BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

.NET MAUI MULTIPLATFORM USER INTERFACE FOR KIMAI

MULTIPLATFORMNÍ UŽIVATELSKÉ ROZHRANÍ PRO SYSTÉM KIMAI V .NET MAUI

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. DÁVID ŠPAVOR**
AUTOR PRÁCE

SUPERVISOR **Ing. DANIEL DOLEJŠKA**
VEDOUCÍ PRÁCE

BRNO 2023

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Master's Thesis Assignment

146870

Institut:	Department of Information Systems (UIFS)
Student:	**Špavor Dávid, Bc.**
Programme:	Information Technology and Artificial Intelligence
Specialization:	Software Engineering
Title:	**.NET MAUI Multiplatform User Interface for Kimai**
Category:	User Interfaces
Academic year:	2022/23

Assignment:

1. Study technologies for multiplatform development of user interfaces in general and applications in .NET MAUI as required by the supervisor.
2. Study the Kimai system, its capabilities, and existing UI solutions. Analyze existing user interfaces.
3. Create an interactive application design according to the supervisor's requirements, focusing on the identified application shortcomings from the UI/UX perspective from point 2.
4. Implement a multiplatform application solution according to your design from point 3, respecting the supervisor's requirements. Deploy the applications on at least two platforms supported by the MAUI framework.
5. Properly test the developed application in a broader audience of users according to the supervisor's requirements. Evaluate the benefits and shortcomings of the developed solution. Discuss further possible development and extension of the implemented application.

Literature:

- KRUG, Steve, 2014. *Don't make me think, revisited: a common sense approach to Web usability*. Third edition. Berkeley, Calif.: New Riders. ISBN 978-0-321-96551-6. TK5105.888 .K78 2014
- RUBIN, Jeffrey and CHISNELL, Dana, 2008. *Handbook of usability testing: how to plan, design, and conduct effective tests*. 2nd ed. Indianapolis, IN: Wiley Pub. ISBN 978-0-470-18548-3. QA76.9.U83 R82 2008
- STONE, Debbie, JARRETT, Caroline, WOODROFFE, Mark and MINOCHA, Shailey, 2005. *User Interface Design and Evaluation*. . Burlington: Elsevier Science. ISBN 978-0-08-052032-2.
- according to the supervisor's instructions

Requirements for the semestral defence:
Points 1, 2 and 3.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Dolejška Daniel, Ing.**
Head of Department:	Kolář Dušan, doc. Dr. Ing.
Beginning of work:	1.11.2022
Submission deadline:	17.5.2023
Approval date:	26.10.2022

Abstract

The main goal of this thesis is to create a multiplatform time-tracking application in .NET MAUI, which will use the open-source tool Kimai. The purpose of this application is to make the use of the Kimai tool easier and more efficient. The other goal is to provide a set of development approaches within .NET MAUI that can be applied to other applications developed on this framework. In the beginning, the problematics of multiplatform development, user interface design, and the Kimai system were discussed. Subsequently, an interactive design of the application was created, which was tested and iteratively improved. Architectural approaches within .NET MAUI were compared and then the application was implemented. In the end, the application was published and tested on a real audience. The result is a multiplatform open-source application called MAUI for Kimai, which is fully supported on Android and Windows platforms and experimentally functional on Mac and iOS platforms. The implementation approaches of this application can be applied to other applications developed in .NET MAUI. This application has been well received by the Kimai community, who appreciated that this application will make it easier and more efficient to use Kimai on desktop and mobile platforms.

Abstrakt

Hlavný cieľ tejto diplomovej práce je vytvoriť multiplatformnú aplikáciu na sledovanie času v .NET MAUI, ktorá bude využívať open-source nástroj Kimai. Táto aplikácia ma za účel zjednodušiť a zefektívniť používanie nástroja Kimai. Ďalším cieľom je poskytnúť súbor vývojových prístupov v rámci .NET MAUI, ktoré môžu byť aplikované na iné aplikácie vyvíjané na tomto frameworku.

Na začiatku bola diskutovaná problematika multiplatformného vývoja, dizajnu užívateľských rozhraní a systému Kimai. Následne bol vytvorený interaktívny návrh aplikácie, ktorý bol testovaný a iteratívne vylepšovaný. Boli porovnané architektonické prístupy v rámci .NET MAUI a následne bola aplikácia implementovaná. Na konci bola aplikácia zverejnená a testovaná na reálnom publiku.

Výsledkom je multiplatformná open-source aplikácia s názvom MAUI for Kimai, ktorá je plne podporovaná na Android a Windows platformách a experimentálne funkčná na Mac a iOS platformách. Implementačné prístupy tejto aplikácie môžu byť aplikované na iné aplikácie vyvijané v .NET MAUI. Táto aplikácia získala ohlas Kimai komunity, ktorá ocenila, že vďaka tejto aplikácii bude možné využívať nástroj Kimai jednoduchšie a efektívnejšie na desktopovej a mobilnej platforme.

Keywords

.NET, MAUI, Kimai, user interface, multi-platform, Android, iOS, Mac, Windows

Kľúčové slová

.NET, MAUI, Kimai, užívateľské rozhranie, multi-platforné, Android, iOS, Mac, Windows

Reference

ŠPAVOR, Dávid. *.NET MAUI Multiplatform User Interface for Kimai.* Brno, 2023. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Daniel Dolejška

Rozšírený abstrakt

Multiplatformný vývoj je v dnešnej dobe veľmi žiadaný. Často, keď firma vytvára softvérový produkt, musí sa zamýšľať, na ktoré platformy svoj produkt nasadí. Toto rozhodnutie môže celkovo ovplyvniť úspech a použiteľnosť daného softvérového produktu. Zväčša sa jedná o mobilné (iOS, Android) a desktopové (Windows, Mac) platformy. Existujú dva hlavné spôsoby, ako je možné vytvárať multiplatformné aplikácie. Prvý spôsob je, že daný softvérový tím firmy môže využiť natívne technológie dostupných platforiem. To znamená, že pre každú platformu sa vytvorí samostatná aplikácia, ktorej vývoj sa musí synchronizovať naprieč všetkými platformami. Síce vyvinuté aplikácie postavené na natívnych technológiách sú rýchle a bohaté na funkcionalitu, avšak udržovateľnosť naprieč všetkými platformami môže byť veľmi náročná a drahá. Druhý spôsob, ktorý môže softvérový tím firmy využiť, je použitie frameworku na multiplatformný vývoj. To znamená, že sa vyvinie jedna aplikácia, ktorej dostupnosť na rôzne platformy bude zabezpečená daným multiplatformným frameworkom. Tento prístup rieši problém udržovatelnosti produktu naprieč platformami, kde vývoj je rýchlejší a stačí udržovať iba jeden kód. Na druhú stranu, schopnosti týchto aplikácií sú limitované daným multiplatformným frameworkom, čiže niektoré funkcionality aplikácie môžu byť obmedzené. Preto je dôležité sa pred vývojom dôkladne zamyslieť, ktorú spôsob vývoju daného produktu si vybrať.

V rámci mojej diplomovej práce riešim práve problematiku multiplatformného vývoja aplikácií pomocou multiplatformných frameworkov. Špecificky, v čase písania tohto textu pomocou nového frameworku .NET MAUI. Mám dva hlavné ciele, ktoré chcem dosiahnuť v rámci tejto diplomovej práce. Prvý cieľ je dôkladne naštudovať framework .NET MAUI a poskytnúť súbor prístupov na vývoj aplikácií na tomto frameworku. Špecificky sa zamerať na architektúru aplikácie, rozdelenie pohľadov aplikácie medzi platformami, použité technológie, návrhové vzory či na komunikáciu medzi komponentami aplikácie. Druhý hlavný cieľ je vytvoriť multiplatformnú aplikáciu na sledovanie času, ktorá bude komunikovať so systémom Kimai. Systém Kimai je open-source online aplikácia, ktorá slúži na zaznamenávanie odpracovaných hodín, sledovanie práce v tíme či na vytváranie faktúr. Následne vytvorená aplikácia by mala byť dostupná Kimai komunite a výskumnej skupine UIFS na VUT FIT na zjednodušenie a zefektívnejšie práce so systémom Kimai.

Na začiatku som naštudoval teóriu ohľadom multiplatformného vývoja, dizajnu aplikácií a systému Kimai. Porovnal som natívny, multiplatformný a webový vývoj a poskytol som prehľad populárnych multiplatformných frameworkov. Následne som podrobne naštudoval framework .NET MAUI a systém Kimai. Na základe naštudovaných informácií som vytvoril prvý interaktívny návrh aplikácie v nástroji Figma. Tento návrh bol testovaný pomocou metódy `usability testing`. Bol vytvorený testovací plán, pomocou ktorého som testoval interaktívny návrh aplikácie na reálnych užívateľoch. Testovanie bolo zamerané na použiteľnosť aplikácie, identifikovanie hlavných vlastností aplikácie a jej celkového dizajnu. Testovanie prebehlo v 2 fázach, na základe ktorých vznikol finálny návrh aplikácie. V rámci implementačnej časti som podrobne porovnal architektonické prístupy v rámci .NET MAUI a vybral najviac vhodnú architektúru pre moju aplikáciu. Poskytol som prehľad použitých technológií a knižníc, ktoré uľahčujú vývoj na .NET MAUI. Následne je v tejto časti riešená problematika pripojenia .NET MAUI aplikácie na API systému Kimai, navigácia v rámci aplikácie a ďalšie implementačné detaily. Táto kapitola obsahuje vývoje prístupy (architektúra, navigácia, komunikácia, technológie a návrhové vzory), ktoré môžu byť aplikované na ďalšie aplikácie vyvíjane vo frameworku .NET MAUI. V poslednej časti bola aplikácia vyhodnotená a testovaná na reálnom publiku, kde bola získaná spätná väzba, ktorá bola zakomponovaná do ďalších verzií aplikácie.

Výsledkom diplomovej práce je vytvorená multiplatformná aplikácia s názvom `MAUI for Kimai`, ktorá je dostupná a podporovaná na Android a Windows platformách. Aplikácia je funkčná tiež na iOS a Mac platformách, avšak len experimentálne, pretože počas vývoja som mal obmedzený prístup k Mac vývojovým nástrojom. Aplikácia je open-source v rámci GitHub repozitára, kde je dostupný zdrojový kód, dokumentácia, články ohľadom implementácie a tiež optimalizované inštalačné balíčky. Aplikácia získala ohlas od Kimai komunity a bola zdieľaná na sociálnych sieťach Twitter a Mastodon ako nová, zaujímavá, multiplatformná aplikácia dostupná pre systém Kimai.

.NET MAUI Multiplatform User Interface for Kimai

Declaration

I hereby declare that this Diploma's thesis was prepared as an original work by the author Bc. Dávid Špavor under the supervision of Ing. Daniel Dolejška. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Dávid Špavor
May 15, 2023

Acknowledgements

I would like to express my gratitude to my supervisor, Mr. Ing. Daniel Dolejška for his guidance and quick and helpful responses to my questions.

Contents

List of Figures

Chapter 1

Introduction

Time-tracking is a very important concept within the software development process, mostly for an effective division of company resources. There are many applications out there for time-tracking purposes, but often they are paid and some of them are complicated.

Kimai time-tracking software was developed in 2006. It is open-source, free software created as a web application. It brings to developers' teams the ability to manage their workflow and accurately bill the working times. Kimai has been updated over the years and various mobile applications have been developed. However, a lot of them are out of date and some of them are also paid.

The Microsoft company released Visual Studio 2022 17.3 along with the .NET Multi-platform App UI (.NET MAUI) on 9 August 2022. .NET MAUI is a framework for the development of multiplatform applications from a single shared codebase. This means that code that is working on the mobile platform can also work on the desktop platform. In my opinion, this presents wonderful opportunities for developers when they need to create the same application for both different platforms.

My main goal of this thesis is to create a multi-platform time-tracking application, which should improve a time-tracking use case within Kimai software. In addition, I would like to provide a set of approaches for multi-platform development in .NET MAUI. Specifically, these approaches should describe how code between platforms should be divided, which design patterns should be used, and how programmers should proceed during development.

The thesis is logically divided into multiple chapters. The first chapter describes the problematics of multiplatform development and existing frameworks. The approaches for the design of the user interfaces are also discussed, and in the end, the Kimai time-tracking software is introduced. The second chapter deals with the design of the user interface of the application, which was improved in multiple iterations due to a method of usability testing. The third chapter discusses implementation, especially my reasoning behind a chosen application architecture, an overview of the technologies used, and the implementation details of my solution. The deployment, usage, and evaluation of the application are described in the last chapter with possible extensions in the future.

Chapter 2

Multiplatform development and system Kimai

This chapter describes multiple ways to develop software applications and their user interfaces. The main focus is on multi-platform and native software development with a target on cross-platform frameworks. Multiple frameworks are discussed, but the main focus is on .NET MAUI and the technology behind it. Moreover, the theory behind the design of user interfaces is introduced with the theory of usability testing. In the end, the Kimai system is introduced with an analysis of existing Kimai applications.

2.1 Native and cross-platform development

Nowadays, developing a software application requires many important decisions to be made. One of them is, on which platform application should be created. This is crucial because it determines how large the audience for the application is and, in the end, its overall success. This leads to different ways of developing software products. In other words, there are three main approaches to developing software applications:

- Native development
- Cross-platform development
- Web development

Native application development

The term native application development refers to the building of an application exclusively for a single platform. The application is built with programming languages and tools specific to that platform [1].

Let us take as an example the Android platform. Java or Kotlin programming languages can be used to develop Android applications. Furthermore, for iOS applications, Swift and Objective-C programming languages are available. Thus, building native applications requires a high level of specialized knowledge for each platform. This leads to development using the OS guidelines, which brings the user experience of developed software as close to the particular platform experience.

Advantages

- **Performance** – native software is singularly familiar with the OS it runs on, then these applications will naturally have more flexibility on their native platforms. Therefore, the result is fast response times and processing speed.
- **Security** – native applications have security protocols that work through all layers of an OS.
- **Broad functionality** – programmers have access to the original API (Application Programming Interface) provided by the platform. Moreover, there is greater compatibility when it comes to certain natural UI and UX elements of the platform on which the software is built [5].

Disadvantages

- **Time-consuming** – when a developer decides to create software using a platform-specific language, it can be an exhaustive task if an application is planned to launch on every available platform.
- **Cost** – since native development is time-consuming, the more time it takes to develop software, the less time it takes to promote the application and collect profits. If multiple platforms are chosen, there is a need for multiple teams for different platform.
- **Accessibility** – many software development companies do not have the resources to build an application for multiple platforms, or at least they cannot release it on all platforms at the same time. Thus, potential users are left disappointed, and the company misses out on business opportunities [5].

Cross-platform development

Cross-platform development is a process of creating applications that work on multiple platforms. Cross-platform development frameworks (CPDF) wrap the system APIs for different platforms into their own APIs, which are consistent and can be accessed through a single programming language. Using CPDF, developers can implement cross-platform applications by writing code only once. This is particularly useful for mobile development since there are two distinct popular platforms, namely iOS and Android. For example, both the iOS and Android APIs define their own button view. CPDFs define their own button implementations, which are then converted to actual native IOS or Android buttons [20].

Advantages

- **One source code** – single code base for all platforms where the code is reusable.
- **Faster development** – development is less time consuming because of one source code for each platform.
- **Faster time to market** – publishing built applications for multiple platform app stores simultaneously.
- **Easier maintenance** – only a single code base needs to be maintained and tested.
- **Less costly** – there is no need for so many resources [16][2].

Disadvantages

- **Limited functionality** – there can be difficulties in accessing some native features.
- **Worse performance** – there is a need for an additional abstraction layer, and the rendering process makes the cross-platform application slower than its native counterpart.
- **Limited user experience** – applications cannot take advantage of native UX components [1].

Web-based application development

A web application is an interactive software built using web development technologies that users can access from their browsers. Typically, web applications have typical front-end and back-end web development technologies [6]. The front-end is the presentation layer of the application. It describes how the application looks and how the user interacts with the application. On the other hand, the back-end development is focused on functionality and data storage. Back-end technology is often deployed on the web server, which serves requests of connected front-end clients.

One of the different types of web software is the Progressive Web App (PWA). Until recently, the web platform lagged behind in mobile-centered innovation in terms of being able to compete with native or cross-platform apps. PWA is a new set of standards advocated by the Google Web Fundamentals group that seeks to bridge that gap by introducing features such as offline support, background synchronization, push notifications, and home-screen installation to the web [3]. In other words, PWA offers more native-like functionality than the basic web application.

Advantages

- **High portability** – web applications are entirely multiplatform. They can be accessed through any device with an internet connection and a web browser.
- **Easier development and maintenance** – there is only one codebase with which to work and a limited set of technologies.
- **More convenience** – applications do not need to be installed or downloaded (unlike PWA, but are really small compared to the native approach) [6].

Disadvantages

- **Limited functionality** – web applications do not have the same capacity to effectively collaborate with the hardware and operating system of a specific device.
- **Availability** – web applications generally require an Internet connection to be accessed [6].

2.2 Cross-platform development frameworks

This section contains an overview of some of the most popular multiplatform frameworks. For each platform advantages and disadvantages are presented for easier comparison.

React Native

React Native is an application development framework where standard web technologies are used, or, in some cases, something similar to standard web technologies, to build application. This means HTML, JavaScript, and CSS. React Native is based on Facebook's React framework, a popular web development framework. The critical difference between the two is that plain old React targets web browsers, whereas React Native typically does not. Android and iOS platforms are currently supported. The created mobile applications perform more like native applications than typical web applications, because the core technology behind them is native [11]. However, when a developer wants to implement some platform-specific functionality, there is a need for platform-specific code.

Some of a popular mobile applications developed with React Native technology are Discord, Facebook, Skype, Instagram, and Airbnb.

Advantages

- Simultaneous development for multiple platforms.
- Excellent set of developer tools (hot reload, integrated development environments (IDEs).
- Load JavaScript-only changes in applications in stores without going through the approval process (both for Apple and Google).
- Possibility to write native code and call it from JavaScript.

Disadvantages

- Difficulties with debugging due to multiple layers of abstractions.
- There can be a length of time during which React Native does not support a new version of Android or IOS (the underlying APIs need to be adjusted to changes).
- In some cases, native code is still needed.
- Possibility to write native code and call it from JavaScript [11].

Xamarin

Xamarin is an open-source platform from Microsoft, which extends .NET ecosystem with tools and libraries specifically for building applications for Android, IOS, watchOS, macOS, and Windows. At its basic, it is a way to use the same language and technology across iOS and Android, allowing to reuse of large amounts of code and third-party libraries across two very different mobile platforms. Best practices around Xamarin are focused on keeping this amount of code sharing as large as possible. That is why the popular Model-View-ViewModel (MVVM) design pattern is used [15].

There are multiple applications created with Xamarin, for example: CA Mobile, The World Bank, Alaska Airlines, Microsoft Azure and UPS Mobile.

Advantages

- Compatibility with MVC and MVVM design patterns.
- .NET development ecosystem (C#, Visual Studio, Nugets).
- Open-source and free.
- Full hardware support.

Disadvantages

- In some cases, the need for the use of platform-specific code.
- Problems may occur when a complex UI is required.
- Large application size.

Flutter

Flutter is Google's portable UI framework for building modern, native and reactive applications for iOS and Android. Google is also working on Flutter desktop embedding and Flutter for the Web (Hummingbird) and embedded devices (Raspberry Pi, home, automotive, and more). Flutter uses Dart, a modern object-oriented language that compiles to native ARM code and production-ready JavaScript code. Dart is used to create user interfaces, removing the need to use separate languages like Markup or visual designers. Flutter is a declarative language, which means it builds the UI to reflect the state of the application. When the state (data) changes, the UI is redrawn, and Flutter constructs a new instance of the widget. Applications made in Flutter are built from a single codebase, use the graphics processing unit (GPU), and can access specific iOS and Android APIs (like GPS location, image library) by communicating through platform channels [18].

Advantages

- Fast and dynamic code writing with the support of hot reload.
- Support for ready-made widgets, which helps to quickly build a user interface.
- Access to native features of the platform, which developers can access with Objective-C, Swift, or Java.

Disadvantages

- Scarcity of third-party libraries, which means that if there is no needed functionality, developers must develop it by themselves.
- The size of the application is quite substantial.
- Low adaptation of the Dart programming language [25].

2.3 .NET MAUI framework

Since .NET MAUI is a relatively new technology (the first Release Candidate of .NET MAUI was released on April 12, 2022), at the time of writing this thesis, there are limited resources of .NET MAUI literature. For these reasons, a substantial part of this section is derived from the MAUI documentation [1] created by Microsoft.

Multiplatform app UI (MAUI) is a new open-source cross-platform framework for creating mobile and desktop applications with C# and XAML. Using .NET MAUI, applications can be developed, which run on Android, iOS, macOS, and Windows from a single shared codebase. The main aim is to reuse as much code as possible between the different platforms. .NET MAUI unifies different platform APIs into a single API that allows using approach „write-once, use-anywhere“ while providing access to native code of each platform. It contains platform-specific libraries for Android, iOS, Mac and WinUI 3.

All these libraries have access to the .NET base class library (BCL). The BCL is the base for all general .NET class libraries and provides code execution environments. BCL abstracts the underlying platform environment away from the written code. For Android, iOS, and MacOS, the environment is implemented by Mono [2]. On Windows, .NET CoreCLR [3] is used to provide the execution environment. The architecture of .NET MAUI is represented in Figure 2.1.

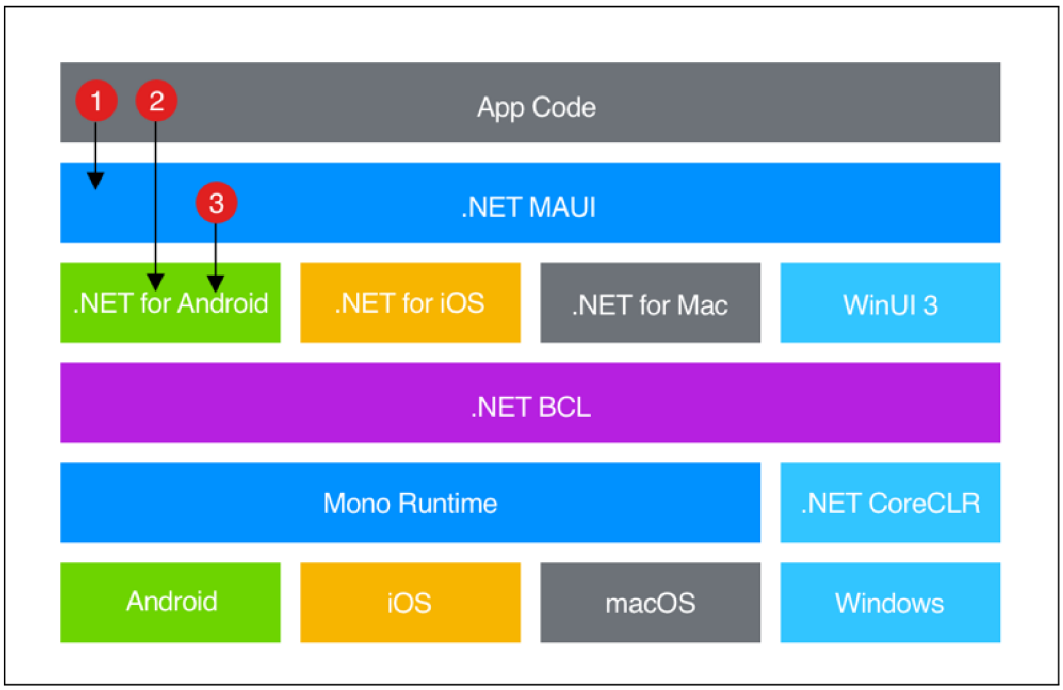


Figure 2.1: Representation of architecture of .NET MAUI, where BCL abstracts underlying platform-specific libraries depending on platform environment. Taken from official .NET MAUI documentation[5].

[1]MAUI documentation: `https://learn.microsoft.com/en-us/dotnet/maui/`
[2]Mono - cross-platform .NET framework : `https://www.mono-project.com/`
[3]CLR (Common Language Runtime): `https://learn.microsoft.com/en-us/dotnet/standard/clr`
[5]Taken from: `https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui?view=net-maui-7.0`

The code that developers write interacts primarily with the .NET MAUI API (1). .NET MAUI API then directly consumes the APIs of the native platforms (3). Moreover, the developer can directly access .NET abstractions of platform APIs (2), if required [10].

.NET MAUI is considered an evolution of Xamarin.Forms. Accordingly, there are many similarities with Xamarin:

- XAML and C# is used to develop applications.
- MVVM design pattern is applied with data binding.
- Usage of improved Xamarin controls, layouts, Shell, gestures, templates, and cross-platform APIs device features.

However, there are also many important differences. One of the main differences is the support for Windows. Xamarin supports UWP [6], while MAUI supports WinUI [7] [21]. Other important differences are concluded in Table 2.1.

	Xamarin	**MAUI**
Architecture	.NET Framework	.NET 6, 7
Project structure	Separate projects for each platform.	Single project structure
Targeted platforms versions	Android 4.4 (API 19) or higher, iOS 9 or higher, UWP	Android 5.0 (API 21) or higher, IOS 10 or higher, WinUI
Hot Reload	No	Yes
Resources maintenance	Resource files maintained separately for each platform	Resources can be maintained in a single location

Table 2.1: Comparison of main properties of Xamarin and MAUI frameworks [21].

.NET MAUI also brings possibilities to develop hybrid applications. That means the application is running on mobile, desktop and web using a Blazor Hybrid pattern. Blazor Hybrid combines web technologies (HTML, CSS, and sometimes JavaScript) with native in .NET MAUI Blazor. Sometimes, web applications need access to the native capabilities of the device and this approach enables it. However, for mobile devices, to develop applications with MAUI Blazor, the support for WebView is needed and therefore, the native-like feeling of the applications is lost.

[6]UWP (Universal Windows Platform): `https://learn.microsoft.com/en-us/windows/uwp/get-started/universal-application-platform-guide`

[7]WinUI (Windows UI Library): `https://learn.microsoft.com/en-us/windows/apps/winui/`

Application life-cycle

.NET MAUI applications generally have four execution states:

- not running,
- running,
- deactivated,
- and stopped.

These cross-platform lifecycle events are raised in the `Window` class based on application transitions. Transitions are shown in Figure 2.2 with the lifecycle described below.

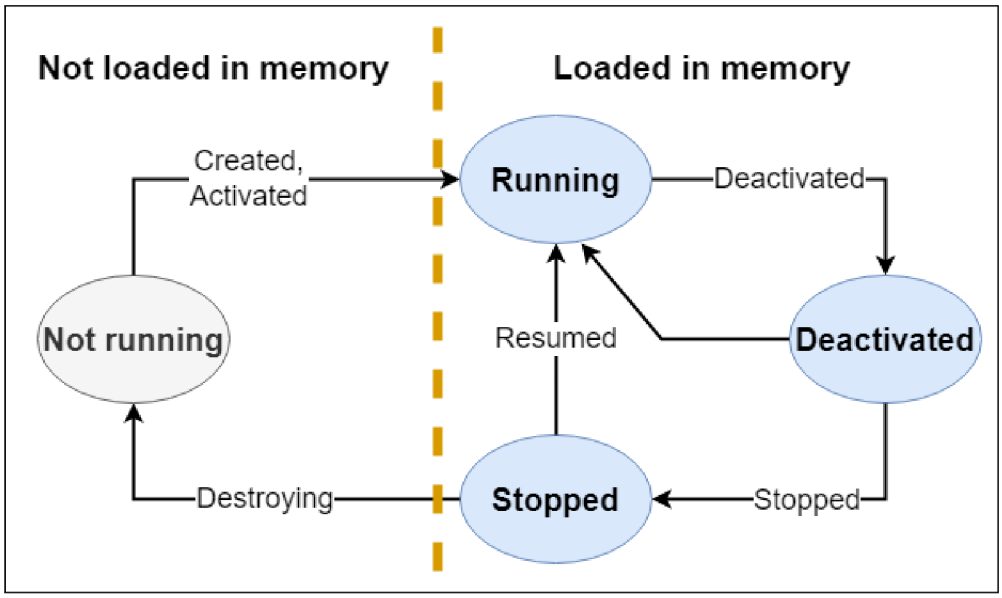


Figure 2.2: MAUI application lifecycle states and events, because of which transitions between states occur.

The execution state of the MAUI application depends on usage history and on events raised by .NET MAUI, which provides notifications. The gray oval of the state `not running` indicates that the application is not loaded into memory. Blue ovals indicate that the application is already loaded into the device's memory.

At the beginning, the application has not yet been started, therefore, it is considered as `not running`. When an application is started, `Created` and `Activated` events are raised and the application is considered as `Running`. When different applications gain focus, the `Deactivated` event is raised and the state changes to `Deactivated`. If the user switches to a different application or returns to the device home screen, the `Deactivated` and `Stopped` events are raised and the application is considered `Stopped`. When the user returns to the application, the `Resumed` event is raised, and the application is again `Running`. Additionally, an application can be terminated by a user or by device software due to resource restriction. Thus, a `Destroying` event occurs and the application is `Not running` again [9].

Behaviors and styles

The functionality of the user interface controls can be added with .NET MAUI behaviors. Due to the behaviors, there is no need to subclass controls for the purpose of adding functionality. In other words, there is no need for inheritance of control properties, all functionality is implemented in a behavior class and attached to the control as if it were part of the control itself. Behaviors enable us to write code-behind code in classes, which can be later reused across multiple controls of different applications.

Two different types of behavior are supported:

- Static behaviors, which are implemented in the `static` class with one or more attached properties.
- Classes deriving from `Behavior` or `Behavior<T>` class, where `T` is the type of control to which the behavior should be applied [7].

The MAUI application can be styled using the `Style` class to group a collection of property values into one object that can then be applied to multiple visual elements. This helps to reduce repetitive markup and allows the appearance of the application to be more easily changed. Styles can inherit from other styles to reduce duplication and enable reuse. This is achieved by setting `Style.BasedOn` on an existing `Style` [8]. Styles can be applied to the UI components by referencing specified `x:Key` value. Example style usage with inheritance is shown in Figure 2.3.

```
<Style x:Key="BaseLabel" TargetType="Label">
    <Setter Property="FontFamily" Value="OpenSansRegular" />
    <Setter Property="TextColor"
    Value="{AppThemeBinding
    Light={StaticResource AccentDark},
    Dark={StaticResource AccentLight}}" />
</Style>

 <Style
    x:Key="SmallLabel"
    BasedOn="{StaticResource BaseLabel}"
    TargetType="Label">
    <Setter Property="FontSize" Value="12" />
</Style>
```

Figure 2.3: Example of .NET MAUI `Style` with inheritance, when `BaseLabel` is defined, from which `SmallLabel` inherits properties specified by `BasedOn` setting.

Styles can also be dynamic, which means that they can be changed on application runtime. The `DynamicResource` markup extension is used instead of `StaticResource` markup extension.

XAML

XAML (eXtensible Application Markup Language) is and XML-based declarative language for creating trees of .NET objects developed by Microsoft. XAML provides a convenient view of constructing user interfaces for .NET applications. It uses `data bindings` to present and interact with data. UI elements can be bound to different kinds of data sources in the form of a .NET object.

XAML is used in multiple Microsoft frameworks: UWP, WPF [8] or MAUI [4]. The basic examination of the XAML syntax is shown in Figure 2.4. This example is used to examine the relationship between XAML and .NET.

```
<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="XamlProj.Window1"
    Title="Main Window">

    <Grid>
        <Ellipse Fill="LightBlue" />
            <TextBlock>
               Name: <TextBlock Text="{Binding Name}" />
            </TextBlock>
        </Grid>
</Window>
```

Figure 2.4: Concrete example of XAML code, which shows a basic structure of XAML with namespaces, classes, properties, and children.

Namespaces

XAML relies on XML namespaces to determine the meaning of elements. Many class names are ambiguous, therefore, the corresponding framework needs to know to what .NET type each control corresponds to. The namespace system is used for disambiguation.

To define a relationship between an XML namespace and one or more .NET namespaces, `XmlnsDefinitionAttribute` is applied to the assembly that contains the types that should be accessible in XAML. The attribute can be applied several times to add multiple.NET namespaces to a single XML namespace.

When a project is built (MAUI, WPF), the namespace definition attributes are looked up by the XAML compiler on all libraries that the project uses [4]. Two namespaces that are used in the example 2.4 are shown in Figure 2.5.

```
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
```

Figure 2.5: Two namespaces are contained in the XAML example in Figure 2.4.

[8] WPF - Windows Presentation Foundation: `https://learn.microsoft.com/en-us/visualstudio/get-started/csharp/tutorial-wpf?view=vs-2022`

Generating classes

Figure 2.4 has an `x:Class="XamlProj.Window1"` attribute on the root element `Window`. The `x:` prefix is the standard XML shorthand to indicate that this particular attribute is in the namespace specified by the `xmlns:x` attribute. The attribute `x:Class` is a signal to the XAML compiler that it should generate a class definition based on this XAML file. It also determines the name of the generated class and will derive from the type of the root element. In this case, it is the `Window1` class [4].

Properties

Attributes, which do not have a namespace qualifier usually correspond to the properties of the .NET object to which the XAML element refers. In this case, the `Title` attribute in the `Window` element indicates that when an instance of the generated `XamlProj.Window1` class is constructed, the `Title` property of `MainWindow` is set [4].

The markup extensions

Type converters and property elements are used to initialize most properties to constant values or fixed structures. But sometimes there is a need to set a property to the value of some particular static property, the value of which is unknown at compile time. Thus, the markup extension class is used. Markup extension decides at run-time how to set a value of property [4]. An example of the usage of the markup extension is shown in Figure 2.6.

```
1 <ImageButton
2     Command="{Binding StartCommand}"
3     IsVisible="{Binding IsTimetrackingActive,
4     Converter={StaticResource InvertedBoolConverter}}"
5     Source="play_button.png">
```

Figure 2.6: Example of the usage of the `StaticResource` markup extension, which use `InvertedBoolConverter` converter for changing visibility of the UI element.

Children

The `Window` element contains more nested elements. The default XML namespace specified by the `xmlns` attribute is in scope for all XML files, so these nested elements are also in the XML namespace. Therefore, the children's elements correspond to the classes, which are contained within defined namespaces. The interpretation of nested elements is defined by the parent element. The parent element decides how the child elements are rendered [4].

2.4 Design of user interfaces

Designing a good user interface is hard, especially for developers with little or no experience. In addition, designing the user interface is a really important part of software development because it can influence potential users of the application. If the UI is poorly designed, it can discourage users to use the implemented application and, in consequence, they move to the competition. This section describes things that should be considered when designing an application. Moreover, the importance of UI and UX is discussed, and important UI patterns are described.

There are several things that need to be considered when a developer wants to design an application:

- **Consistency** – implemented application should work and look like other applications on the same platform. When the user is used to some functions on a specific platform, this function in the implemented application should work as the user expects.
- **User experience** – the application should be easy to use and intuitive. The user should be able to gain knowledge of the use of the app without any prior training. This closely relates to the consistency principle. If the application has a consistent user interface across platforms, it should be easier for the user to learn how to use the application.
- **Flow** – when designing functions of applications, the developer must consider what steps must be taken to achieve a specific goal. The flow of actions to achieve specific functions should be easy, and important UI controls should be placed within reach. For example, if a camera application is developed, the options for editing or sharpening a photo should be on the same screen where view of the taken photo is, not buried in a menu that involves multiple steps to navigate.
- **Good looks** – the application should be modern, where images are well drawn and fit the device size, the text is readable and the colors are consistent.
- **Accessibility** – the developer should consider the possibilities of increasing the font size and implementing some visual alerts. A well-designed application should take into account all possible users [15].

Usability and design of user interfaces

Usability plays a very important role in the software development product. Usability means absence of frustration, when users use software product. In other words, a product or a service is truly usable when the user can do what he wants to do and what he expects to be able to do it, without hindrance, hesitation, or questions. To be usable, a product or service should be useful, efficient, effective, satisfying, learnable, and accessible. The substantial part of this section about usability is taken from the book Handbook of Usability Testing by Jeffrey Rubin and Dana Chisnell [14].

The usefulness of a product enables a user to achieve his goals. If achieving a goal is an easy task, the user should have a motivation to use the product again. On the other hand, if a system is easy to use, easy to learn or even satisfying to use but does not achieve the specific goals of an user, it will not be used. Therefore, the flow of actions to achieve a specific goal must be designed in detail during product design. The speed with which the

user achieves his goal accurately and completely is called efficiency. Often it is a measure of time. In addition, effectiveness refers to the extent to which the product behaves in the way that users expect it to. It defines how easily users can achieve what they want. Effectiveness is usually measured quantitatively with the error rate. In addition, part of effectiveness is learnability, which defines ability of user to operate the system at some level.

When users are reading sites or using some user interface, they don't want to think. Based on the author Steve Krug, who has written the book: Don't Make Me Think! A common sense approach to web usability, this is the first law of usability. The user interface should be self-evident, obvious, and self-explanatory. For example, buttons. It should be obvious that the button is clickable. However, if the developer chooses the wrong UI control, it can confuse the user who has to think about it. It should be evident that this is a clickable control. All these small things can add to the poor usability of the designed user interface. However, not everything can be made self-evident. At least, it should be self-explanatory. The developer can create an explanatory page for the problem, but it should not be long or overwhelming. As a result, the overall experience of using designed user interface should be much better.

People do not read pages. They scan them. They are looking if the page contains something important, and if they find it, then only after that they read it. And this can be applied to any kind of user interface. It is important to create a clear visual hierarchy. The more important something is, the more prominent it is. For instance, the most important headings are either larger, bolder, in a distinctive color, set off by more white space. Similarly, things that are related logically should also be related visually. Similar things should be grouped together and visualized in a similar style. In addition, it is important to show what is part of what. Visual elements that are connected to each other should be nested and should create visual hierarchy. An example of a nested visual hierarchy of headlines is shown in Figure 2.7. For that, we can use some conventions of UI.

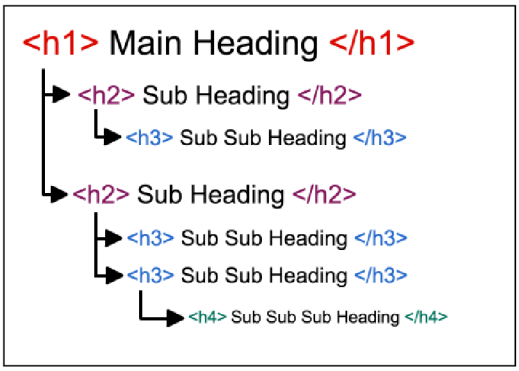


Figure 2.7: Example of visual hierarchy of headlines, which then can be used to dived pages to logically divided sections. Taken from Nomensa article[10]about web accessibility.

Another important thing in a good user experience is the presence of persistent navigation. Good persistent navigation contains:

- site or product identification,
- a way to access the home page,
- a way to search,

[10]Taken from: `https://www.nomensa.com/blog/2017/how-structure-headings-web-accessibility`

- sections,
- and utilities.

Almost every page of applications should have navigation available. However, there are some exceptions. For example, a home page, which can have a different purpose and can be a starting point of navigation, or forms, where persistent navigation can be a distraction to user. For these pages, it is useful to have a minimal version of navigation, with a site ID, a link to home page, and fundamental utilities.

The sections are the top-level hierarchy. They are links to the lower level of the page. It is important to show which section or subsection is currently selected. Good practice is to use *I'm here* indicator, which is a way to show the exact location of the user within the page. Furthermore, the utilities are links to the important elements of the site or application, which are not really part of the content hierarchy. For example, utilities are links to the Help page, the Log-in page, or the About Us page. The possibility of searching things across the page is really important because searching using keywords can often be faster than navigating through an application.

Usability testing

The term usability testing is a technique used to evaluate a product or a system. It is a process in which testing participants who are representative of the target audience evaluate the product to the degree to which a product meets specific usability criteria. The goal of usability testing is to improve the profitability of products, where design decisions are made from data collected from representative users, and design issues are minimized. The design of the product is influenced by the collected data, which can eliminate design problems and user frustration. The usability testing process contains the following parts:

- develop the test plan,
- set up a testing environment,
- find and select participants,
- prepare test materials,
- conduct the test sessions,
- debrief the participant and observers,
- analyze data and observations,
- report findings and recommendations [14].

2.5 Time-tracking system Kimai

Kimai is an open-source time tracking software. It is an online tool which tracks work time and prints out a summary of activities on demand: yearly, monthly, by customer, by project, etc. Kimai is a browser-based application with a responsive user interface [17].

The main goal of Kimai is to track working hours. There is a possibility to add a billing and thus creating invoices. Using a time-tracking software brings multiple benefits to freelancers, teams or even companies. Some of benefits are:

- **Increase profitability** – accurately tracking time and monitoring project budget and resources.
- **Bill and pay accurately** – exact working hours available.
- **Set clear objectives** – each member is focused on a specific task, which is time-tracked.
- **Increase productivity** – within the time tracking period developers stay on task.
- **Better project management** – because of time tracking and utilized resources, the project manager can better prioritize future projects and team assets.
- **Transparency** – giving transparency into developers work to keep them accountable.
- **Better communication of progress** [12].

Brief history and current status of development

The first version of Kimai was released in 2006. It was developed by a team around Torsten Höltge. It had a responsive design, which was not widely spread at this time. Due to personal time limitation of the main developer, the project should be shut down, however Kevin Papst[11] stepped in as the new maintainer and with the help of the open source community, he was able to keep it alive.

Kimai 1 had fundamental features such as the following:

- administration of users, projects, and customers,
- time-tracking of tasks with time-sheets,
- generating invoices,
- calculating rates and fees,
- exporting possibilities,
- and plugins extensions.

Kimai 1 required at least MySQL 4.3 or higher and a PHP version between 5.5 and 7.3. In the meantime, the PHP community made great strides forward, while Kimai was still based on old versions. There were also major drawbacks to the developed architecture, such as the limited extension capabilities of the project and the problematic database design. Also, driven by a community request, the idea of rewriting Kimai was born.

[11] Kevin Papst Github: `https://github.com/kevinpapst`

The new version of the PHP framework Symfony 4.0 with a new front-end stack was released in 2017 and subsequently, the development of Kimai 2 has begun, which was released later in 2018 [17].

The current stable version of Kimai 2 is 1.3 which is available on the Github repository[12] with more than 2200 stars. Developers are working on the next major release of version 2.0, where a new user interface is in development, security is improved with new features (such as two-factor authentication), and much more.

Features

This subsection presents some of the key features of the Kimai 2 time tracking system with examples. Other useful features can be added to Kimai by purchasing plugins from Kimai Market Place [13]. Some of the available plugins are: Audit trail logs, Expense management, KioskMode with Barcode and RFID support, Task planning and Translations. Substantial parts of the text are taken from the official Kimai documentation [14] and from my own analysis of the Kimai system.

Timesheets

A timesheet is recorded time-tracked activity with additional information about duration, customer, project, rate, etc. A table with an overview of all timesheets is available with filtering possibilities. The user can start a new time-tracking activity. When the activity is stopped, it is saved in the time sheet table. There is the possibility to restart activity that has already ended. An example of a timesheet table is shown in Figure 2.8. Multiple formats of duration of activities are supported:

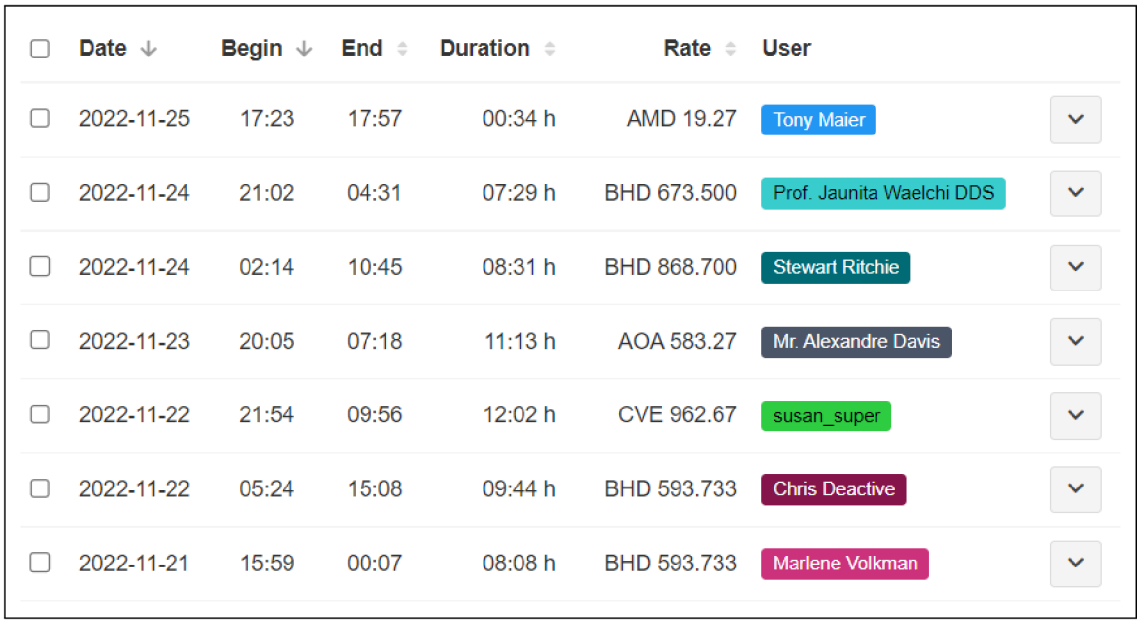
- **Timespan** (2:27 = 2 hours and 27 minutes),
- **Time interval (ISO 8601)** (3h14m = 3 hours and 14 minutes) ,
- **Decimal duration** (1,5 = 1 hour and 30 minutes).

When the record is set to `exported`, modification is blocked to prevent manipulation of exported data. In addition, timesheet records can be set as billable or non-billable. Non-billable records are excluded from invoices and budget calculations. Timesheet records can also be tagged. Tags are used to create arbitrary logical structures that can be later used to filter records in exports, invoices, or reports.

Management of customers, projects, and activities

Customers in Kimai are used to manage projects and activities, which are then used for timesheets. The best practice is to have a customer to track administrative and other internal work. The customer view contains a table with all customers added in the system with the possibility to view details about each customer. The user can set the following attributes to the customer: name, description, address, country, email, timezone, currency, account, VAT-ID.

[12]Github repository of Kimai2: `https://github.com/kevinpapst/kimai2`
[13]Kimai Market Place: `https://www.kimai.org/store/`
[14]Official Kimai documentation: `https://www.kimai.org/documentation/`



Date	Begin	End	Duration	Rate	User
2022-11-25	17:23	17:57	00:34 h	AMD 19.27	Tony Maier
2022-11-24	21:02	04:31	07:29 h	BHD 673.500	Prof. Jaunita Waelchi DDS
2022-11-24	02:14	10:45	08:31 h	BHD 868.700	Stewart Ritchie
2022-11-23	20:05	07:18	11:13 h	AOA 583.27	Mr. Alexandre Davis
2022-11-22	21:54	09:56	12:02 h	CVE 962.67	susan_super
2022-11-22	05:24	15:08	09:44 h	BHD 593.733	Chris Deactive
2022-11-21	15:59	00:07	08:08 h	BHD 593.733	Marlene Volkman

Figure 2.8: Table of recorded timesheets, which contains information about date, duration, rate and user. Timesheet can be edited and restarted with options located within grey drop-down button on right.

Projects are assigned to customers and linked to activities. Projects can also be assigned to teams. An overview of all projects is available. The detailed view of each project contains the following information: name, description, assigned customer, order number, date of start and end of project, and more.

Activities serve to explain the kind of work done on a specific timesheet. Activities have two different types:

- **Global** – activities are not linked to the project and can be used in combination with every project.
- **Project specific** – activities are assigned to projects and are available only if the linked project is active.

Project-specific activities cannot be reassigned to another project. There is only option to change project-specific activity to global activity, however this change is irreversible. An example of the detail view of project with assigned activities is shown in Figure 2.9.

For each customer, project, or activity, additional properties are available to set:

- **Colors** – setting custom color for easier identification,
- **Currency** – setting different currencies for calculating rates and budgets,
- **Billable** – possibility to set billable configuration, which determines whether time records will be billable,
- **Budgets** – to watch the progress of work and stay within contract boundaries,
- **Rates** – for cost calculation.

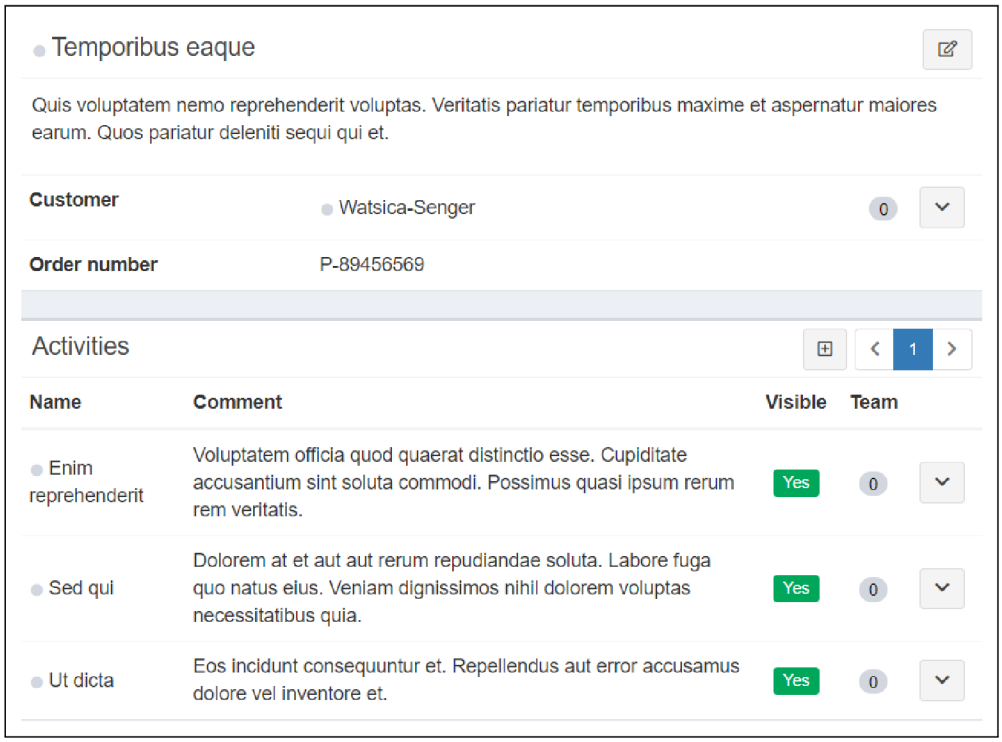


Figure 2.9: Detail view of project with assigned customer, order number and activities.

Kimai contains two different types of budgets. Lifetime budget is the default type, where all records of all times are used to calculate progress of budget usage. On the other hand, monthly budget use records of specific month to calculate progress. Moreover, the configuration of different levels of rates is possible. It starts from the user hourly rate and goes from customers to projects and activities. The rate configuration contains the following settings:

- **User** – the user, the rate which is applied,
- **Rate** – hourly rate to be charged,
- **Internal rate** – internal costs, if the internal rate is not specified, the normal rate is used,
- **Fixed rate** – each time record gets the configured rate value applied, regardless of the duration of the record.

Management of users, teams and roles

A user must be created for using a Kimai system. The self-registration of the user is disabled by default, however, it can be enabled in settings. The user can be part of the team, which can limit or extend the visibility of data. In addition, access to certain features within Kimai is handled by permissions and roles. Users have avatars, which can be either auto-generated or custom created with the image URL. Furthermore, the user can be set as deactivated, which will hide the user from the listings and reports, and disable the possibility to log in to Kimai.

Teams allow users to organize into groups and limit access to customers and projects. The team consists of team leaders and team members. Team leaders have access to all the content and team timesheets. Once a team is assigned to a customer or project, access to this object and its children is limited. Teams are created by a system administrator.

Kimai contains the following user roles in the default permission configuration, the permissions of which differ depending on the role in the team:

- **System-Admin** – management of all content and administration functions,
- **Administrator** – management of all content, but access to system-specific settings is limited,
- **Teamlead** – management of unprivileged contents and all contents assigned to his teams,
- **User** – can see all unprivileged contents and all contents assigned to his teams.

Moreover, Kimai provides a flexible permission system, which is based on user roles and permissions. Figure 2.10 shows an example of permissions for the activity management of an administrator user. These permissions can be turned on and off for these roles. An overview of all existing permissions is available in the Kimai documentation [15].

Name	System-Admin	Administrator	Teamlead	User
Activity (Admin)				
budget_activity	Yes	Yes	No	No
create_activity	Yes	Yes	Yes	No
delete_activity	Yes	Yes	No	No
edit_activity	Yes	Yes	No	No
permissions_activity	Yes	Yes	No	No
time_activity	Yes	Yes	No	No
view_activity	Yes	Yes	No	No

Figure 2.10: Default permission configuration of administrator user for activity with possibility to turn on and off each permission.

Invoices, exports, and reports

The Kimai system provides the creation of invoices from timesheet data in several formats. The user can mix arbitrary characters to create his own invoice format. Some examples of arbitrary characters are:

[15] Overview of existing permissions: `https://www.kimai.org/documentation/permissions.html`.

- `date` – shortcut of date in format ymd (year, month, date),
- `cname` – the customer name,
- `c` – counter for all invoices within the system,
- `cc` – per customer invoice counter,
- `uid` – the internal user ID.

Kimai contains more arbitrary characters, the list of which is available in the Kimai documentation for invoices [16]. Invoices can have multiple states:

- **New** – the invoice that was just created,
- **Waiting for payment** – the invoice that was sent to the customer,
- **Invoice Paid** – the invoice was paid by the customer,
- **Canceled** – the invoice was accidentally created and will not be used.

Invoices should be canceled rather than deleted. The removal of invoices is disabled by default, however, it can be enabled by activating `delete_invoice` permission. Invoices should not be deleted, because it can cause problems with the calculation of the invoice counter. Kimai calculates the invoice numbers from the number of entries in the database. By deleting invoices, the amount of invoices in the database is lowered and the next invoice number might be the same as that already used. Moreover, invoices can be exported to multiple types of documents such as PDF, HTML, DOCX, ODS, XLSX and CSV,

Kimai also contains a separate module for export, which allows one to export filtered timesheet data into several formats. However, there are a couple of differences between the invoice module and the export module. Firstly, invoices can only be created for a dedicated customer, whereas export can be created without selecting a customer. Moreover, invoices do more calculations (for example, calculation of taxes) and they support templates in more formats. Exports can be created only by the user with permission `create_export`. Exports and invoices share the export state, which is used to mark associated timesheet records as processed. The consequence is that these records cannot be edited and are excluded from future invoices and exports.

Kimai provides screens for reporting various information about users activity and projects. For users, working times are reported and can be displayed on a weekly, monthly, or daily basis. Example of yearly reporting is shown in Figure 2.11. Furthermore, reports for projects contain information about resources, summed up times, budget, and time-budget progress bars with links to invoice and export screens.

[16] Kimai documentation for invoices: `https://www.kimai.org/documentation/invoices.html`

	Working hours total	January 2022	February 2022	March 2022	April 2022	May 2022
admin	00:00 h					
Adrian Terry	156:52 h	28:35 h	13:53 h		09:22 h	18:45 h
Alene Weber	161:13 h	41:01 h	34:44 h		01:40 h	09:31 h
Anna Smith	88:06 h				18:03 h	06:53 h
Aurelie Quigley	75:40 h	22:24 h		03:22 h	16:03 h	
Brionna Wolf Jr.	102:46 h	13:19 h	09:21 h	09:59 h	14:41 h	
Camren Goyette	105:57 h	11:54 h	08:27 h	12:38 h		25:06 h

Figure 2.11: Report of total time-tracked working hours of users during the year.

REST API

Kimai provides a detailed implementation of the API with multiple ways to interact with the Kimai server. To access the API, the API password must be configured in the Kimai user profile for each user. Therefore, the API password should be different from the normal user password. Depending on the security level of the user, certain API endpoints are available. When API is called from an external application, two additional headers must be submitted to every API call:

- `X-AUTH-USER` – holds the username or email address,
- `X-AUTH-TOKEN` – holds the configures API password set in the user profile.

Kimai contains API endpoints for:

- **Activity** – management of activities,
- **Default** – for information about Kimai software,
- **Customer** – management of customers,
- **Project** – management of projects,
- **Tag** – management of tags,
- **Team** – management of teams,
- **Timesheet** – management of timesheets,
- **User** – management of users.

The following table 2.2 contains an overview of the API endpoints for activities for the Kimai version 1.28.1. An overview of all API endpoints is available within the Kimai installation at the `/api/doc` URL endpoint in the form of Swagger[17] documentation.

[17]Swagger: `https://swagger.io/`

Request method	Endpoint	Description
PATCH	`/api/activities/{id}/meta`	Sets the value of a meta-field for an existing activity.
GET	`/api/activities`	Returns a collection of activities.
POST	`/api/activities`	Creates a new activity.
GET	`/api/activities/{id}`	Returns one activity.
PATCH	`/api/activities/{id}`	Update an existing activity.
GET	`/api/activities/{id}/rates`	Returns a collection of all rates for one activity.
POST	`/api/activities/{id}/rates`	Adds a new rate to an activity.
DELETE	`/api/activities/{id}/rates/rateId`	Deletes one rate for an activity.

Table 2.2: An overview of API endpoints, which are used to management of activities within Kimai software.

Analysis of existing time-tracking applications

Analysis of UI and UX of existing time-tracking applications is described in this subsection. The main focus of the analysis is on usability theory, described in Section 2.4. More specifically, the focus will be on the present of persistent navigation with all important items, like logo, way to home page, I am here indicator, etc. In addition, a visual hierarchy of applications will be analyzed, all based on the usability theory mentioned above.

Kemai

Kemai[18] is a Kimai desktop client available on Windows. It uses Kimai API 2.5 for connection to the Kimai server. The application is implemented in C++ programming language.

Kemai is providing only fundamental features, such as time-tracking use case, creation of customers and projects with limited data. The application does not implement team management. It has a relatively simple user interface, as shown in Figure 2.12. The UI has only one page with a simple static menu on top. The server management in *Profile* section is available in the top menu. It uses dialog windows with forms to create instances of customers, projects, and activities. The time-tracking use case is available after all fields are filled and is controlled by the tracking icon at the bottom.

The application lacks any visual hierarchy mentioned in Section 2.4. All headlines, including the main one, have the same size. The application can be too tedious to use when Kimai contains a lot of instances of projects and activities. Since the application is using a drop-down component choosing of instance, when there will be a lot of instances, it will not be effective.

[18]Kemai: `https://github.com/AlexandrePTJ/kemai`

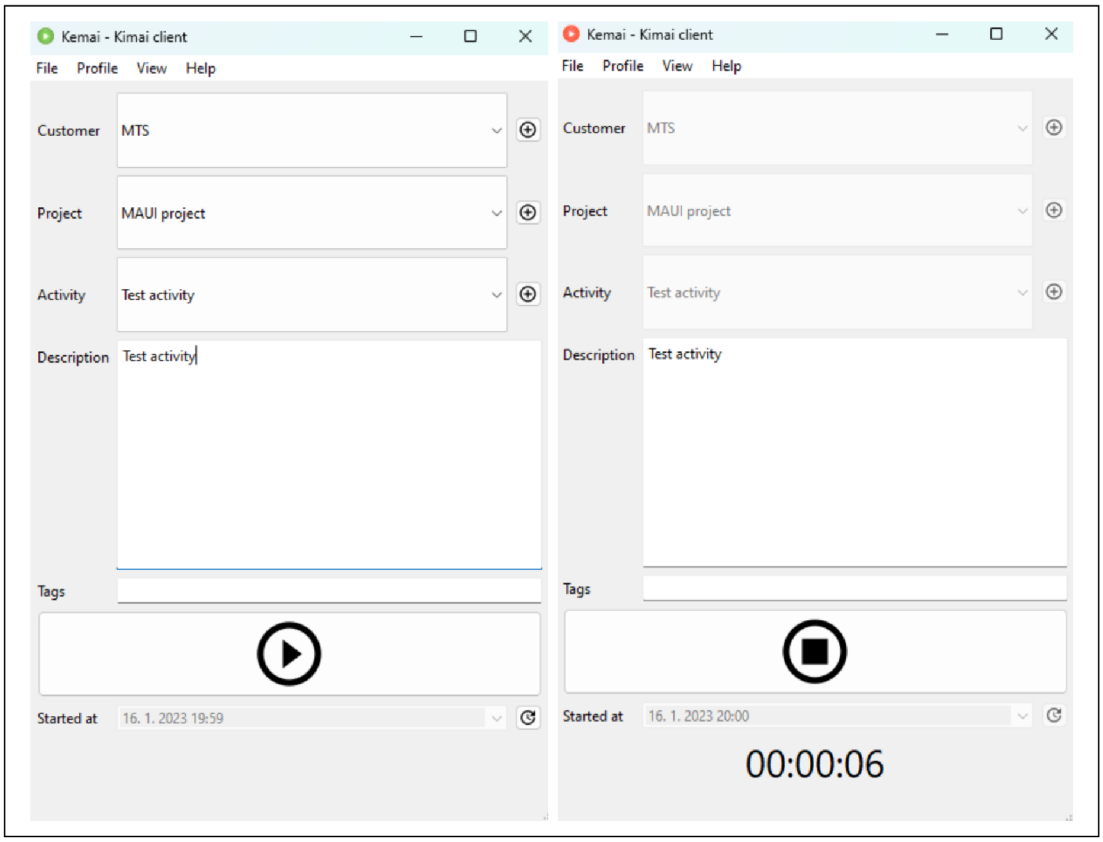


Figure 2.12: Example of Kemai desktop application, which contains basic fields, which need to be filled for time-tracking use case. On the left is application before start of time-tracking, on the right is the application after starting a time-tracking activity.

Toogl Track

Toogl Track [19] is time-tracking software for productivity and profitability needs. It contains a web and mobile application. It has 4 pricing methods, each containing different features [24].

The view of time-tracking use case is shown in Figure 2.13. Toggl uses a tabbed interface for navigation located at the bottom. The first tab is *Timer*, which contains a list of tracked activities and the possibility to start a new activity. When activity is started with the „Play button“, time tracking is instantly started, as can be seen in the image on the right. The activity can be edited after it has been stopped. In addition, new timesheet can be created with the *Plus* button located at the top. After that, a page for activity creation is shown, with input time component and rounding abilities. An example of this view is shown in Figure 2.14 on the right.

The visual hierarchy from Section 2.4 is present on all pages. Each page contains the largest headline and is divided into smaller sections with corresponding smaller headlines. This can be seen in Figure 2.13. The headline also represents an *I am here indicator* for each page. However, the product logo is missing from the application interface.

[19] Toggl Track: `https://toggl.com/track/`

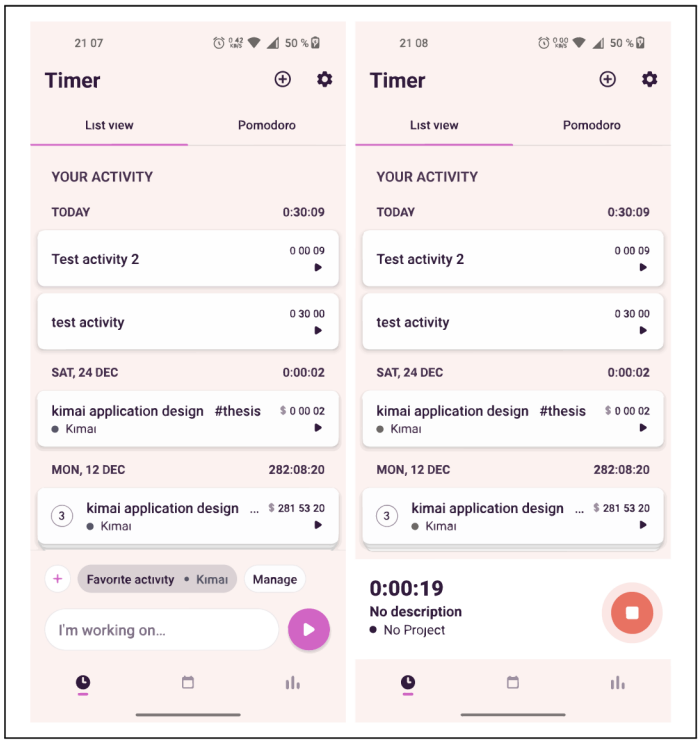


Figure 2.13: Active and inactive time-tracking views of Toggl. On the left is view before time tracking is started, on the right is view after time tracking is started.

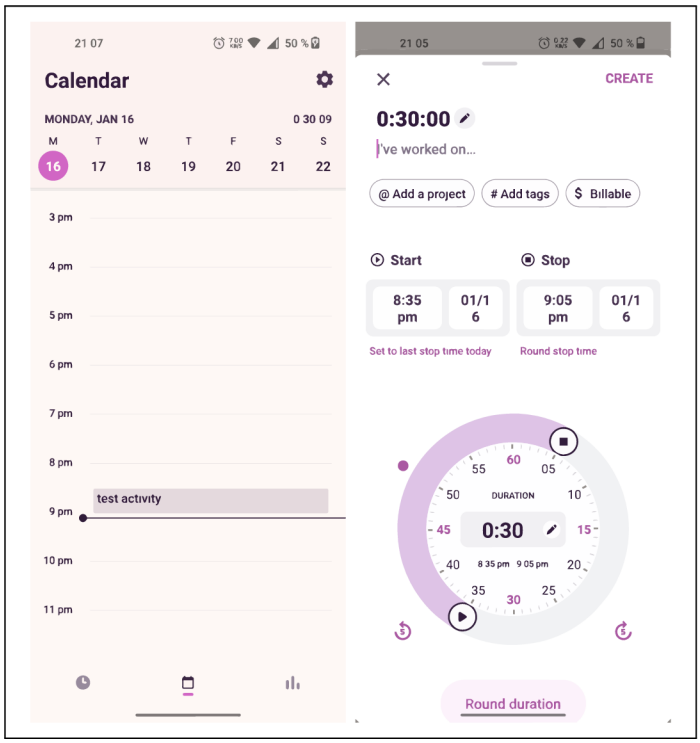


Figure 2.14: Implementation of calendar in Toggl, is shown on the left. View, for adding new activity with time input component is shown on the right.

The second tab represents a calendar. Tracked activities are incorporated into the calendar to create timesheet history, which can be accessed later. The calendar is represented in Figure 2.14 on the right.

The third tab represents reports. Reports can be displayed for a week, day or by custom preferences. Reports contains total tracked hours, billable hours, income, and charts. There is a bar chart for billable and non-billable hours and pie charts for time spent on projects and clients. These reports are represented in Figure 2.15.

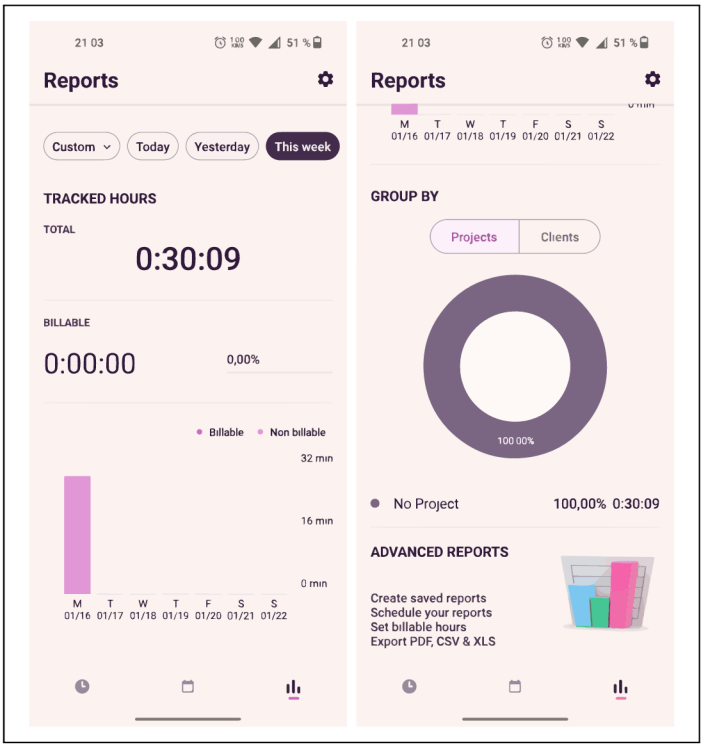


Figure 2.15: Implemented reports within Toggl Track. Image on the left contains information about total tracked time and bar chart for billable hours. Image on the right contains charts for tracked time on different projects or clients.

Toggl track is widely used and is one of the most popular time-tracking application. It provides a lot of features, which varies based on chosen price plan.

Chapter 3

Design and usability testing

This chapter presents my design of the MAUI application created in the Figma online interface design tool [1]. For simplicity, only the mobile version of the application is shown. Additionally, the design was tested with the usability testing method in two iterations with the developed test plan. After each iteration of usability testing, the results were concluded in the corresponding sections. Later, this feedback from users was considered and consequently incorporated into the next iterations of the design. The final user interface design is described in the last section of this chapter.

3.1 Initial design

Based on the analysis of the user experience and the features of other time-tracking applications I created my own user interface design for the MAUI mobile application for Kimai. The user interface was inspired by one of the Figma UI resources, specifically the Google Drive UI resource [2]. Furthermore, the user interface and user experience tips from Section 2.4 were incorporated into the design.

Login page and servers management

First and foremost, I created the server management view, where multiple Kimai servers can be added. After creating a Kimai server with the correct user name and password, the connection can be established. The connection can be established by clicking on the server button within the *Recent servers* panel or by clicking on the connection button within the list of all servers. The information of each server can be modified by clicking on the individual server in the list of servers. User interfaces, which represent these use-cases, are available in Figure 3.1.

[1] Online interface design tool Figma: `https://www.figma.com/`
[2] Google Drive UI resource: `https://uikitfree.com/g-drive-ui-exploration-figma-freebie/`

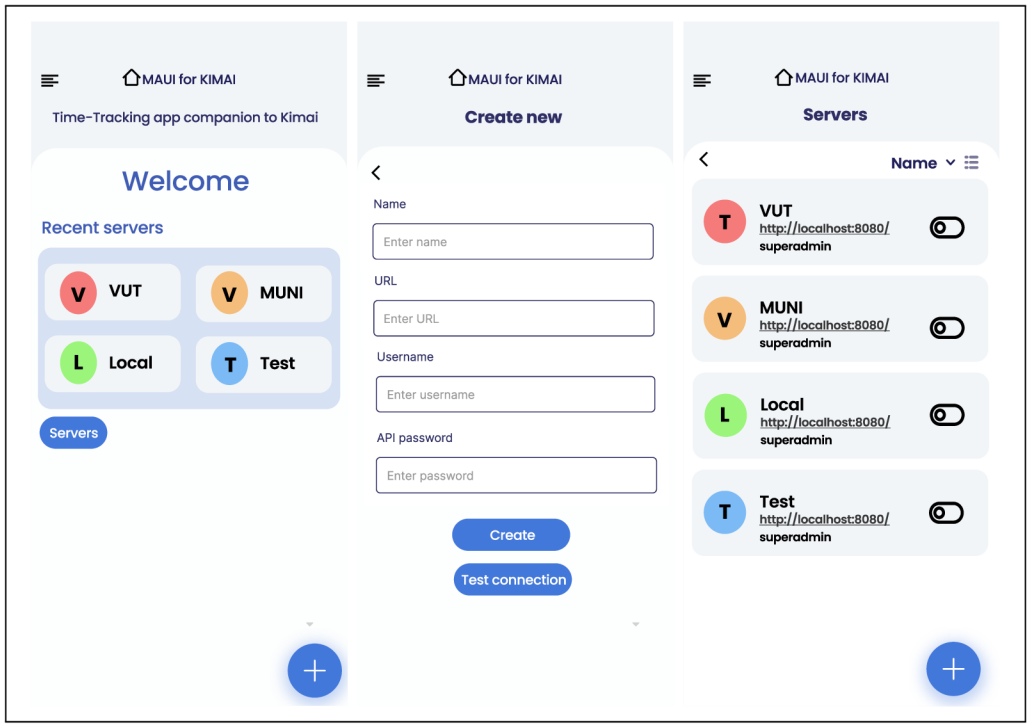


Figure 3.1: Server management views, where users can create new servers, edit them and log in. The left figure represents the login page with recent servers, the middle figure represents view for creating a new server and the right figure describes lists of available servers.

Menu, home page, and time-tracking

After successful log-in, the user is redirected to the home page. The home page is represented in Figure 3.2 by an image in the middle. The home page contains a widget with information about time-tracking and also a way to start a use-case of time-tracking. Based on user experience tips from Section 2.4, the page contains persistent navigation in the upper part of the image. This navigation is available across all pages of the application. It contains product identificator, which is also a link to home page, page description, way to a search, and access to a menu. Each tracked timesheet is then stored in the catalog of all timesheets, which is represented in Figure 3.2 on the right. In addition, each timesheet in catalog contains a tag, which corresponds to assigned project. The menu is shown in Figure 3.2 on the left and contains a simple way to navigate between different views with the possibility to log out of a server.

Time-tracking can be started by clicking on the button *Start tracking*. After that, a dialog will be shown, where the user can choose whether to create a new time sheet or to select a template. This dialog is shown in Figure 3.3 on the left. If user decides to create a new timesheet, *Create new* view will be shown with all required fields to fill up. By clicking the *Start* button, time tracking will start. This case is represented in Figure 3.3 in the middle. On the other hand, if the user clicks the *Choose template* button, a corresponding view will be shown, where the existing template of the timesheet can be chosen. After clicking on the template, the time-tracking will automatically start. The choice of template is represented in Figure 3.3 on the right.

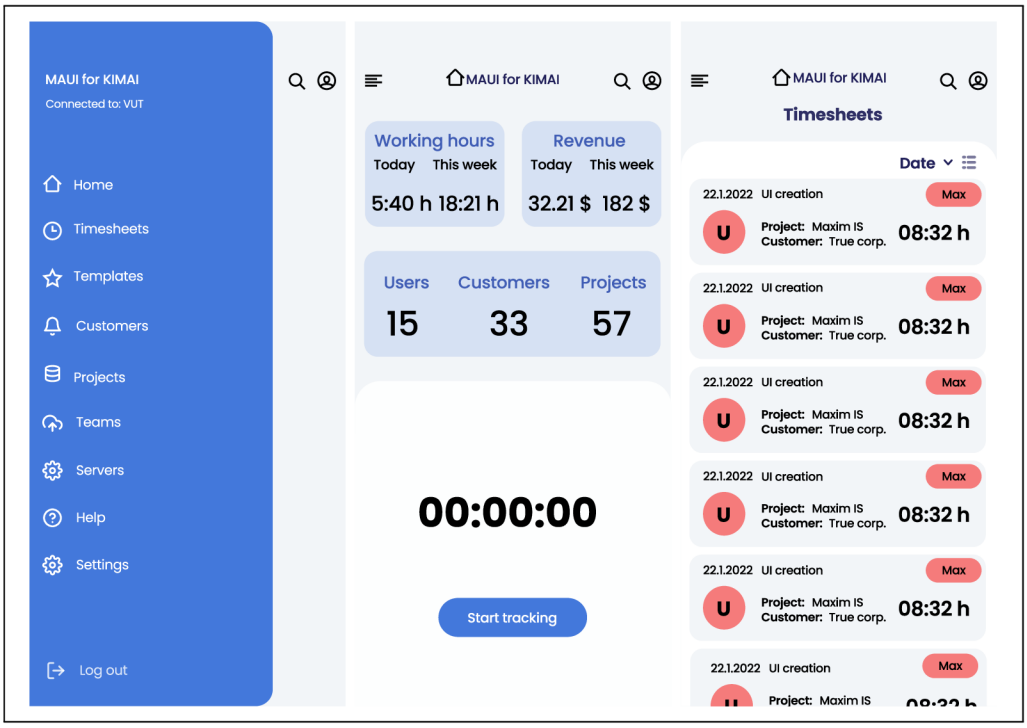


Figure 3.2: On the left of this figure is designed menu, which represents easy navigation between each page, in the middle is described home page with information widgets and possibility to track time and on the right is showcased catalog of all timesheets.

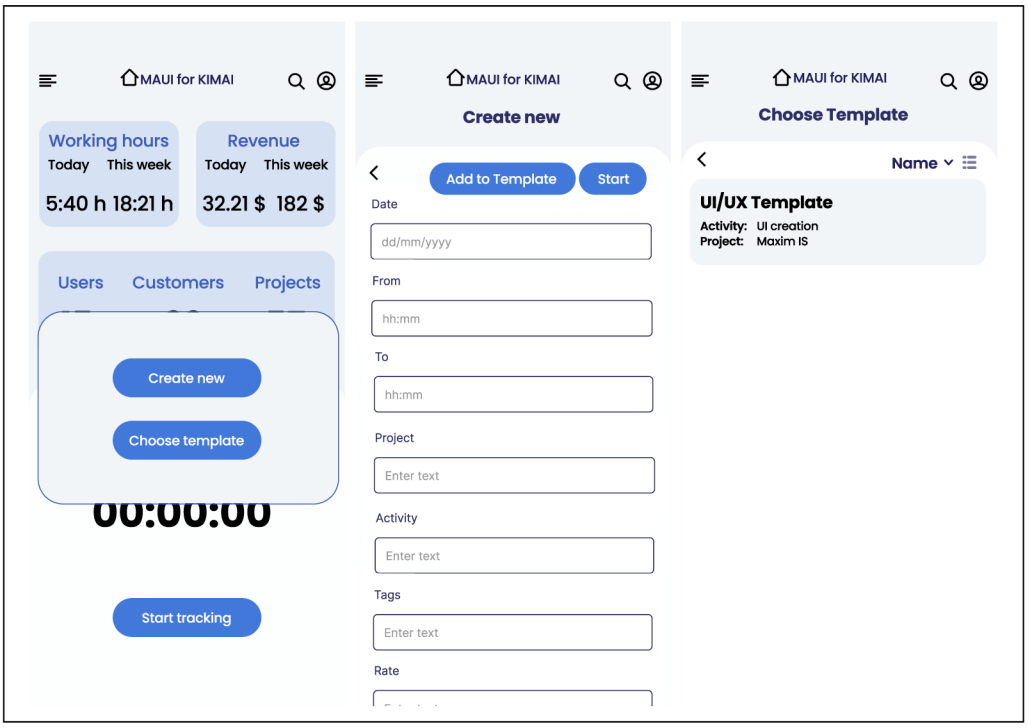


Figure 3.3: Use-case of time-tracking, where the user can choose between creating a new timesheet or using a template.

Customers, projects and teams

I decided to implement views for managing customers, projects, and teams within the mobile application. This will be useful for teamleads and administrators within Kimai. In my opinion, management of these entities would not be as effective in mobile application as on a desktop. Therefore, better implementation of these use cases will be on the desktop application. Management of customers, projects, and teams is similar. All three contain a catalog of available entities. An example of this catalog is shown in Figure 3.4 on the left, specifically the catalog for projects. In the middle, detail of entity is shown, in this case, specifically of the customer, where more information can be found and edited. The right view can be found for adding projects and users to the team.

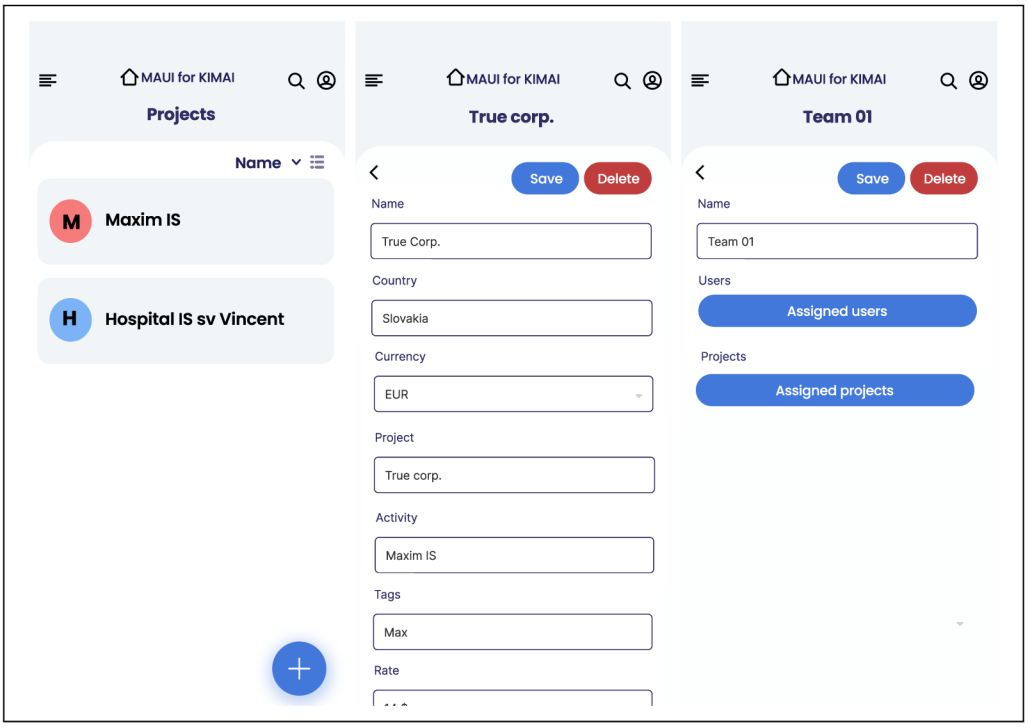


Figure 3.4: Examples of catalog, details view for projects, customers and teams for management purposes.

3.2 Process of usability testing

After the initial user interface design is created, usability testing can be performed to test the user experience of the designed application and collect user feedback. Based on the usability testing theory described in Section 2.4, I developed a test plan with tasks and questions for the participants.

Test plan

The main objectives of the usability testing of the MAUI for Kimai application are:

- Test user experience of designed UI,
- identify missing features,
- observe users behavior and identify problems of UI design,
- gather feedback of participants to improve design in next iteration.

Usability testing will be conducted in the area of Brno University of Technology, particularly in the Faculty of Information Technology. Participants will be students and employees of the Faculty of Informatics. The test session is divided into 3 parts: introduction, tasks and questions, and conclusion.

Introduction

In introduction, I will introduce myself and explain the problematics of my diploma thesis. I will explain Kimai time tracking software and my aim for usability testing. Moreover, I will ask for the name and occupation of participants and also for consent, whether the results of test session can be used within my diploma thesis.

Tasks and questions

After the introduction, the usability session tasks will be proposed. The one test session contains five tasks for each participant with additional questions. The test contains the following tasks with questions:

1. The goal is to connect to the existing Kimai server. The welcome page is displayed and the participant is asked questions. Participants are asked how they feel about the user interface. How do you feel about the *Recent servers* UI element? Are you missing something in the log in use case?

2. The goal is to perform a time-tracking use-case. After successful login, the home page is displayed and questions are asked. How do you feel about elements on the home page? Is the corresponding information on the widgets sufficient? Are you missing something in the *Timesheet* catalog? What do you say about the menu for choosing between templates and new timesheets?

3. The goal is to create a new project. The project page is displayed, and the participant is asked questions. What do you say about the implemented use case of project creation? Do you think it is sufficient for mobile devices? Do you miss something in the project catalog?

4. The goal is to add users and projects to existing *Team 01*. The team page is displayed and the participant is asked questions. What do you think about the detail view of *Team 01*? Are existing UI elements sufficient to add users or projects to the team?

5. The goal is to display the user profile and edit existing information. The profile view is presented and the participant is asked questions. Do you like the described reports of users? Are you missing something in those reports?

Conclusion

In conclusion, participants will be asked about the overall feel of the application. Moreover, participants can explore an interactive design prototype and add more views on their own. More questions will be asked. For example: Are you missing something in the application? What did you like and disliked? Can you imagine that you can use this application in the future?

3.3 Report after the first iteration of usability testing

I tested 10 users in the first iteration of usability testing. The participants were students of Brno University of Technology. The test sessions were held in the library of the Faculty of Informatics. Reports are concluded for each task.

1. Task: Connecting to Kimai server

- There will not be many servers, therefore, it would be good to consider managing recent servers.
- When users accessed the list of all servers, they were confused about the toggle button, which should connect to each server. It would be better to connect to the server by clicking on an item in the list and accessing the server details with the other button.
- In general, there were not many complaints about the server management use case.

2. Task: Time-tracking use-case

- When accessing the time-tracking page through the menu, some participants had problems in correctly navigating through the menu items. One of them proposed that it would be better to visually divide menu items by logical meaning.
- For some participants, the information within the widgets was not sufficient. In addition, server-specific information is not relevant to all users.
- Some participants would like to have customization of information widgets with the possibility of accessing reports of time-tracking.
- The time-tracking use case with a dialog menu has too many steps. The dialog menu can be deleted, and the corresponding buttons can be moved to the home page.
- Some participants requested the possibility of stopping tracking. Therefore, it would be nice to consider the icon of start and stop.

- If the user wants to restart an existing timesheet, the user has to access the timesheet catalog. It would be easier to access some of the recent timesheets directly from the home page.
- On the *Choose Template* page, there is a missing button to create a new template.

3. Task: Create a project

- Most of the participants missed the visual connection between the projects and the customers. Visual connection should be present within catalog items in both projects and customers.
- It was proposed to differentiate between active and inactive projects.
- Each project has an assigned customer. However, each customer should also have an assigned list of projects.

4. Task: Adding a member and project to the existing team

- The team lead button within the catalog item is not intuitive.
- Overall, the participants were okay with this use case.

5. Task: Edit of user profile

- The profile icon within the menu is not necessary.
- The list of available projects is missing.

Conclusion of the test session

The participants were mostly satisfied with the interface of the application UI. There were a few gimmicks with some design decisions (mentioned in reports above), however, this will be fixed in the next iteration of design. One of the participants proposed biometric authentication to the Kimai servers. I am not sure that this is a necessary feature since most of the users will use only one server.

3.4 Design after the first iteration

After the first iteration of usability testing, some changes were incorporated into the second version of the design based on feedback from the participants.

First, the menu was reorganized by logical meaning. Related items were grouped together and rearranged by importance. The modified menu can be seen in Figure 3.5 on the image on the left. Most of the participants were confused about the switch button in the server catalog. Therefore, this button was replaced by the setting button, which can be seen in Figure 3.5 in the middle image. Now, the connection to the server will be established by clicking on the catalog item, and the server description will be accessed by the settings button.

Most of the changes were made on the home page. These changes can be seen in Figure 3.5 in the image on the right. Now, there is only one customizable widget. By clicking on the widget, the time-tracking reports will be accessed for each user. Additionally, a list of

recently tracked timesheets is added. Users can now restart recent timesheets directly from the home page. Furthermore, if the user forgets to track the time, the timesheet can be added retrospectively using the *Add* button. Last but not least, the time-tracking dialog was removed and a tracking button was introduced. This change should remove redundant steps within the time-tracking use case.

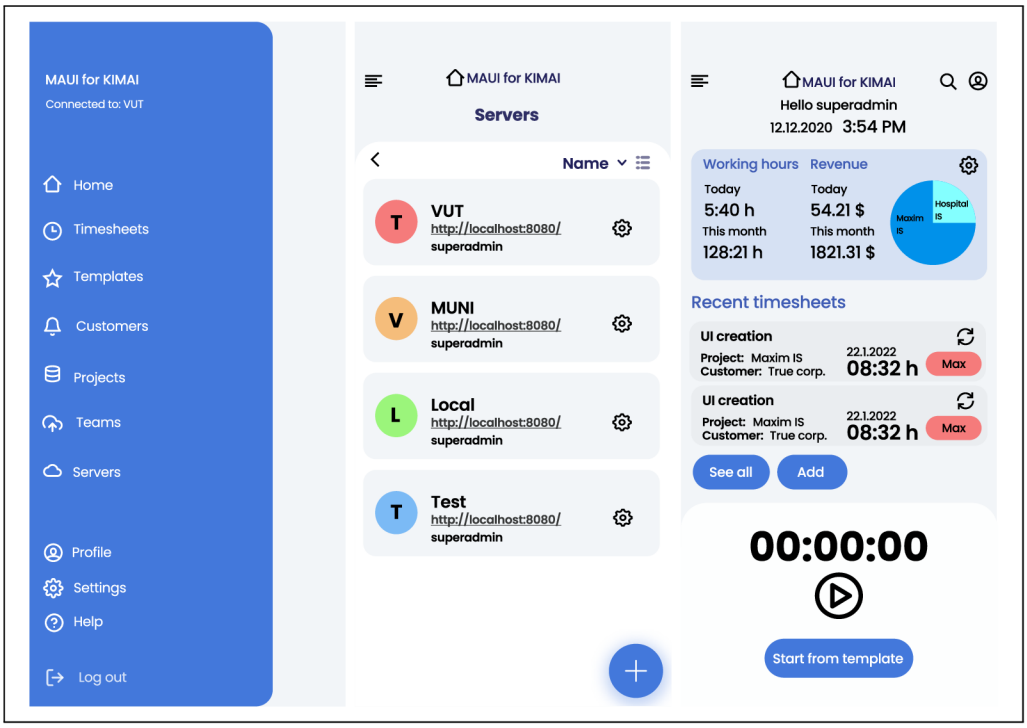


Figure 3.5: Examples of pages, where changes were introduced after the first iteration of usability testing. On the the left is a reorganized menu, in the middle is a list of existing servers with settings button and on the right is restructured home page with customizable widget, new time-tracking button and with catalog of recent timesheets.

In addition, customers now have a list of assigned projects. The possible design can be seen in Figure 3.6 on the left. After clicking on a project within the list, the user will be redirected to the details of the project. Additionally, items in the catalog of all projects now contain tags that assign the project to each customer. Because of these tags, each project is visually differentiated from others, which should lead to better navigation. The redesigned project catalog with tags is shown in Figure 3.6 in the middle.

Other changes were made to the user profile. These changes can be seen in Figure 3.6 on the right. Now, the profile contains detailed information about the user. Through the profile section, the user can see the assigned projects and teams. If the user has a management role, it is possible to access server information through the UI element tab at the bottom of the page.

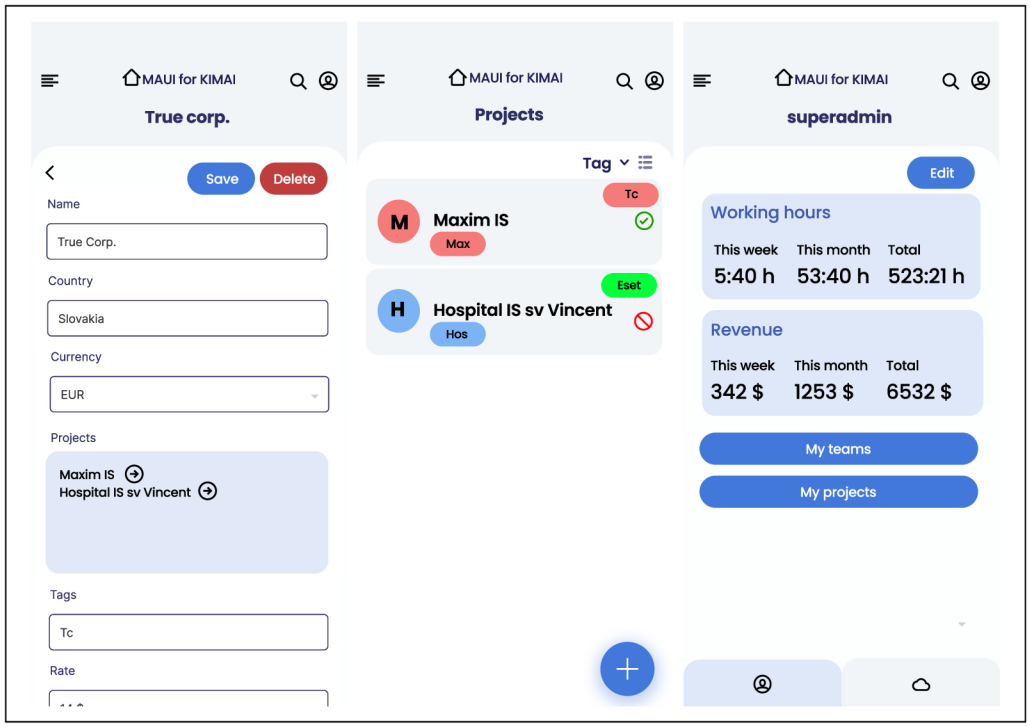


Figure 3.6: Examples of pages, where changes were introduced after first iteration of usability testing. On the left is customer detail with list of assigned projects, in the middle is catalog of all projects with tags and indicator of activation of project and on the right is redesigned profile section.

3.5 Report after the second iteration of usability testing

In the second iteration of usability testing, I tested 6 users. The participants were employees of Brno University of Technology. These employees are possible users of the MAUI application. The test sessions were held in the network vestibule of the Faculty of Informatics. Feedback is concluded for each task below:

1. Task: Connecting to Kimai server

- In most cases, there will be only one server, so the automatic log-in feature should be considered.
- Also, if there are a small number of servers, the list of servers can be moved to the main page to minimize the number of steps within the UI interaction.
- Some participants were confused about log-in use case, it would be nice to integrate some feedback with information about log-in into server.

2. Task: Time-tracking use-case

- For some participants, the icon of restarting the timesheet within the *Recent timesheet* catalog was misleading. It should be considered to use other icons.

- When tracking is running, information about the currently running timesheet should be shown. Also, there should be the possibility of editing that timesheet or adding it to *Templates*.
- Participants often track time at work related to a specific task within Gitlab or Github issues. Therefore, this feature should be built into the application.
- Some of the participants proposed using gestures to track time.
- Some participants were confused with usage of tags. Kimai contains *Acitivity type* settings that can be used as an identificator.

3. Task: Create a project

- Not all participants will access this section. It was proposed to hide this section.
- However, there were not many complaints about the project creation use case.

4. Task: Adding a member and project to the existing team

- The same as for the third task, not all participants will be accessing this section.
- Assigning teamlead from an item of the team catalog was not intuitive, it would be better to move it to the team detail.

5. Task: Edit of user profile

- The administrator should have access to a view that contains server information.
- Users should have access to reports from profile. Reports should be defined by the user role. For example, administrator should see server reports, teamlead should see reports about yourself and team members.
- There should be a history of time-tracking within the team. Consider some implementation of a calendar or other component.
- Assigning teamlead role from an item of the team catalog was not intuitive, it would be better to move it to the team detail.

Conclusion of the test session

Participants did not have many problems with the application design. They quickly mastered the design of the application and were able to navigate the application effectively. The availability of reports was a feature that was widely requested. The user would like to have detailed information about the history of time-tracking, whether about yourself or within the team. The application should also contain an implementation of the export mechanism with support for different formats. In addition, some of the participants proposed to introduce automatization of some tasks. For example, create an implementation of the CLI (Command Line Interface) within the desktop application, where some basic tasks can be planned and automatized. This planning approach would be interesting to incorporate into mobile application as well.

3.6 Final design

This section represents the final design of the user interface after two iterations of usability testing. Again, I was focusing on changes from the second iteration of usability testing.

On the basis of user feedback, new changes were introduced in the main menu. Now, the menu items are divided into sections by logical meaning, and the *Reports* menu item is introduced. In addition, when the user toggles the *Management role* button within the details of server login, *Management section* can be seen in the menu. Otherwise, it is hidden. The redesigned menu is shown on the left and the new server detail is shown in the middle in Figure In addition, *Welcome page* is redesigned. Based on participants' feedback, there will often be a few Kimai servers. Therefore, the server catalog is moved to the *Welcome page*, and the *Recent servers* UI component from the first iteration is replaced by a default server. When the default server is set, the user is automatically logged into the corresponding Kimai server. The redesigned *Welcome page* is shown in Figure 3.7 on the right.

The home page now features a new swipe-up detail page located at the bottom. With this menu, the user can easily start and change the activity. In addition, a reference to the GitHub or Gitlab issue is added, and the start icon of *Recent timesheet* is replaced. All of these changes can be seen in Figure 3.8. The swiped-up detail menu is shown in the middle. Details of the project with the possibility to activate it are shown in Figure 3.8 on the right.

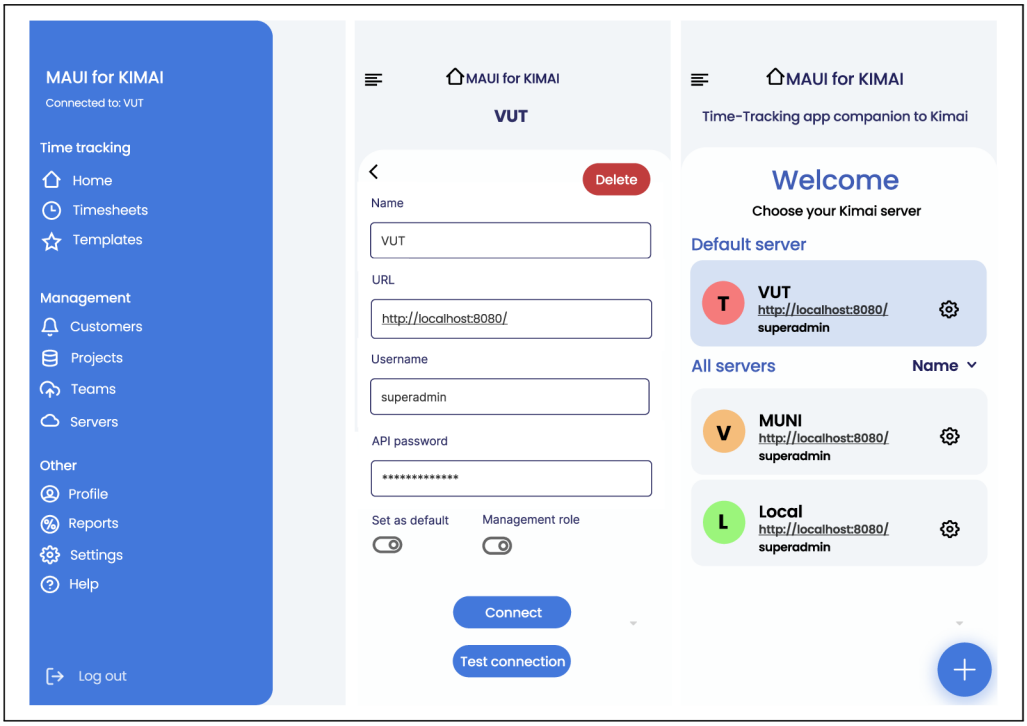


Figure 3.7: Example of design, where newly redesigned menu by logical section is shown on the left. The detail of server now contains buttons for setting a default server and for a management role, shown in the middle. Catalog of all servers is moved to the Welcome page and new default server feature is introduced, as can be seen on the right.

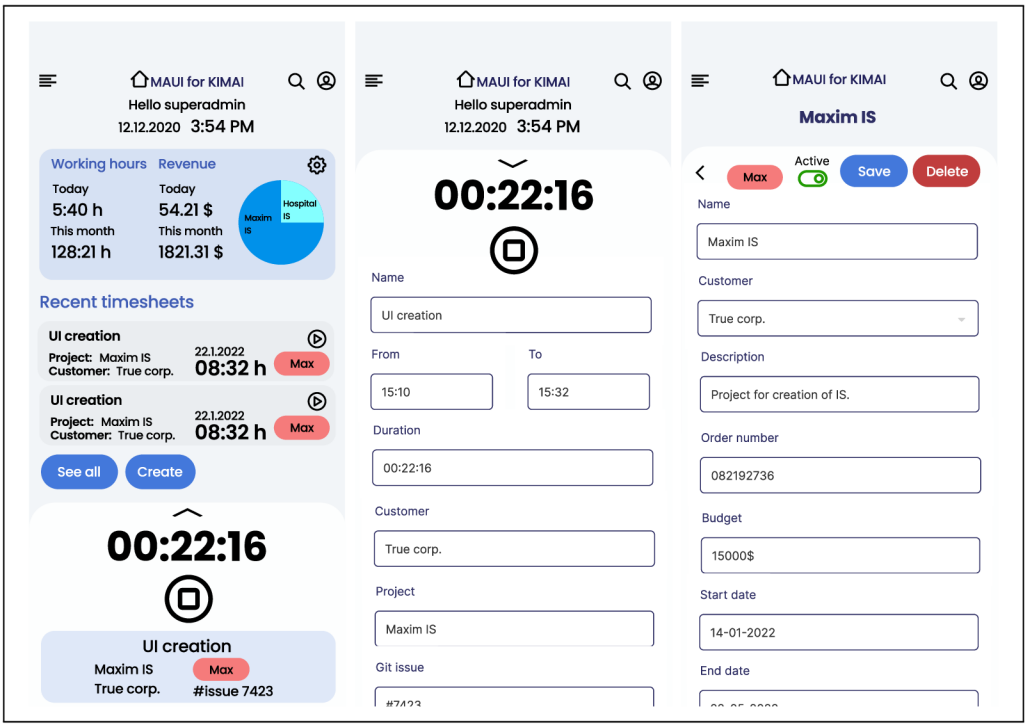


Figure 3.8: Example of design of home page, where new swipe-up detail view is introduced. This can be seen in the images on the left and in the middle. Moreover, the detail view of individual project with possibility to set it on active is shown on the right.

Tabbed interface is introduced with three individual tabs within the user profile. The first tab contains access to user teams and projects, the possibility to edit users' information, and some basic stats. The second tab contains user-based reports, which show tracking information within the team, and time spent on different projects and customers. The third tab is shown only for the admin user, where server-wide statistics can be found. The design of these tabs is shown in Figure 3.9. Within the management of team, new way of assigning teamlead was introduced. Now, teamlead is assigned within the detail of the individual team. Examples of team management can be seen in Figure 3.10.

The final design of the application contains all identified features which are needed by potential users. The design was iteratively improved and may be a little bit different in implementation, based on limitations of the .NET MAUI framework.

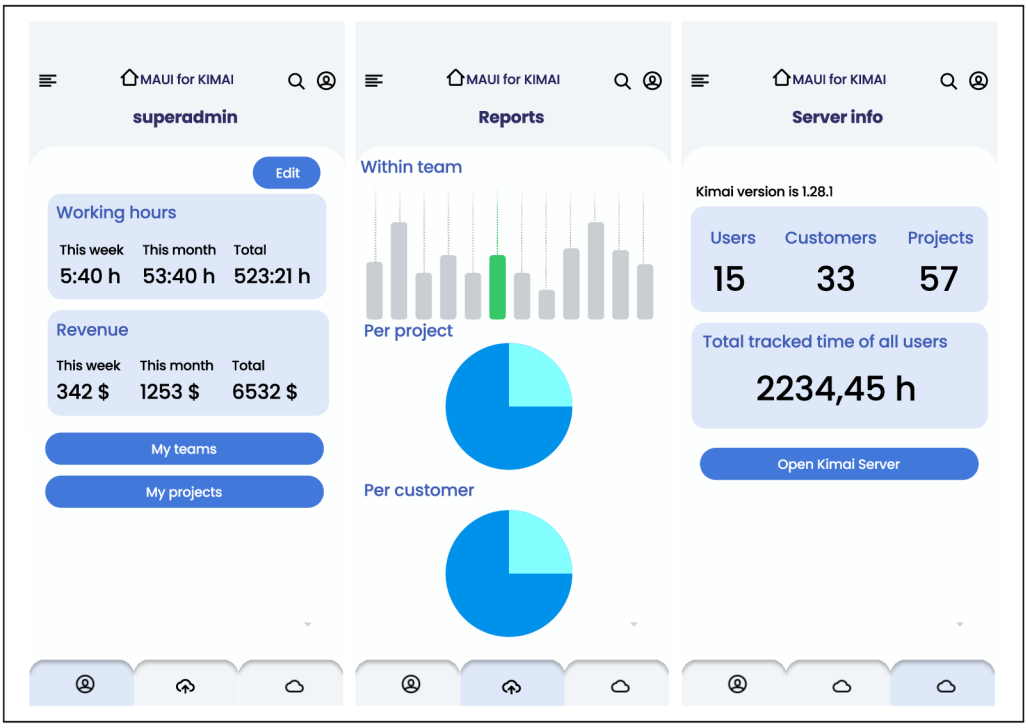


Figure 3.9: Example of design of tabs within the user profile. The first tab contains user information located on the left, the second tab contains reports, located in the middle and the third tab contains server-wide information, located on the right.

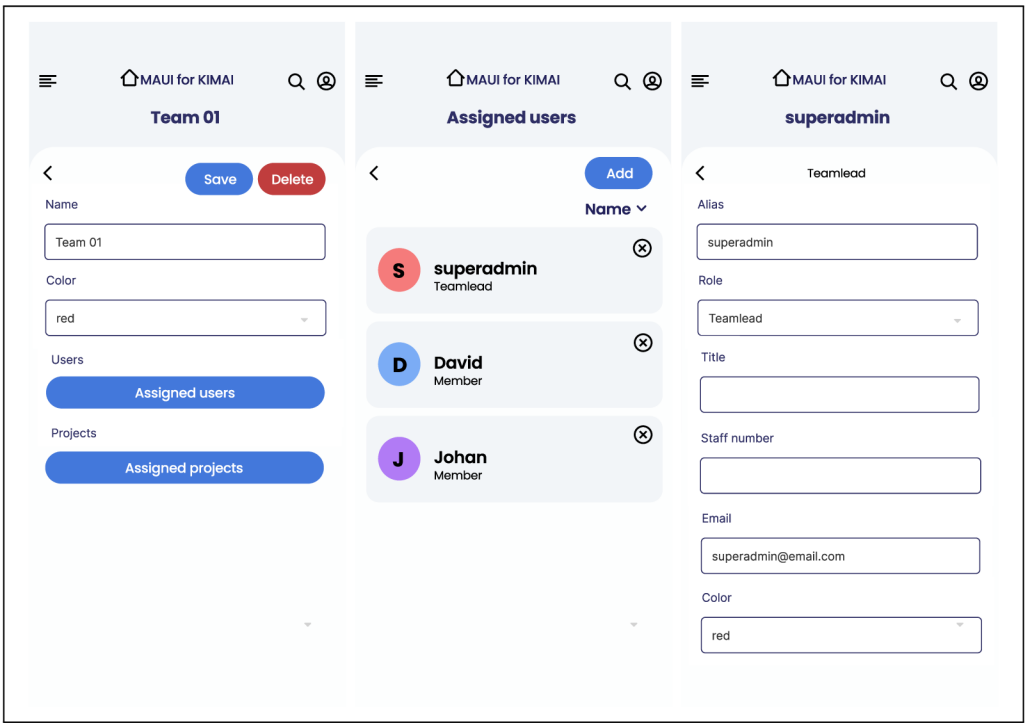


Figure 3.10: Example of design of management of teams and assigning users to the team.

Chapter 4

Implementation

This chapter describes my reasoning for choosing the most suitable architecture for this kind of application. It also describes used libraries, technologies, and implementation with my approaches to various problems which occur during development. This chapter should serve as a basic approach for the development of multiplatform applications in .NET MAUI.

4.1 Architecture

Choosing right architecture is important for any type of application. In general, it defines the overall structure and organization of an application and how the application components interact with each other. It also influences the maintainability, extensibility, and testing of the application.

I found three main approaches for architecture on .NET MAUI:

- Code-behind approach,
- Model-View-ViewModel (MVVM) approach,
- MVVM + Clean architecture.

Code-behind approach

This is a basic approach, where the application logic is written in a separate file, which is linked to the UI (in this case, the markup XAML file). When creating XAML-based UIs, the default setup is for a XAML file with an associated code-behind file typically written in C#. In other words, the two different files are partial classes written in two different languages that create a single class which can then be used like any other class [19]. For instance, the following code snippets 4.1 and 4.2 represent an XAML file and its corresponding code-behind class.

One of the advantages of this approach is the relatively simple handling of user interactions in the UI. As is shown in code snippets 4.1 and 4.2, button click is handled by `Button_Clicked` event declared in code-behind class. However, there are some potential drawbacks to using this approach. First and foremost, UI and application logic are tightly coupled in code-behind class. That means, when developer wants to change the UI, often the logic must be changed as well. This also applies the other way around. What is more, there is poor separation of concerns. When a developer wants to separate presentation

```
1  <ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
2               xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
3               x:Class="MauiForKimai.Views.NewPage1">
4
5      <VerticalStackLayout>
6          <Label Text="Welcome to .NET MAUI!" />
7          <Button Clicked="Button_Clicked" Text="Click me" />
8      </VerticalStackLayout>
9  </ContentPage>
```

Figure 4.1: Example XAML markup file, which is tied to code-behind class named `NewPage1` by `x:Class setting` on line 3.

```
1  public partial class NewPage1 : ContentPage
2  {
3      public NewPage1()
4      {
5          InitializeComponent();
6      }
7
8      private void Button_Clicked(object sender, EventArgs e)
9      {
10         //handle button click
11     }
12 }
```

Figure 4.2: Example C# code behind file, which is tied to XAML markup file by `partial` keyword and contains methods for handling a click button action.

logic, business logic, and data access it can be really challenging. This also makes it harder to reuse code, test the application, or modify it in the future.

Model-View-ViewModel approach

The Model-View-ViewModel (MVVM) solves problems arising from the code-behind approach. As in the code-behind approach, the UI and application logic is tightly coupled, MVVM clearly separates the application's business and presentation logic from its user interface. Maintaining a clean separation between application logic a and UI makes the application easier to test, maintain, and evolve. Furthermore, it can significantly improve the reusability of code and allows developers and UI designers to collaborate in an easier way [23].

There are three fundamental components of MVVM:

- View – user interface, which is typically implemented in XAML,
- ViewModel – separated application logic, which is connected to UI by bindings,
- Model – encapsulated application data models.

Figure 4.3 shows how each component interacts with each other. At a high level, the view knows about the viewmodel, and the viewmodel knows about the model. However, the model is unaware of the viewmodel, and the viewmodel is unaware of the view. Therefore, the viewmodel isolates the view from the model and allows the model to evolve independently of the view.

Notifications between components are handled by the notification mechanism, which is implemented by the `INotifyPropertyChanged` interface. UI interactions are handled by `Commands` and data from viewmodel to view is provided by `Data Bindings`. Moreover, communication between different viewmodels is often needed. This can be handled by implementing `Messenger`. These approaches are discussed in more detail in the CommunityToolkit section 4.2, from which implementations of notifying, commanding, and messaging are taken over.

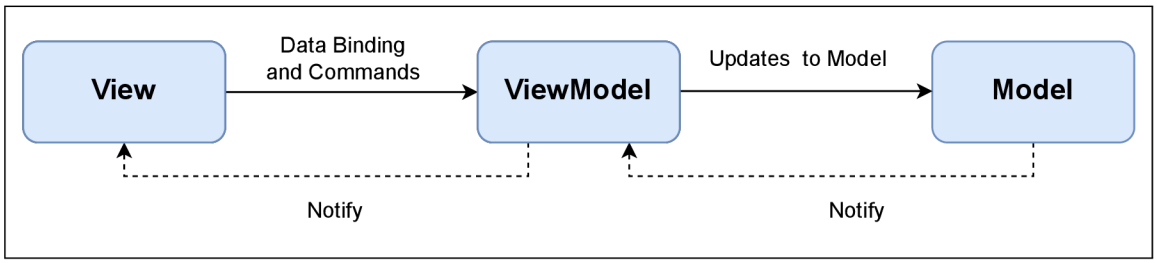


Figure 4.3: Diagram of MVVM architecture, which shows interactions between core components of MVVM.

Typically, .NET application architecture developed with MVVM contains separated files as views (often XAML files), which binding context is set to the corresponding view-model. In other words, by setting the binding context of view, it is possible to bind the user interface elements (such as entries) to the corresponding viewmodel variables. Furthermore, such applications contain multiple different services. For instance, services for persistence, API communication, routing, etc. These services are often provided by dependency injection into viewmodels. Dependency injection is discussed more in Section 4.2.

MVVM and Clean architecture

The MVVM approach can also be extended by Clean architecture. This can be really beneficial for developers who develop .NET MAUI application and want to connect application to their own APIs and databases implemented in ASP.NET [1]. This basically enables full-stack development within .NET ecosystem from one solution, where code can be maximally reused. Clean architecture divides application into four main structures:

- Domain – contains core entities and core logic,
- Application – defines application main services,
- Infrastructure – defines external systems (persistence, wrappers),
- Presentation – GUI, with which users interact.

[1] ASP.NET is framework for building web applications and web services in .NET: `https://dotnet.microsoft.com/en-us/apps/aspnet`

What is really important in clean architecture is a flow of dependencies. Dependencies flow inward as shown in Figure 4.4. The domain layer should have no dependency as it should be the most stable part of the system. The application layer depends on the domain layer but not on anything outside of itself. It depends only on abstractions defined in the interfaces, and these interfaces are implemented in outer layers. This is possible by the Dependency Inversion principle [2]. The infrastructure layer contains external systems (such as persistence, authorization) and depends on application layer. Last but not least, the presentation layer depends on Application and Infrastructure [13].

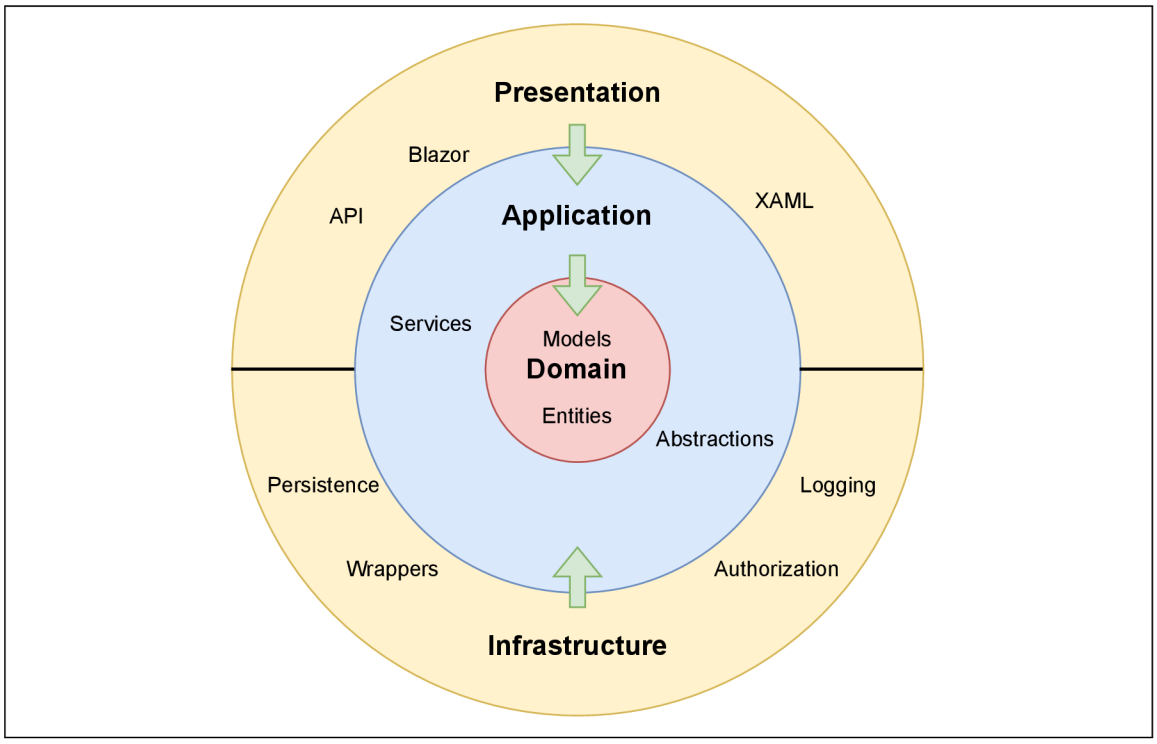


Figure 4.4: Overview of Clean architecture, where dependencies between components flows inwards.

Clean architecture brings a lot of advantages for code reusability and maintainability with MVVM pattern. However, for small and simple applications, it can be overwhelming and can introduce a lot of boilerplate code.

[2]Dependency Inversion principle: `https://deviq.com/principles/dependency-inversion-principle`.

Overview of application architecture

Based on the architecture approaches available for .NET MAUI and its advantages and disadvantages, I have decided to use **MVVM approach** with the use of **services** without a clean architecture for my application. Services are abstracted by correspondent interfaces on which the application is dependent. Since the Kimai API has already been implemented, using a clean architecture is not necessary, as it only brings more complexity. On the other hand, using the code-behind approach for this kind of complex application would lead to problems with code reusability between multiple platforms.

The implemented application is divided into four main structures (libraries):

- MauiForKimai.Core,
- MauiForKimai.ApiClient,
- MauiForKimai.App,
- MauiForKimai.Tests.

`MauiForKimai.Core` is a core library of the application. It is the most stable and it contains as little as possible external dependencies. It only has dependencies in form of NuGet packages for CommunityToolkit, Sqlite database and FluentValidation. All other application libraries depend on this core library. In addition, this library contains entities for database tables, main data models classes and mappers, which map entities to models. It also incorporates essential interfaces and enums used across application and validators models.

`MauiForKimai.ApiClient` is a library, which contains an auto-generated Kimai API client and its data transfer objects (DTOs). This API client is abstracted into interfaces in form of services, which implementation is then use in application to access data from API. Moreover, mock implementation of these services is also used for unit test purposes. ApiClient library also contains implementation of authorization to Kimai and essential mappers between API's DTOs and application models. The implementation specifics of Kimai API integration into .NET MAUI are discussed in Section 4.3. ApiClient has dependency only on Core library.

`MauiForKimai.App` is a concrete implementation of .NET MAUI application and its resources. It contains exact implementation of multiplatform views in XAML, application business logic in form of viewmodels, important application services such as access to persistent storage and routing. It also contains wrappers classes, which are used for abstraction of coupled code and application static resources. It depends on ApiClient and Core libraries.

`MauiForKimai.Test` is a testing library, which servers for the unit testing of application. It contains fundamental unit tests with mock implementation of services, which are needed for testing purposes. It has dependency to .NET MAUI application.

The package diagram of the application architecture described above is shown in Figure 4.5.

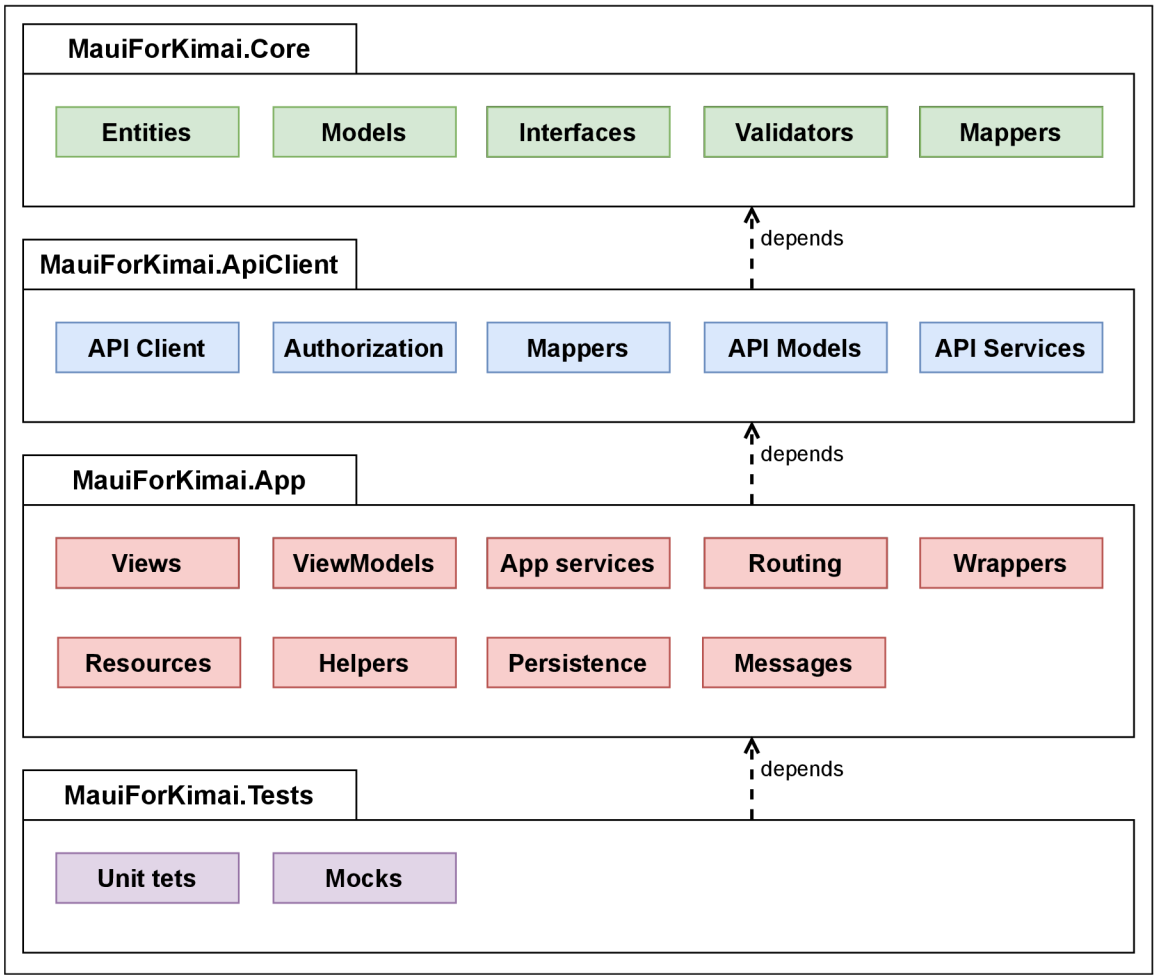


Figure 4.5: Package diagram of application architecture, which shows four main libraries and their elements with dependencies between them.

The two main fundamentals of this architecture are the use of the MVVM approach and the use of services. On the first hand, the MVVM approach brings a correct separation of concerns within the application. Therefore, I can create reusable code across multiple views, which is relatively easy to test. On the other hand, the services abstract all essential features of application (data access, API access, authorization, routing, etc.). The one thing that glue everything together is a principle of **dependency injection**. Due to dependency injection, services are injected into the corresponding viewmodels and can be used to access data from abstracted entities. As a result of this, the obtained data can be shown in the UI using data bindings. This fundamental concept of application is shown in Figure 4.6

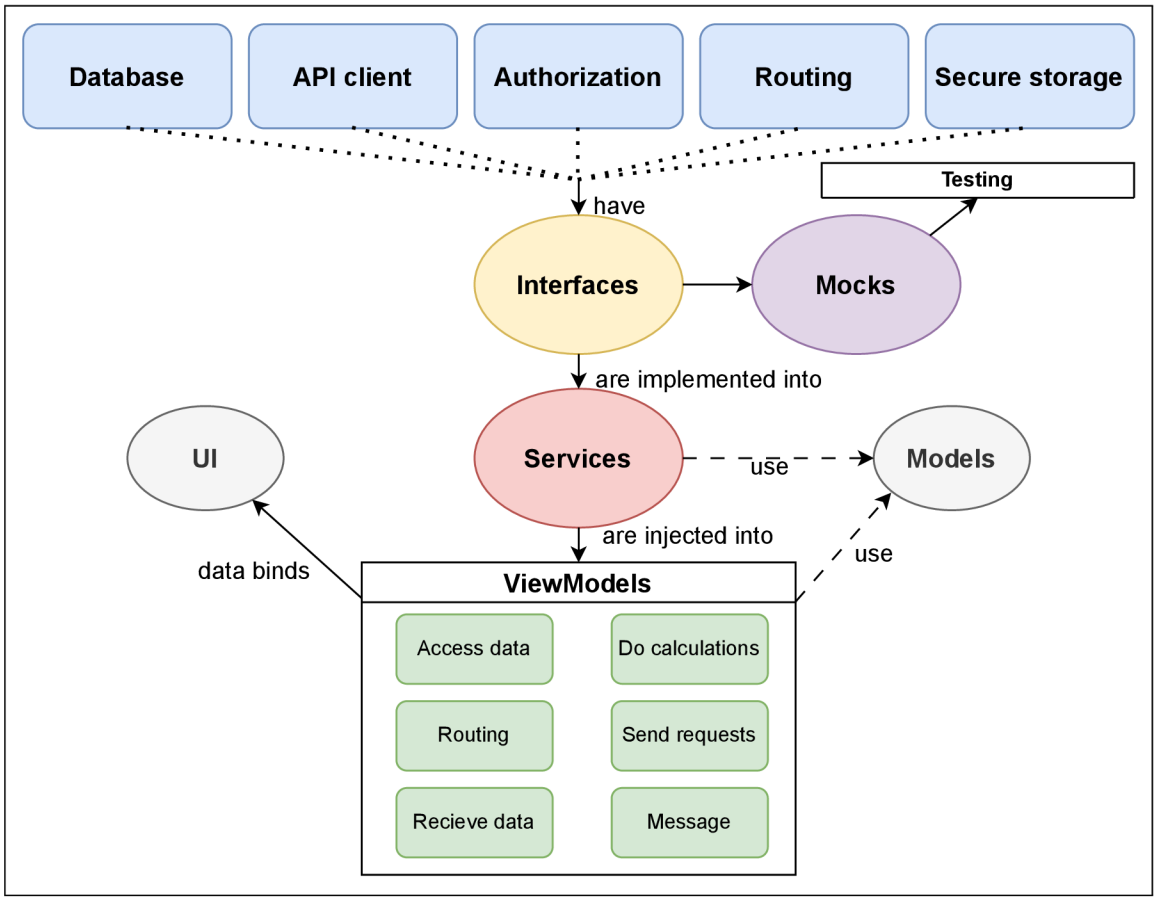


Figure 4.6: Fundamental concept of application, where services (which are abstractions of different entities) and viewmodels are glued together by dependency injection.

4.2 Essential technology and used libraries

Before I move on to the details of the implementation of my multiplatform application, firstly, I would like to mention the most important technology and libraries that I used during the implementation part of my diploma thesis. Thanks to these libraries, a lot of things were simplified and streamlined. Most of these libraries and technologies are open-source and well-maintained.

The application has been developed on Windows 11 with Visual Studio 2022 on .NET 7.

Dependency injection with Scrutor

As mentioned in section 4.1, the dependency injection (DI) is a very important concept used in MAUI applications. On the startup of the application, the DI container is initialized. DI container is an object that knows how to instantiate and configure other objects (for instance, their life-time) and all their dependent objects. After initialization, these objects can be accessed from different parts of the application. Often, they are accessed by constructor injection into an object. The injected objects can have three different life-times:

- Singleton – only one instance of object exists within application,

- Transient – for every request for object DI container returns new instance of objects,
- Scoped – based on application scope, new instances of objects are returned.

The number of dependencies can increase rapidly in the .NET MAUI application. Often, the developer must manually add/remove these dependencies. The open-source library Scrutor[3] simplifies this, because it offers the possibility of scanning application assemblies. Therefore, it is possible to decorate common dependencies (for example, with a common interface), find them in assemblies, and then add them to the DI container as a group. The advantage of this approach is that when a new common dependency is added or removed, there is no need to manually edit the DI container. Also, Scrutor provides possibilities to add objects into DI container with different life-times. The example extension method, which used Scrutor to acquire the dependencies needed, is shown in Figure 4.7. In this example, I decorate the viewmodels that I want to instantiate as singletons with the `IViewModelSingleton` interface. After that, on the application startup, assemblies are scanned for these interfaces, and acquired objects (viewmodels) are added to the DI container with an extension method `AddSingletonVMs`.

```
1 public static void AddSingletonVMs(this IServiceCollection services)
2 {
3     services.Scan(selector => selector
4         .FromAssemblyOf<App>()
5         .AddClasses(filter =>
6             filter.AssignableTo<IViewModelSingleton>())
7         .AsSelfWithInterfaces()
8         .WithSingletonLifetime());
9 }
```

Figure 4.7: Extension method for DI container, which uses scanning abilities of Scrutor to acquire objects decorated with a common interface from application assemblies.

Community toolkit

.NET Community Toolkit is a collection of helpers and APIs that work for all .NET developers and are agnostic of any specific UI platform. The toolkit is maintained and published by Microsoft and is part of the .NET Foundation [22].

The community toolkit provides tools which are missing in .NET MAUI framework and can help and simplify the application development process. There are two important community toolkits available as NuGet packages that can be used in .NET MAUI:

- CommunityToolkit.Mvvm,
- CommunityToolkit.Maui.

`ComunnityToolkit.Mvvm`[4] is an MVVM modular library, which contains useful tools to build modern apps using the MVVM pattern. One of the most important features are

[3]Scrutor: `https://github.com/khellang/Scrutor`
[4]CommunityToolkit.Mvvm : `https://learn.microsoft.com/en-us/dotnet/communitytoolkit/mvvm/`

Roslyn source generators. Source generators can automatically generate code, reducing the boilerplate when writing code using the MVVM architecture. CommunityToolkit Mvvm contains implementations for:

- Notifying – implementation of the `INotifyPropertyChanged` interface, which serves to notify view, when data are changed in view model, and the other way around.
- Commanding – implementation of the `RelayCommand`, which encapsulates the request as an object. For example, RelayCommands can be used to handle button clicks on view in an MVVM way.
- Messaging – implementation of the `WeakReferenceMessenger`, which can be used to send messages between different viewmodels.

`CommunityToolkit.Maui`[5] is a collection of reusable elements for application development with .NET MAUI, including animations, behaviors, converters, effects, and helpers. It simplifies and demonstrates common developer tasks when building iOS, Android, macOS and WinUI applications.

TinyMvvm

TinyMvvm[6] is another library of tools for the MVVM applications. However, in contrast to the community toolkit, it contains useful tools for:

- navigation,
- passing objects as parameters in Shell navigation,
- and lifecycle events in viewmodels.

I found it really useful, as it streamlined my experience with Shell navigation and passing objects between viewmodels. What is more, I was able to override lifecycle events of .NET MAUI in viewmodels and, therefore, set the initialization code of different viewmodels based on the application lifecycle.

Other used libraries

Here is a list of other libraries and technologies used in my solution, which can be helpful in developing .NET MAUI applications:

- SQLite-net – open-source minimal library, which allows .NET applications to store data in SQLite databases,
- FluentValidation – library for building strongly-typed validation rules,
- LiveCharts2 – cross-platform charting library,
- Plugin.LocalNotification – plugin for showing notifications on mobile devices,
- Material Design Icons – icon pack.

[5] CommunityToolkit.Maui: `https://learn.microsoft.com/en-us/dotnet/communitytoolkit/maui/`
[6] TinyMvvm for MAUI: `https://github.com/dhindrik/TinyMvvm`

4.3 Integration and authorization of Kimai API

As mentioned in section 2.5 Kimai contains implemented REST API. This API contains the OpenAPI specification[7], which defines a standard for HTTP APIs which allows both humans and computers to discover and understand the capabilities of the service. This standardized specification can be used for auto-generation of the API client for different applications.

I used NSwag [8] toolchain, specifically the NSwagStudio application, for the generation of the API client for the stable version of Kimai 1.30.11. The API client uses .NET `HTTP client` for sending and receiving requests and `NewtfonSoft.Json` library for serializing and deserializing requests.

However, here the first problems occurred. After manually testing the generated code for different API endpoints, I found some inconsistencies. I found following problems between API generated DTO's and open specification models:

- using API endpoint `/api/timesheets/recent`, `Project` model has a customer field as integer, however, the API returns `Customer` type,
- inconsistency in `TeamMembership` model, where the user can be added to multiple teams in Kimai, but the API specification allows only one user to be in team.

As a result, I created issue [9] on the official Kimai GitHub repository with my findings. These problems were verified and fixed in the Kimai version v2.0.9. However, this fix influences only the version Kimai 2.0, therefore, I had to fix these problems on my own. I manually edited the affected models and fixed inconsistencies.

Authorization and login context

For testing purposes, authorization tokens were added to each request globally as `Default Request Headers`. These headers can be set only once during the initialization of the HTTP client. However, this approach is not feasible for the case where a new user or server is added to the application and new credentials are needed. As a result of this, custom `HTTP Message handler` is introduced.

The HTTP client factory (object, which handles initialization and deinitialization of HTTP clients) is initialized on application startup and added to the DI container. This HTTP client factory is provided with my custom HTTP request handler named `AuthHandler`. `AuthHandler` contains override of `SendAsync` method from HTTP client. This is main method, which is invoked every time when new HTTP request is sent. In this overridden method, authorization headers are added to each request. An example of the overridden `SendAsync` method is shown in the code snippet 4.8.

[7] OpenAPI specification: `https://swagger.io/specification/`
[8] NSwag toolchain: `https://github.com/RicoSuter/NSwag`
[9] GitHub issue with API inconsistencies: `https://github.com/kimai/kimai/issues/3923`

```
1  protected override async Task<HttpResponseMessage> SendAsync(request,
2  cancellationToken)
3 {
4     request.Headers.Add("X-AUTH-USER", _loginContext.UserName);
5     request.Headers.Add("X-AUTH-TOKEN", _loginContext.ApiPassword);
6     return await base.SendAsync(request, cancellationToken);
7 }
```

Figure 4.8: Simplified example of overridden `SendAsync` method in custom HTTP handler `AuthHandler`, where authorization headers are added to the request.

The credentials needed for authorization are provided by a special class called `LoginContext`. The `LoginContext` is a singleton object, which contains essential information about the user currently logged on a specific Kimai server. It is derived from `Observable object` (from the Community Toolkit), which means it implements the `INotifyPropertyChanged` interface. That means, when an authorization context is changed (for example, a user will log in or log out into application), the UI and viewmodels are notified about the change and are reinitialized accordingly.

The whole log-in use-case is glued together with the implementation of `Login service`. The simplified sequence diagram of login use-case with example of request is shown in Figure 4.9.

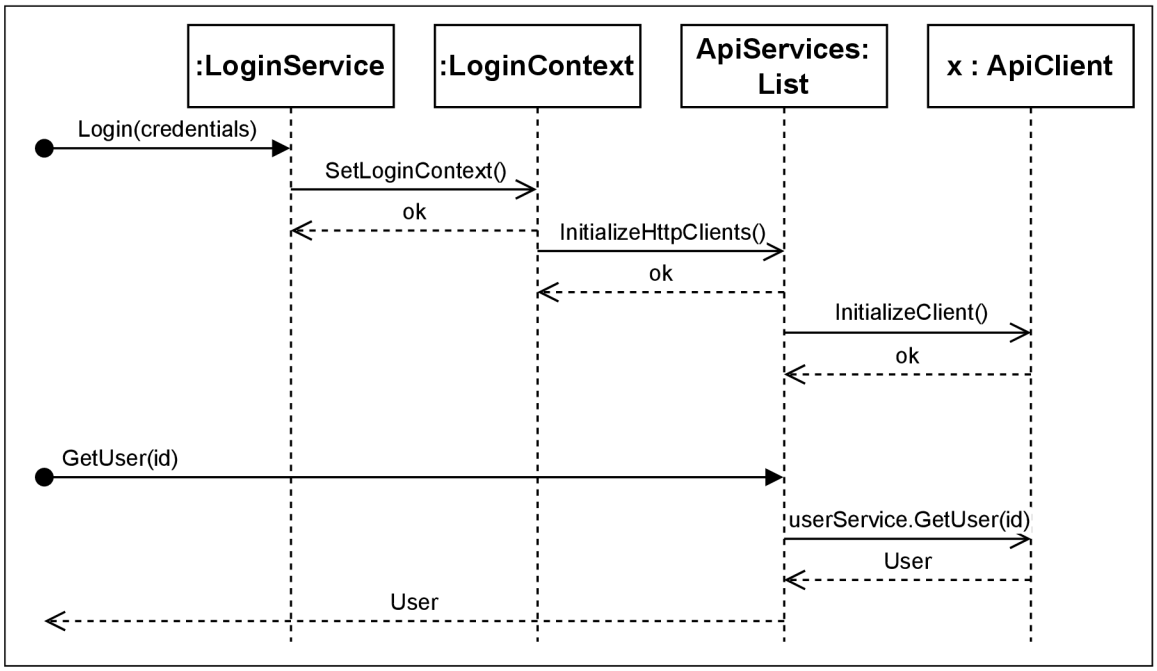


Figure 4.9: Simplified sequence diagram of login use-case, which shows how login context is created and how API services and specific instance `x` of API client is initialized. It also shows example how example request is handled.

The `Login service` provides methods for logging in and logging out users to the Kimai server. These methods initialize and deinitialize accordingly `LoginContext` with credentials

and also reinitialize HTTP clients that are used in API services (specific abstractions of the generated API client).

4.4 Navigation using Shell

.NET MAUI introduces the concept of Shell. The purpose of Shell is to reduce the complexity of the application development. It provides fundamental features like:

- a single place to describe the visual hierarchy of an app,
- a common navigation user experience,
- a URI-based navigation scheme,
- an integrated search handler.

There are multiple types of visual hierarchies available in Shell applications:

- Flyout menu – represents the expandable menu that contains flyouts,
- Tab bar – represents bottom tabs,
- Menu bar – presents a set of menus in a horizontal row, at the top of desktop application.

My first idea was to use `Flyout menu` as the main navigation pattern as shown in Section 3.6 about the final design. However, the first idea of a dynamic change of items in the menu based on the authorization state was wrong. First of all, when offline time tracking is introduced in the future, the user would not have access to the needed views. Also, when the user is offline, it would be convenient to still have some time-tracking information available. In addition, I ran into the .NET MAUI bug, when flyouts which are dynamically disabled are displayed incorrectly on Windows [10].

Based on all these factors mentioned, I have decided to use the `Tab bar` as a main navigation pattern in the mobile application and the `Flyout menu` for the desktop. I also decided that views are available when the application is offline and nothing is dynamically hidden. For now, when data cannot be accessed, the corresponding labels and alerts are shown. However, when time-tracking offline functionality will be added in the future, these views can be easily replaced to show cached offline data.

Routing based on viewmodels

The implemented application uses the URI navigation scheme provided by Shell to navigate between pages. This means that each page has its own unique URI identification. These identifications must be registered within the Shell application for the navigation to work correctly. The registration can be done anywhere in the application, however, it should be registered before the use of actual route. These URI identificators can be associated with views and viewmodels. Thanks to this association, specific URIs can be obtained based on a specific type of view or viewmodel. This is one of the approaches which can be used for navigation in .NET MAUI. Another approach is to navigate directly using URIs, but this approach is worse for code readability and maintainability. I created the `RouteModel` class, which contains:

[10] Github issue, which describes bug with flyouts on Windows: `https://github.com/dotnet/maui/issues/12159`

- Route (e.g. „//favourites/detail"),
- View (e.g. typeof(FavouritesDetailView)),
- ViewModel (e.g. typeof(FavouritesDetailView)).

Afterward, I implemented a Routing service, which stores registered route models in the List data structure called Routes. The route can be accessed using the publicly available generic method GetRouteByViewModel<T>(). When the viewmodel type is passed as a generic argument to this method, the corresponding route URI is returned, and navigation can be performed. The implementation of the GetRouteByViewModel method is shown in the code snippet 4.10.

```
1 public string GetRouteByViewModel<TViewModel>()
2 where TViewModel : IViewModel
3     => Routes.First(route =>
4         route.ViewModelType == typeof(TViewModel)).Route;
```

Figure 4.10: Implementation of generic method, which returns specific URI based on provided type of viewmodel for navigation purposes.

The Routing service is initialized and added to the DI container at the beginning of the application, where the routes are registered. Consequently, Routing service is injected in the viewmodels, where can be used for navigation purposes.

Navigation is performed by the Navigation helper from the TinyMvvm library (4.2). This is a wrapper around the Shell.Current.GoToAsync() method available in .NET MAUI. The Navigation helper provides the ability to pass objects as Navigation parameter. This simplifies the usage of navigation parameters compared to the default implementation. The example method for performing a navigation with a parameter is shown in the code snippet 4.11.

```
1 [RelayCommand]
2 async Task ShowDetail(TimesheetModel timesheet)
3 {
4     var route = rs.GetRouteByViewModel<TimesheetDetailViewModel>();
5     var wrapper = new TimesheetDetailWrapper(timesheet);
6     await Navigation.NavigateTo(route, wrapper);
7 }
```

Figure 4.11: Example of performing Shell navigation with Navigation helper from TinyMvvm library and passed object as a navigation parameter.

Multiplatform views

When developing a multiplatform application from a single-shared codebase, there is a question of whether to divide views for each platform. In some cases, it makes sense to divide the views. However, some specific views can be shared across platforms. .NET MAUI provides two main strategies for implementing multiplatform views:

- different views for each platform implemented in separate files,
- different views for each platform, where implementation specifics are mixed in one file.

The different views in separated files strategy

Views should be implemented in separate files when it makes sense to visualize certain UI elements differently on each platform for a specific use case. Let us take as example implementation of the main menu in my solution. It makes sense to show the menu differently on mobile and desktop platforms for a better user experience.

On the first hand, the real estate of the application is considerably smaller on the mobile platform than on the desktop platform. In addition, the user interacts with the application by touching. Therefore, it is a good choice to implement a main menu using a tab bar, which provides simple navigation across the application. This example is shown on figure 4.12.

On the other hand, when the user uses the desktop version of the application, often the mouse is used to interact with the GUI of the application. In this case, the static flyout menu is a better choice, because it shows all available options, with which a user can interact anytime with mouse. Moreover, desktops provide considerably larger real-estate, and user experience will be much better. This example is shown in Figure 4.13.

In this case, there are a lot of different multiplatform code, thus, separating views in two different files is a much better option than mixing everything together in one file.

The different views in one file strategy

This approach is beneficial when there are minor differences between desktop and mobile views. If there are many differences, the disadvantage of this strategy is that the unrelated code is coupled in one file and the complexity of the code increases. For example, when an application contains forms, the layout is often the same on both platforms. Sometimes, it is necessary to add or hide some elements for better UX. For instance, when user interacts with picker elements, it makes sense to introduce button on desktop view, which helps to visualize the picking use case, whereas in mobile picker element visualization is sufficient, and button is not needed.

To differ between platform-specific UI elements, .NET MAUI contains XAML markup extensions `OnIdiom` and `OnPlatform`. `OnIdiom` markup extension customizes the appearance of the user interface based on the idiom of the device the application is running on. It can be on Desktop, Mobile, Tablet, TV or Watch. Whereas the `OnPlatform` markup extension customizes the UI appearance per specific platform basis. It can be whether Android, iOS, MacCatalyst, Windows, or Tyzen. For example, the visibility of a specific element on different platforms can be customized using the markup extension `{OnIdiom Phone=False, Desktop=True}` to set the `IsVisible` property. This means that button will be hidden on mobile platform but will be visible on desktop platform.

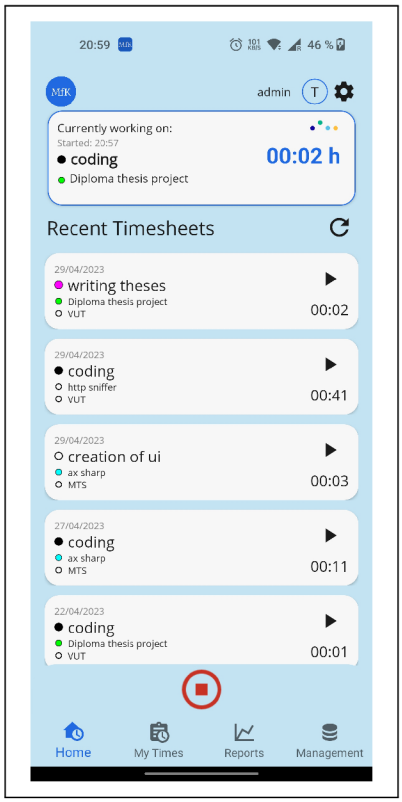


Figure 4.12: Implementation of main menu as a tab bar, which is better for UX on mobile platforms.

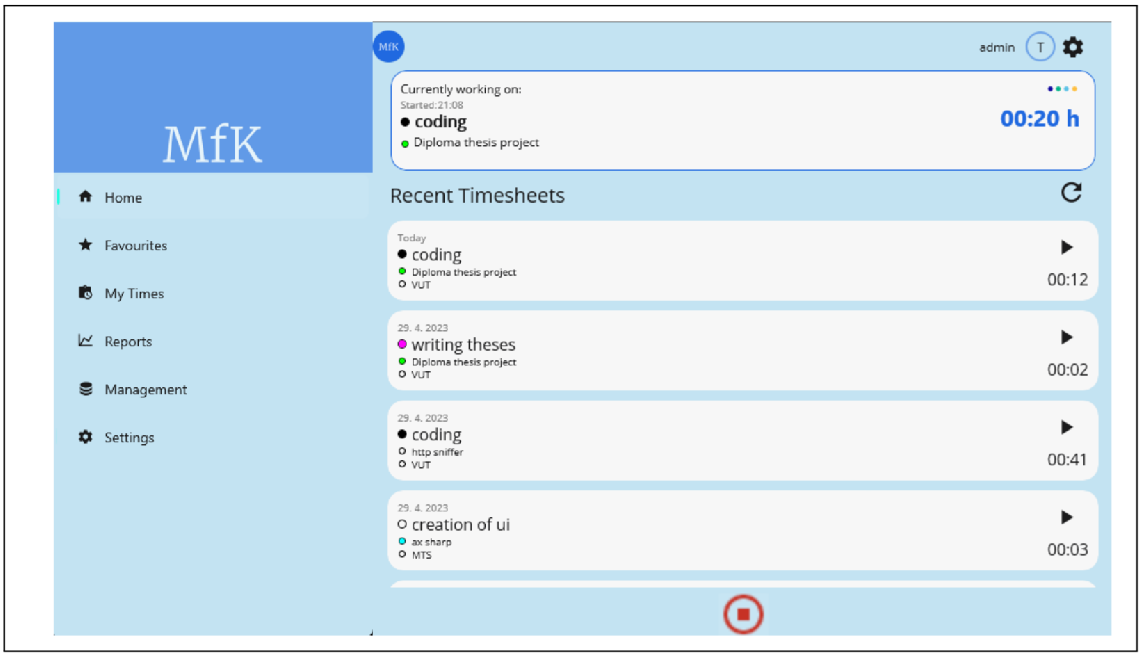


Figure 4.13: Implementation of main menu as a static flyout menu, which is better for UX on desktop platforms.

4.5 Viewmodels

Views and viewmodels are the core of the implemented solution. They are interconnected through data bindings and the `INotifyPropertyChanged` mechanism. Viewmodels contain implemented logic of the application, which is based of the user interaction on the views. The interactions from the views are handled using `Commands` 4.2, which are publicly available methods called from the UI. The overview of implemented viewmodels and their purpose is described below:

- **ViewModelBase** – base viewmodel, from which all other viewmodels inherit. It contains `LoginContext` 4.3 and fundamental services (like `Routing service`), which are available for all viewmodels.
- **HomeViewModel** – main viewmodel for the application, which is set as a binding context for `HomeView`. It incorporates main logic for time-tracking, notifications, and acquiring a recent timesheets.
- **TimesheetListViewModel** – logic for acquiring timesheets incrementally from server.
- **TimesheetDetailViewModel** – logic for editing, starting, and deleting a timesheet.
- **FavouritesListViewModel** – logic for acquiring favorite timesheets, which are saved in the local database.
- **FavouritesDetailViewModel** – logic for editing, starting, and deleting a favorite timesheet.
- **ServerListViewModel** – logic for acquiring servers added to application.
- **ServerDetailViewModel** – logic for editing, deleting, and connecting to a specific Kimai server.
- **ReportsViewModel** – serves for calculations needed for showing a graph reports.
- **ChooseItemViewModel** – it contains logic for choosing customer, project, and activity needed when creating a new timesheet.
- **SettingsViewModel** – handles the logic specific to the application's settings, such as dark or light mode.

There are two ways in which viewmodels can communicate with each other. The first way is using messages, which are provided by `WeakReferenceMessenger` available from the CommunityToolkit. For example, when an event of connection to Kimai server occurs, the `RefreshMessage` is sent to specific viewmodels to initialize new data from a Kimai server. The second way is usage of `Navigation parameter` from TinyMvvm library. This approach is used, when navigation between list views and detail views is performed, specifically when selected item from a list is passed to a detail viewmodel.

Chapter 5

Deployment and evaluation

This chapter contains an evaluation of the functionality of the implemented solution. It deals with the deployment, features, and usage of the implemented application. In addition, user feedback is discussed, and possible future extensions of the application are presented.

5.1 Deployment

The implemented solution is available as an open-source project called MAUI for Kimai (MfK) available in the Github repository [1]. This repository contains the source code of the application, a reference to the documentation page, and the released assets. The documentation page [2] contains API documentation of code and articles about architecture and multi-platform development.

Publishing and installing the application

Currently, MAUI for Kimai officially supports the following platforms:

- Windows 11 and Windows 10 version 1809 or higher,
- Android 5.0 (API 21) or higher.

Support for iOS and MacCatalyst is experimental. Since I do not own a Mac, I was unable to do extensive testing on these platforms. However, with the Mac provided from university, I was able to run the application on those platforms and test the fundamental features (like time-tracking). The iOS and Mac platforms need more testing and to publish experimental packages, an enrollment in Apple development program is needed. Therefore, experimental binaries for these platforms are not available. If someone wants to try MAUI for Kimai on the iOS/Mac platform, there is a need to build the application from source.

The application is released using a `Semantic Versioning`[3] versioning scheme. This means that the application is released in standardized versions, using the basic implementation of CI/CD. CI/CD is implemented using Github actions in the form of different workflows for building, testing, and publishing purposes. When a new version of the application is ready, the source code is tagged with its actual version by the `git tag` command. Afterward, tag is pushed to repository and publish workflow is triggered. Publish workflow

[1] MAUI for Kimai repository: `https://github.com/Specter-13/maui-for-kimai`
[2] MAUI for Kimai documentation page: `https://specter-13.github.io/maui-for-kimai/`
[3] Semantic Versioning: `https://semver.org/`

will build source code, test the code using unit tests, and create Github release [4] with actual version. Subsequently, the Powershell script `publish.ps1` is started, which will create optimized application binaries. Afterward, these binaries are uploaded to a corresponding Github release.

Currently, MAUI for Kimai can be installed on two platforms:

- Android – by downloading and manually installing signed `.apk` file from Github release,
- Windows – by downloading and manually installing `.msix` file from Github release.

5.2 Features overview and usage

In time of writing this text, MAUI for Kimai application has following features:

- Multiplatform support,
- Time-tracking possibilities and management of timesheets (start, stop, delete, recent, all),
- Quick start of timesheet,
- Possibility to set a timesheet as favorite (stored in local database),
- Multiple Kimai server and user support,
- Role-based time-tracking (possibility to customize time-tracking features based on user role),
- Integration with GitLab Kimai plugin GitLabBundle[5],
- Support for dark and light theme,
- Notifications for an active timesheet (in mobile platform),
- Forms validation.

There are also following limitations and known issues:

- Kimai v2.0 is not supported yet (as API specifications are not compatible),
- Limited support for iOS/Mac platforms (as I do not own Mac, I am not able to test all features properly),
- There can be some UI inconsistencies between dark and light mode,
- Missing detailed reports for teams.

[4] MAUI for Kimai Github releases: `https://github.com/Specter-13/maui-for-kimai/releases`

[5] GitLabBundle is Kimai plugin, which enables to assign timesheet to a specific GitLab issue, available at: `https://github.com/LibreCodeCoop/GitLabBundle`

After the first installation, the user is prompted to add a new instance of the Kimai server. This can be done in the `Management` tab, which contains the list of Kimai servers and the possibility of adding a new server. To add a new server, the user must set an API password in the Kimai web application. The API password is then used to log into the server. The log-in form also contains the possibility to set a server as default (on application startup automatic log-in will be initiated), the ability to use a GitLab plugin, and also the ability to override time-tracking permissions. The list of Kimai servers and an edit form of the server are shown in Figure 5.1.

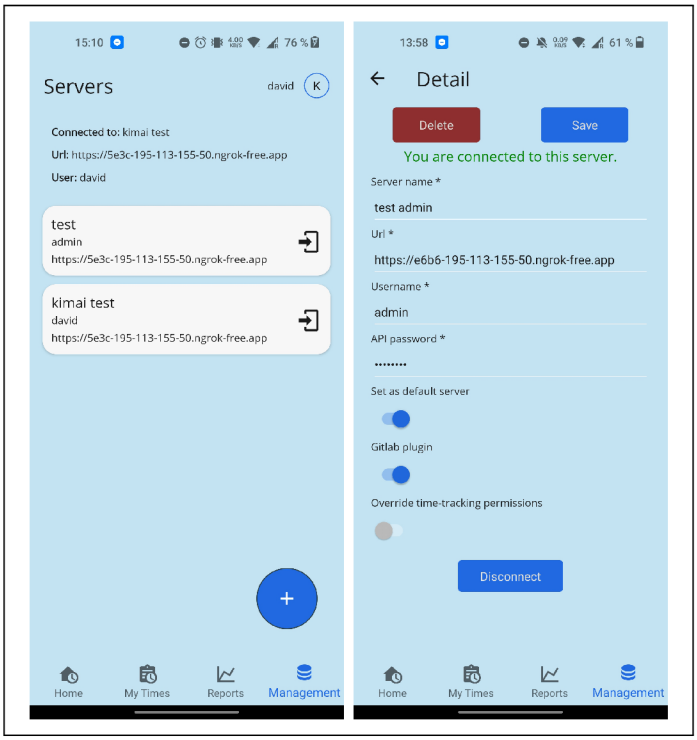


Figure 5.1: The server management views, where on the left is a list of currently added Kimai servers and on the right is a edit form of a specific server.

The home view is the most important part of the application. It contains a widget with information about today time-trackings, the list of unique recent timesheets with a quick-start functionality, and an ability to start a new timesheet. When the internet is not available or the user is not connected to the Kimai server, the empty view is shown. Also, when a time-tracking is active, the widget is used to show the most important information about currently tracked timesheet. The empty view and the difference between the home view with inactive and active timesheet are shown in Figure 5.2.

By clicking on the specific timesheet, the edit view is presented. Properties that can be edited depend on the user role. In addition, there is an ability to edit the timesheet duration, which is interconnected with the start and end times. Moreover, the user can select the customer, project, or activity from a dedicated view, with search possibilities. Edit view also contains buttons for starting the timesheet and setting it as a favorite. Favorite and all timesheets are located in the `My times` tab. The implementation of the views mentioned is presented in Figure 5.3.

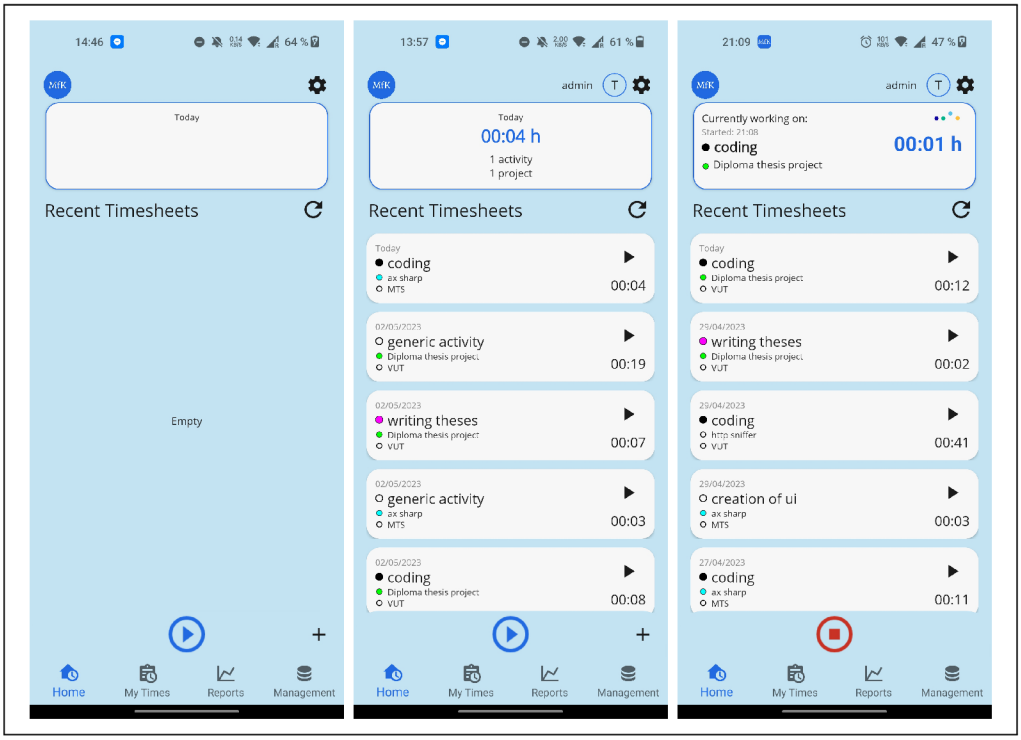


Figure 5.2: The home view of the application. On the left is presented an empty view. The middle and the right images show the difference, between an active and inactive mode.

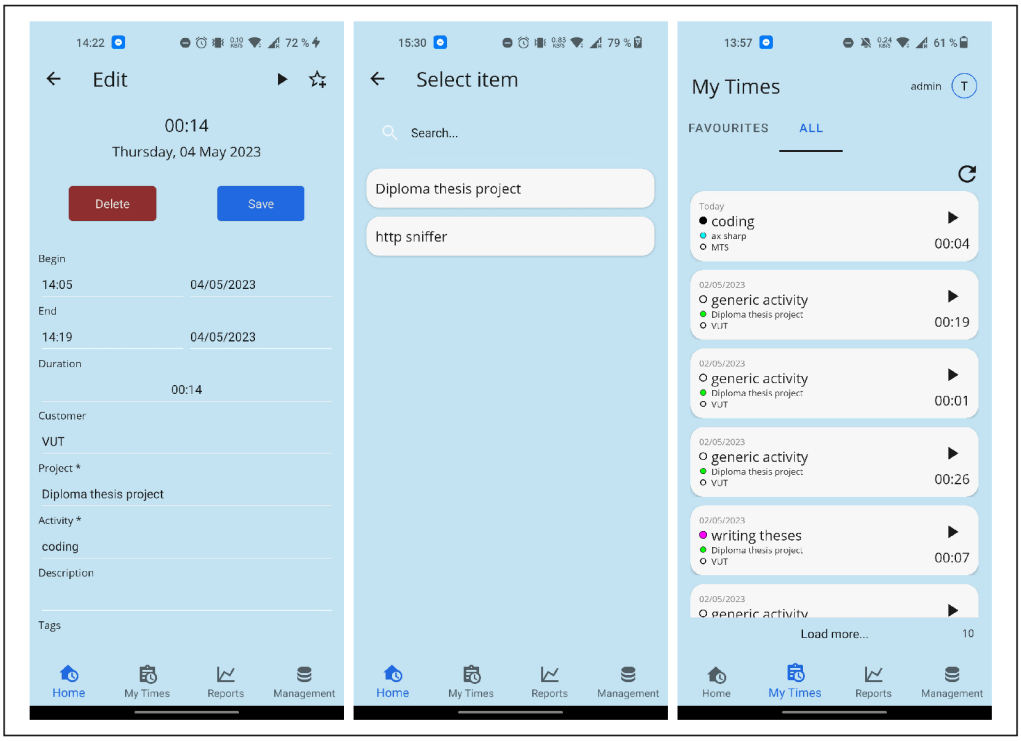


Figure 5.3: Views dedicated to timesheets, where on the left is an edit view, in the middle is a selecting view, and on the right is a list of all tracked times.

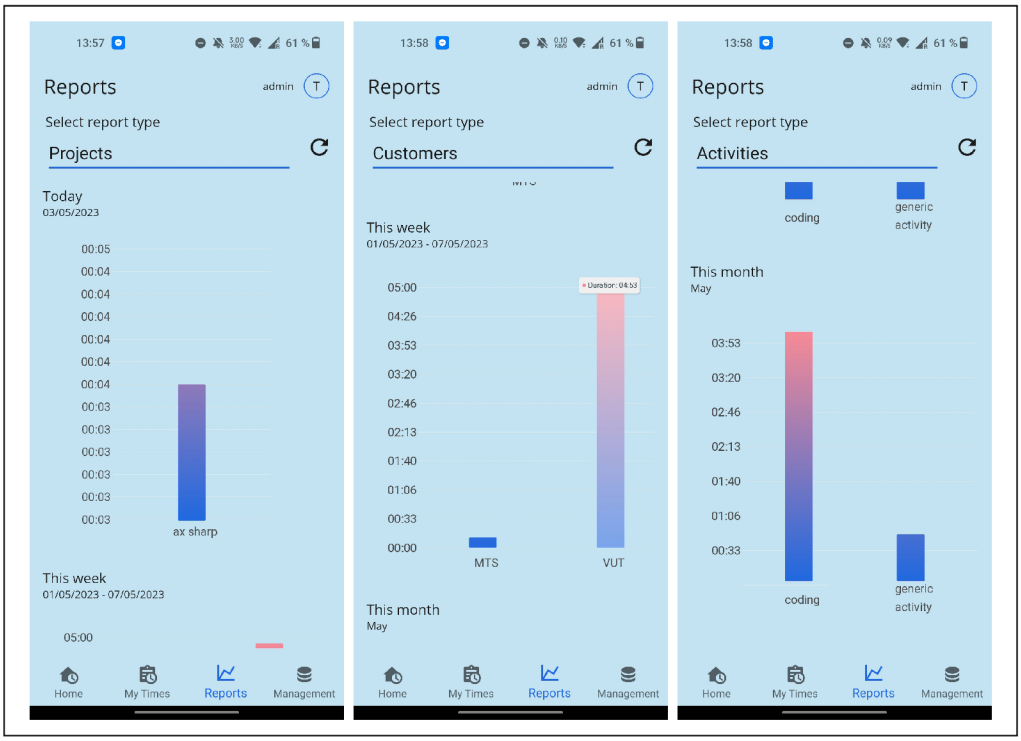


Figure 5.4: Implementation of reports, which can be shown for actual day, week and month.

MAUI for Kimai also contains a dedicated view for reports. Reports are shown as bar charts for the actual day, week, and month. The charts are fully responsive and supported on all platforms. Currently, reports are provided for projects, customers and activities. Figure 5.4 shows implemented reports for each entity.

5.3 Evaluation of the application and user feedback

The implemented application was tested publicly. Feedback was collected from the Kimai community and also from public testing of potential users. The application testing was conducted similarly to the usability testing of the interactive design of Section 3.2. Although the application is still in the alpha version, it is usable on a daily basis.

User testing

The testing was again conducted in the Faculty of Informatics of the Brno University of Technology. Desktop (Windows) and mobile (Android) versions were available for testing. The users were briefly instructed about the purpose of the application and shown the fundamentals of the application usage and features. The application was connected to a locally installed Kimai instance on my own laptop, which was available online through the Ngrok tunnel service [6]. The task of the users was to use the application and its main use-cases, like time tracking, accessing reports, or using favorite timesheets. The following shortcomings were identified due to user testing:

- **Insufficient visualization of active timesheet** – as time tracking is the main use case of the application, there were complaints about visualizing the currently active timesheet. Information about the current active timesheet should be prominent in the application UI. Also, it should be clearly visible that the activity is running. In addition, it was suggested to implement notifications that can show information on active timesheets anywhere in the mobile app.
- **Missing pause functionality** – since Kimai does not support pausing feature of timesheets, it was suggested to implement UI use-case, when timesheet will be paused inside Maui for Kimai application, and UI will provide quick restarting of paused timesheet.
- **Excessive error handling using toasts** – toasts are used as a main information element when an error occurs. However, it was pointed out that it is sometimes overused. For example, when the application is offline and the user is trying to refresh view, the toast is shown with a notification sound that data cannot be accessed. It would be better to show only the label with some basic information on the user interface than toast with a notification sound.
- **Small tap targets** – when an application was used, sometimes users tapped the wrong button or missed the button. For example, the timesheet list item contains a quick start feature in the form of a play icon. It was suggested that the size of this icon (or tapping context of button) should be increased.
- **Misleading naming** – tab named `Timesheets` was misleading, the suggestion was to rename the tab to `My Times`.
- **Missing team statistics** – team statistics was the most requested feature of the tested users. They would like to have an overview of statistics of the team, colleagues, and more.

[6] Cross-platform tunneling service Ngrok: `https://ngrok.com/`

- **Missing offline time-tracking** – application is unusable when internet connection is lost. It was suggested to implement an offline time-tracking feature in the future.
- **Omission of color indicators** – when a completely new timesheet was created, the color indicators were not present.

Thanks to user testing, multiple bugs were explored and subsequently fixed. One of the revealed bugs was the application behavior when a connection to the Kimai server was lost, while the login context was still set. The application was not able to handle thrown exception and it shutdown unexpectedly.

In general, users were satisfied with the usability of the application. They appreciated the quick-start functionality of the timesheet and also the ability to set the timesheet as favorite. They welcomed the interactive graphs and multiplatform support of the application.

At the time of writing this text, most of the problems found by a user testing were fixed. Insufficient visualization of an active timesheet was improved by using the home widget to provide essential information to the user about the active timesheet. Moreover, the notification service is implemented on a mobile platform, from which information about active timesheet is not used actively. The example of active timesheet information visualized in the home widget is shown in Figures 4.13 and 4.12. The naming of a specific elements was appropriately improved, and the tapping areas of a specific buttons were increased.

5.4 Feedback from the community and future development

The Maui for Kimai application has been available as a public open-source project since April 17th, 2023. Since then, development has continued and multiple releases have been published. I created posts on different .NET oriented Discord servers about my project and promoted my application to users of Kimai. I also created video [7] in which I present the features of the application.

As a result, the application was shared on the official Kimai social networks, specifically on Twitter [8] and Mastodon [9] presented as an interesting multi-platform project for the Kimai time tracker. I also received a response from the official Kimai maintainer, who praised the idea of an application and offered to help with testing on the iOS and MacOS platforms.

The application is constantly being improved. After the initial release, the main aim is now to fix some multiplatform inconsistencies, specifically for iOS/Mac platforms. Afterwards, I need to figure out a correct way to deploy these Apple-specific binaries to users for testing purposes. Another important task is to add support for a Kimai v2.0. This will require generating a new Kimai API client and implementing the needed services, which are used across the application. Also, the way of user authorization must be changed, as the API specification of the newer version is a little bit different from the older version. Moreover, there is a plan to add more features, like team-specific views, offline support for time-tracking, possibilities for management of activities, projects, customers, and so on.

On the basis of the user's feedback, I believe that this application has the potential to grow and be used within a Kimai community.

[7] MAUI for Kimai features overview video: `https://youtu.be/XoMrFvPtgJw`
[8] Twitter post: `https://twitter.com/kimai_org/status/1650599223597973504`
[9] Mastodon post: `https://phpc.social/@kimai/110255704127585195`

Chapter 6

Conclusion

The main goal of the diploma thesis was to create a multiplatform application in .NET MAUI, which should serve as a companion to open-source time-tracking software Kimai. Also, another goal was to investigate multiplatform development on .NET MAUI and specifically to provide a set of approaches for development on this framework.

The set goals were achieved, as the result is a multiplatform application called MAUI for Kimai, which is available and fully supported for Android and Windows platforms. The application is also functioning on iOS/Mac platforms, however, only for experimental purposes, as I have had limited access to Mac development tools. The application is available as an open-source project on GitHub[1], where it is available to the Kimai community. In addition, the .NET MAUI framework was thoroughly examined, and as a result, a set of approaches for multiplatform development was provided within the implementation chapter of the diploma thesis.

In the beginning, the problematics of multiplatform frameworks, user interface design, and the Kimai system were discussed. Afterward, the requirements for the application were gathered and the first design was created. The design was iteratively improved with the usability testing method. Multiple architectural approaches were discussed and the best architecture was chosen. Then the MAUI for Kimai application was implemented and published according to the requirements. In the end, feedback was collected from the community and from the public testing.

The application was shared with the Kimai community to gather feedback. I was contacted by an official Kimai maintainer, who praised the idea of a multiplatform application and offered help with testing. In addition, the application was shared on the official Kimai social pages on Twitter and Mastodon as an interesting new multiplatform project.

The development of the MAUI for Kimai application continues. Multiple versions have been released, which contain fixes and improvements based on user feedback. Now, I would like to focus on stabilizing the application for iOS/Mac and add more useful features, such as more detailed reports for teams, management possibilities for administrators, or ability for offline time-tracking.

I learned a lot during the development of the MAUI for Kimai application. However, at the time of writing this text, .NET MAUI is still a new technology and I think is not ready yet for more complex applications, as there are still a lot of issues, which limit specific development approaches and bring inconsistencies between platforms.

[1]MAUI for Kimai GitHub repository: `https://github.com/Specter-13/maui-for-kimai`

Bibliography

[1] Anastasiya Marchuk. *Native Vs Cross-Platform Development: Pros & Cons Revealed* [online]. 2022 [cit. 2022-10-19]. Available at: https://www.uptech.team/blog/native-vs-cross-platform-app-development.

[2] Andrade, P. R. M. d., Frota, O., Silva, F., Albuquerque, A. and Silveira, R. Cross Platform App: A Comparative Study. *Journal of Computer Science and Technology.* february 2015. DOI: 10.5121/ijcsit.2015.7104.

[3] Biørn Hansen, A., Majchrzak, T. A. and Grønli, T.-M. Progressive Web Apps: The Possible Web-native Unifier for Mobile Development. january 2017, p. 344–351. DOI: 10.5220/0006353703440351.

[4] Chris Sells, Ian Griffiths. *Programming WPF.* 2nd ed. O'Reilly Media, 2007. ISBN 9780596510374.

[5] Cordenne Brewster. *Native App Development Guide for 2022* [online]. 2021 [cit. 2022-10-29]. Available at: https://www.trio.dev/blog/native-app-development.

[6] Cordenne Brewster. *Web App Development in 2022: Everything You Need to Know* [online]. 2021 [cit. 2022-10-30]. Available at: https://www.trio.dev/blog/web-app-development.

[7] David Britch. *Behaviors* [online]. 2022 [cit. 2022-7-11]. Available at: https://learn.microsoft.com/en-us/dotnet/maui/fundamentals/behaviors.

[8] David Britch. *Style apps using XAML* [online]. 2023 [cit. 2023-05-09]. Available at: https://learn.microsoft.com/en-us/dotnet/maui/user-interface/styles/xaml.

[9] David Britch, Genevieve Warren, Minh Vuong. *App lifecycle* [online]. 2022 [cit. 2022-7-11]. Available at: https://learn.microsoft.com/en-us/dotnet/maui/fundamentals/app-lifecycle.

[10] David Britch, Ivan Gechev. *What is .NET MAUI?* [online]. 2022 [cit. 2022-7-11]. Available at: https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui.

[11] Frank Zammetti. *Practical React Native: Build Two Full Projects and One Full Game using React Native.* 1st ed. Apress, 2018. ISBN 9781484239384.

[12] Hubstaff. *The Benefits of Tracking Your Employees' Time* [online]. 2022 [cit. 2022-9-11]. Available at: https://hubstaff.com/time-tracking/benefits-time-tracking-employees#contents-category-2.

[13] Jason Taylor, Luke Parker. *Rules to Better Clean Architecture* [online]. 2023 [cit. 2023-4-24]. Available at: https://ssw.com.au/rules/the-main-principles-of-clean-architecture/.

[14] Jeffrey Rubin, Dana Chisnell. *Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests*. 2nd ed. Wiley, 2008. ISBN 978-0470185483.

[15] Jim Bennet. *Xamarin in Action*. 1st ed. Manning Publications Co., 2018. ISBN 9781617294389.

[16] Kanchan Yewale. *Cross Platform Application Development: Benefits and Technology* [online]. 2021 [cit. 2022-10-29]. Available at: https://www.clariontech.com/blog/cross-platform-application-development-benefits-and-technology.

[17] Kevin Papst. *Kimai - Free Time-Tracking App (open-source)* [online]. 2022 [cit. 2022-9-11]. Available at: https://www.kimai.org/.

[18] Marco L. Napoli. *Beginning Flutter*. 1st ed. John Wiley & Sons, Inc., 2020. ISBN 978-1-119-55082-2.

[19] Markus Egger. *XAML Anti-Patterns: Code-Behind* [online]. 2021 [cit. 2023-4-20]. Available at: https://codemag.com/article/1505101.

[20] Mascetti, S., Ducci, M., Cantù, N., Pecis, P. and Ahmetovic, D. *Developing Accessible Mobile Applications with Cross-Platform Development Frameworks* [online]. May 2020 [cit. 2022-10-29]. Available at: https://www.researchgate.net/publication/341396146_Developing_Accessible_Mobile_Applications_with_Cross-Platform_Development_Frameworks.

[21] Selva Ganapathy Kathiresan. *Xamarin Versus .NET MAUI* [online]. 2022 [cit. 2022-6-11]. Available at: https://www.syncfusion.com/blogs/post/xamarin-versus-net-maui.aspx.

[22] Sergio Pedri, Michael Hawker. *.NET Community Toolkit Introduction* [online]. 2022 [cit. 2023-4-24]. Available at: https://learn.microsoft.com/en-us/dotnet/communitytoolkit/introduction.

[23] Stonis, M. *Enterprise Application Patterns using .NET MAUI* [online]. 2022 [cit. 2022-04-21]. Available at: https://dotnet.microsoft.com/en-us/download/e-book/maui/pdf.

[24] Toggl. *Toggl track* [online]. 2023 [cit. 2023-1-16]. Available at: https://toggl.com/track/.

[25] Yuliia Fedyk. *7 Benefits of Flutter: Why Use Flutter for Your Next App* [online]. 2022 [cit. 2023-09-05]. Available at: https://inveritasoft.com/blog/7-flutter-advantages-a-perfect-platform-for-your-next-app.

Appendix A

Content of attached data medium

- **mauiforkimai/**
  - **src/** — source code of the application
  - **docs/** — source code of the documentation
  - **assets/** — figures used in README.md
  - **binaries/** — Android and Windows installation binaries
  - **README.md** — the essential information about project in english
- **latex/**
  - source code of text in latex
- **text/**
  - text of diploma thesis in pdf
- **README.txt**
  - fundamental information about needed tooling and building the application in slovak language