online]. [cit. 2023-03-22]. Available from: https://en.wikipedia.org/wiki/IOS

[31] Benefits of using .NET MAUI [online]. [cit. 2023-03-22]. Available from: https://www.slideshare.net/narolainfotechnarola/benefits-of-using-net-maui

[32] What Is .NET MAUI And Should You Use It? [online]. [cit. 2023-03-22]. Available from: https://xam.com.au/what-is-net-maui-and-should-you-use-it/

[33] Why is .NET MAUI the best tool for cross-platform mobile development? [online]. [cit. 2023-03-22]. Available from: https://luismts.com/dotnet-maui-best-tool-for-mobile-development/

[34] .NET MAUI and the future of Xamarin [online]. [cit. 2023-03-22]. Available from: https://innowise-group.com/blog/net-maui-vs-xamarin/

[35] .NET MAUI Vs Xamarin.Forms: A Comparison of Cross-Platform Frameworks [online]. [cit. 2023-03-22]. Available from: https://grialkit.com/blog/learn-the-key-differences-between-net-maui-vs-xamarin-forms-for-cross-platform-mobile-and-desktop-development

[36] Xamarin Versus .NET MAUI [online]. [cit. 2023-03-22]. Available from: https://www.syncfusion.com/blogs/post/xamarin-versus-net-maui.aspx

[37] Blazor Hybrid Web Apps with .NET MAUI [online]. [cit. 2023-03-22]. Available from: https://www.codemag.com/Article/2111092/Blazor-Hybrid-Web-Apps-with-.NET-MAUI

[38] Sharing Code with Blazor & .NET MAUI [online]. [cit. 2023-03-22]. Available from: https://www.telerik.com/blogs/sharing-code-blazor-dotnet-maui

[39] ASP.NET Core Blazor [online]. [cit. 2023-03-22]. Available from: https://learn.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-3.1

[40] WebAssembly FAQ [online]. [cit. 2023-03-22]. Available from: https://webassembly.org/docs/faq/

[41] What Is WebAssembly, and Should You Use It? [online]. [cit. 2023-03-22]. Available from: https://www.howtogeek.com/devops/what-is-webassembly-and-should-you-use-it/

[42] What Is WebAssembly and Why Do You Need It? [online]. [cit. 2023-03-22]. Available from: https://thenewstack.io/what-is-webassembly-and-why-do-you-need-it/

[43] Why WebAssembly? Top 11 Wasm benefits [online]. [cit. 2023-03-22]. Available from: https://www.theserverside.com/tip/Why-WebAssembly-Top-Wasm-benefits

[44] Webassembly vs. JavaScript: How Do They Compare [online]. [cit. 2023-03-22]. Available from: https://snipcart.com/blog/webassembly-vs-javascript

[45] What are disadvantages of WebAssembly compared to current HTML/JS? [online]. [cit. 2023-03-22]. Available from: https://www.quora.com/What-are-disadvantages-of-WebAssembly-compared-to-current-HTML-JS

[46] MudBlazor [online]. [cit. 2023-04-18]. Available from: https://mudblazor.com/

[47] MudBlazor GitHub [online]. [cit. 2023-04-18]. Available from: https://github.com/MudBlazor/MudBlazor/

[48] MudBlazor Icons [online]. [cit. 2023-04-20]. Available from: https://mudblazor.com/features/icons#icons

[49] BlazorDownloadFile [online]. [cit. 2023-04-25]. Available from: https://github.com/arivera12/BlazorDownloadFile

[50] BlazoredLocalStorage [online]. [cit. 2023-04-25]. Available from: https://github.com/Blazored/LocalStorage

LIST OF ABBREVIATIONS

API - Application Programming Interface

CRUD - Create, Read, Update, Delete

CSS - Cascading Style Sheets

GPU - Graphics Processing Unit

GPS - Global Positioning System

HTML - Hypertext Markup Language

MAUI - Multi-platform App User Interface

PWA – Progressive Web Application

SDK - Software Development Kit

SVG - Scalable Vector Graphics

UI - User Interface

XAML - Extensible Application Markup Language

LIST OF FIGURES

APPENDICES

Appendix P I: CD with the source code of the application.