Aplikace pro evidenci spotřeby kávy zaměstnanců s využitím dotykového displeje a čtečky karet

Ondřej Černošek

Bakalářská práce
2022

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav počítačových a komunikačních systémů

Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Ondřej Černošek**
Osobní číslo:	**A19467**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Aplikace pro evidenci spotřeby kávy zaměstnanců s využitím dotykového displeje a čtečky karet**
Téma práce anglicky:	**Application for recording coffee consumption of employees using a touch screen and card reader**

Zásady pro vypracování

1. Popište současný stav technologií pro vývoj desktopových a webových aplikací.
2. Zvolte a popište vhodné technologie pro tvorbu dané aplikace.
3. Navrhněte aplikaci s využitím popsaných technologií, definujte funkční a nefunkční požadavky a případy použití.
4. Realizujte vývoj navržené aplikace a popište její klíčové části.
5. Implementujte komunikaci se čtečkou karet zaměstnanců.
6. Demonstrujte výsledky a formulujte závěr.

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

Seznam doporučené literatury:

1. FOWLER, Martin. Patterns of enterprise application architecture. Boston: Addison-Wesley, c2003. The Addison-Wesley Signature Series. ISBN 9780321127426.
2. GAMMA, Erich. Design patterns: elements of reusable object-oriented software. Boston: Addison-Wesley, 1995. ISBN 978-0201633610.
3. WATSON, Ben. Writing High-Performance .NET Code. 2nd. 2018. ISBN 978-0990583431.
4. BUONANNO, Enrico. Functional Programming in C#: How to write better C# code. Manning, 2017. ISBN 978-1617293955.
5. XU, Jack. Practical WPF Charts and Graphics. 2010th edition. Apress, 2011. ISBN 9781430223160.
6. YOSIFOVICH, Pavel. Windows Presentation Foundation 4.5 Cookbook. Packt Publishing, 2012. ISBN 9781849686228.
7. SKIENA, Steven. The Algorithm Design Manual. 2nd. Springer, 2008. ISBN 978-1849967204.

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

Datum zadání bakalářské práce: **3. prosince 2021**
Termín odevzdání bakalářské práce: **23. května 2022**

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

doc. Mgr. Milan Adámek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
garant oboru

Ve Zlíně dne 24. ledna 2022

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 11.05.2022

Ondřej Černošek v.r.
podpis studenta

ABSTRAKT

Cílem této bakalářské práce je vytvoření aplikace pro evidenci spotřeby kávy, umožňující snadné evidování spotřebovaných káv jednotlivými uživateli za pomoci RFID čtečky karet, a také snadné a rychlé vyúčtování díky webové aplikaci. V teoretické části se práce zabývá popisem aktuálně používaných technologií pro vývoj desktopových a webových aplikací. V praktické části jsou definovány funkční a nefunkční požadavky, případy užití, návrh databáze a také obsahuje popis nejdůležitějších částí aplikace.

Klíčová slova: Windows Presentation Foundation, Blazor, C#, XAML, EntityFramework

ABSTRACT

The aim of this bachelor's thesis is to create an application for recording coffee consumption, enabling easy registration of consumed coffee by individual users using an RFID card reader, as well as easy and fast billing thanks to a web application. The theoretical part deals with the description of currently used technologies for the development of desktop and web applications. The practical part defines the functional and non-functional requirements, use cases, database design and also contains a description of the most important parts of the application.

Keywords: Windows Presentation Foundation, Blazor, C#, XAML, EntityFramework

Tímto bych chtěl poděkovat vedoucímu mé práce panu Ing. Eriku Královi Ph.D. a panu Ing. Romanovi Slabému Ph.D. za jejich ochotu, rady, trpělivost a čas který mi během tvorby této práce věnovali.

OBSAH

ÚVOD

Cílem této bakalářské práce je vytvoření aplikace pro evidenci spotřeby kávy, umožňující snadné evidování spotřebovaných káv jednotlivými uživateli za pomoci RFID čtečky karet, a také snadné a rychlé vyúčtování díky webové aplikaci.

Na některých pracovištích si zaměstnanci sami organizují nákup kávy a evidují si spotřebovanou kávu a dělí si mezi sebou náklady. A právě těmto zaměstnancům usnadní aplikace pro evidenci spotřeby kávy práci se zaznamenáváním jednotlivých káv. Oproti zaznamenávání káv například na papír, při kterém je uživatel nucen najít své jméno na papíře a následně připsat čárku, umožňuje aplikace evidovat informaci o vypité kávě pouze přiložením karty zaměstnance ke čtečce karet a jedním kliknutím. Největší výhodou této aplikace není samotné evidování káv, ale následné vyúčtování, při kterém uživatel provádějící vyúčtování není nucen počítat ručně každou čárku každého zaměstnance, ale stačí mu vyplnit informace o vyúčtování ve webové aplikaci a ta se postará o zbytek, dokonce vypočítá cenu jedné porce kávy a rozešle uživatelům emaily s výší jejich dluhu a žádostí o uhrazení. Při vyúčtování je ve webové aplikaci také možnost zvolit soutěž pro daný měsíc, ve které je třem uživatelům s nejvyšším počtem zaznamenaných káv, udělena sleva.

Tato práce je rozdělena do jedenácti kapitol, přičemž prvních pět kapitol tvoří teoretickou část a zbylých šest kapitol tvoří část praktickou. První tři kapitoly se zabývají vývojem desktopových a webových aplikací, jejich výhodami, popisem vybraných frameworků a nástrojů pro vývoj. Čtvrtá kapitola je věnována návrhovým vzorům, jejich obecnému popisu a popisu vybraných návrhových vzorů. V kapitole páté je prostor věnovaný bezpečnosti aplikací a bezpečnostním rizikům. Kapitola šestá, která již patří do praktické části práce se zabývá návrhem aplikace, tvorbou funkčních a nefunkčních požadavků a případy užití aplikace. Následující čtyři kapitoly jsou věnovány tvorbě aplikace, popisu nejdůležitějších částí, databáze a bezpečnosti. Poslední jedenáctá kapitola demonstruje výsledky vývoje aplikace.

I. TEORETICKÁ ČÁST

1 VÝVOJ DESKTOPOVÝCH APLIKACÍ

Vývoj desktopových aplikací je proces, kdy vývojáři vyvíjí aplikaci, kterou bude možné používat pouze na desktopových zařízeních (případně noteboocích, …). Ve většině případů jsou tyto aplikace vyvíjeny pro operační systémy Windows, MacOS a Linux.

1.1 Proč psát desktopové aplikace

Podle statistik z roku 2020 68,1 % všech návštěv webových stránek bylo provedeno z mobilních zařízení a pouze 35,7 % z desktopových zařízení, nicméně pokud se podíváme na celkový čas strávený na webových stránkách zjistíme, že v tomto případě zastávají desktopová zařízení 53,3 % ve spojených státech amerických a 46,4 % celosvětově. Z čehož vyplývá, že lidé stále tráví spoustu času u desktopových zařízení. [1]

Na desktopových aplikacích mohou ovšem běžet jak již zmíněné webové aplikace, tak čistě desktopové aplikace. Proč tedy psát desktopové aplikace?

- Desktopové aplikace nabízejí lepší výkon při komplexních výpočtech, než nabízí webové aplikace.
- Používání více vláken procesoru je mnohem jednodušší a efektivnější.
- Programátor může snadno začít vyvíjet a testovat své algoritmy bez nutnosti řešení nastavování serverů, řešení připojení nebo kompatibility s prohlížečem.
- Debugování aplikací je mnohem efektivnější oproti debugování webových aplikací.
- Lepší přístup k periferiím desktopového zařízení (kamery, bluetooth zařízení, čtečky karet, …).[2]

1.2 Popis vybraných frameworků pro vývoj desktopových aplikací

V následujících odstavcích jsou popsány některé vybrané frameworky pro vývoj desktopových aplikací, při výběru frameworků byli zahrnuty nejlepší frameworky podle článku od Solace Infotech [3] a některé další frameworky.

1.2.1 Universal Windows Platform (UWP)

Je framework od Microsoftu umožňující vývoj aplikací na všech zařízeních běžících na operačních systémech Windows 10 a 11, operačních systémech pro XboX a také na operačním systému Windows Mixed Reality, což je operační systém, na kterém běží brýle pro rozšířenou realitu Microsoft HoloLens. UWP framework byl představen v roce 2015 společností Microsoft. Pro vývoj UWP aplikací se používá nejčastěji vývojové prostředí Visual Studio od společnosti Microsoft a aplikace jsou psány v jazycích C#, C++, Visual Basic, JavaScript (backend aplikace) a XAML, HTML, DirectX, WinUI (frontend aplikace). [2]

1.2.2 Windows UI Library 3 (WinUI 3)

Je nativní komponenta uživatelského rozhraní od Microsoftu, která je dodávána společně s Windows App SDK, které poskytuje jednotnou sadu API rozhraní a nástrojů pro tvorbu desktopových aplikací pro operační systém Windows 10 nebo novější. WinUI 3 je považován jako nástupce frameworku UWP a dá se použít pro tvorbu desktopových aplikací i pro tvorbu UWP aplikací. WinUI aplikace jsou psány v jazycích C#, C++, Visual Basic, JavaScript. [2]

1.2.3 Qt

Je open-sourceový framework pro tvorbu multiplatformních aplikací, mezi podporované platformy patří například Linux, MacOS, Windows, Android, iOS nebo třeba QNX. Pro vývoj aplikací se nejčastěji používá programovací jazyk C++, ale jsou podporovány i další jazyky jako třeba Python nebo Qt QML. Mezi hlavní výhody Qt frameworku patří rychlost a také již zmíněná multiplatformnost. [4]

1.2.4 Cocoa

Je framework pro tvorbu nativních objektově orientovaných aplikací pro MacOS, operační systém od společnosti Apple. Pro vývoj aplikací ve frameworku Cocoa se používá primárně vývojové prostředí Xcode, a jazyky Swift a objektové C, ale využívá se také jiných vývojových prostředí a programovacích jazyků. [5]

1.2.5 SwiftUI

Je framework určený pro vývoj aplikací na operační systémy společnosti Apple, jako jsou MacOS, iOS, watchOS a další. SwiftUI byl vyvinutý společností Apple v roce 2019. Pro vývoj aplikací v tomto frameworku se používá programovací jazyk Swift. Jednou z výhod SwiftUI je to, že je obsažen přímo v XCode, což je vývojové prostředí od společnosti Apple, mezi další výhody patří například to, že SwiftUI je nativní. [6]

1.2.6 Electronjs

Je open sourceový framework, který založil v roce 2013 Cheng Zhao, vývojář ve společnosti GitHub. Po třech letech vývoje, v roce 2016, bylo vydáno API 1.0. Pro vývoj v tomto frameworku se používají jazyky HTML, CSS, JavaScript. Narozdíl od výše zmíněných frameworků je Electron.js multiplatformní a podporuje vývoj na operační systémy Windows, MacOS i Linux. [7]

1.2.7 Swing

Je framework patřící pod Java Foundation Classes od společnosti Oracle, API poskytující grafické uživatelské rozhraní pro Java programy. stejně jako framework Electron.js je i Swing multiplatformní a umožňuje vyvíjet desktopové aplikace pro všechny operační systémy. Pro vývoj aplikací v tomto frameworku se používá jazyk Java. [8]

1.2.8 Windows Presentation Foundation (WPF)

Windows Presentation Foundation neboli WPF je vývojový framework používaný pro tvorbu desktopových aplikací, je součástí .NET framework. Ve WPF aplikacích je

uživatelské rozhraní aplikace vytvářeno v jazyce XAML, pro ostatní části aplikace se používá jazyk C#.

WPF framework byl vytvořen, aby nahradil WinForms. WPF nabíral na popularitě velmi pomalu kvůli své velikosti a také kvůli velkým odlišnostem od WinForm, které měl nahradit a na které byli vývojáři zvyklí. Již tehdy byla síla WPF evidentní, ale bylo těžké dostat své znalosti na vysokou úroveň.

Časem se ovšem věci změnily vývojáři si začali zvykat na změny a na nové způsoby, jak pracovat. XAML už nevypadal jen jako další věc potřebná k naučení, ale jak pohodlný a výkonný nástroj.

Poté přišli nové návrhové vzory, především Model-View-View Model (MWM), varianta jiného existujícího vzoru oddělujícího view a data (MVC a MVP), které učinili život jednodušším, ale co je důležitější, nastavili standard způsobu interakce View a dat.

WPF vedl také k vytvoření řady dalších technologií postavených na podobných principech, jmenovitě Silverlight (vývoj webového klienta pro různé prohlížeče v .NET), Windows Phone 7. (OS Phone od společnosti Microsoft, který používá variantu Silverlight) a v poslední době Windows 8 a Windows Phone 8 – vše postavené kolem podobné koncepty, jako je XAML, vlastnosti závislostí, šablony, styly a vazby – to vše ukazuje sílu a dopad WPF. [2][9]

Výhody WPF:

Široká integrace – Před WPF, vývojář, který chtěl pracovat kromě běžné 2D grafiky také s 3D, videem a dalšími, se musel naučit používat několik různých technologií, pro vše co chtěl dělat, a v rámci svého programu tyto technologie spojit. S příchodem WPF se situace změnila, jelikož WPF pokrývá tyto oblasti konzistentním programovacím modelem a úzkou integrací, při které je každý druh média renderován.

Hardwarová akcelerace – WPF je postaven na 3D, což znamená, že obsah ve WPF aplikaci (ať už 2D, 3D, text nebo grafika), je převeden na 3D trojúhelníky, textury a další Direct3D objekty a poté je renderován hardwarem. To znamená, že WPF využívají výkon hardwaru pro plynulejší grafiku a lepší výkon.

Deklarativní programování – Deklarativní programování není unikátní věc, kterou má pouze WPF, jelikož Win16/Win32 programy využívaly deklarativní zdrojové scripty k definování rozvržení dialogových oken a nabídek dávno před příchodem WPF. Ale WPF dostalo deklarativní programování na novou úroveň díky Extensible Application Markup Language (XAML). Kombinace WPF a XAMLu je podobná používání HTML k definování uživatelského rozhraní, jen s většími možnostmi. A tyto možnosti sahají až za hranice Uživatelského rozhraní. WPF využívá XAMLu například jako formátu dokumentů, nebo pro reprezentaci 3D modelů a mnoho dalšího.

Kompozice a přizpůsobení – ovládací prvky WPF lze spojovat různými způsoby. Uživatel například může vytvořit ComboBox a naplnit jej animovanými tlačítky nebo menu plné živých videoklipů, jednoduše řečeno uživatel nemusí psát spoustu kódu, aby mohl upravovat ovládací prvky. [9]

2 VÝVOJ WEBOVÝCH APLIKACÍ

Vývoj webových aplikací je proces, při kterém vývojáři vyvíjí aplikaci, která poběží na serveru a uživatelé k ní budou přistupovat skrze webový prohlížeč (například Google Chrome, Opera, Brave, …). Rozdíl mezi statickou webovou stránkou a webovou aplikací spočívá v její interaktivitě, webová aplikace dovoluje uživateli manipulovat s obsahem (například přidávat, mazat, nebo jinak upravovat data a obsah).

2.1 Proč psát webové aplikace

Jelikož webové aplikace není potřeba stahovat ani instalovat, tak nezabírají uživateli žádné místo na disku, a ani žádný čas potřebný k instalaci.

- Webové aplikace jsou většinou nezávislé na operačním systému uživatele a uživatel je může spustit na všech zařízeních které mají internetový prohlížeč, jak na desktopových zařízeních, tak i na mobilních zařízeních.
- Automatické aktualizace jsou další výhodou webových aplikací, jelikož se aplikace aktualizují na serveru, tak uživatel neztrácí čas ani úložiště svého zařízení, stahováním a instalováním aplikací.
- Webové aplikace jsou většinou hardwarově méně náročné než desktopové aplikace, jejich hardwarová náročnost je tak velká, jak velkou vyžaduje webový prohlížeč.

2.2 Aktuálně nejpoužívanější frameworky pro vývoj webových aplikací

V následujících odstavcích jsou popsány některé z nejpoužívanějších frameworků pro vývoj webových aplikací podle průzkumu Stack Owerflow Developer Survey [10] a framework Blazor.

2.2.1 React

Je open-sourceová JavaScriptová knihovna vytvořená společností Facebook v roce 2011 sloužící pro tvorbu uživatelského rozhraní.

React umožňuje vývojářům rozsáhlé webové aplikace, které mohou měnit data bez nutnosti znovu načítání webové aplikace. React framework by měl být rychlý, škálovatelný a jednoduchý. React pracuje v aplikaci pouze s uživatelským rozhraním, což v návrhovém vzoru MVC odpovídá View. React může být používán v kombinaci s jinými JavaScriptovými frameworky nebo knihovnami, jako je například Angular. [11]

2.2.2 Angular

Angular je open-sourceový aplikační framework, vyvinutý společností Google, založený na TypeScriptu, a nejčastěji se využívá pro vytváření single-page stránek. První verze byla vyvinuta společností Google v roce 2009 (AngularJs). V roce 2016 Google výrazně přepracoval AngularJs, aby splňoval zvyšující se požadavky moderních webů, a vydal Angular 2 (Angular). Přepracování to bylo znatelné, Angular narozdíl od svého předchůdce nevyužívá návrhový vzor Model-View-Controller, ale component-based architekturu. Dalším velkým rozdílem je samotný jazyk, Angular využívá TypeScript, zatímco AngularJs využívá JavaScriptu, a spoustu dalších rozdílů (mobilní podpora, Dependency Injection, …). [12]

2.2.3 Vue.js

Je open-sourceový framework pro tvorbu uživatelských rozhraní, vyvinutý Evanem Youem v roce 2014. Pro vývoj v tomto frameworku se používá především jazyk JavaScript.

Vue.js se primárně používá pro tvorbu webových rozhraní a single-page stránek, dá se ale také použít pro vývoj mobilních a desktopových aplikací. Díky HTML rozšířením a JavaScriptovému základu, které usnadňují práci s frameworkem Electron, je Vue.js oblíbeným nástrojem pro tvorbu uživatelských rozhraní. [13]

2.2.4 Asp.Net

Je open-sourceový framework vytvořený společností Microsoft v roce 2002, sloužící pro tvorbu webových aplikací a servis s pomocí .NETu. ASP.NET framework je vytvořen tak aby pracoval se standardním HTTP protokolem (protokol využívaný webovými aplikacemi). ASP.NET framework je také multiplatformní a umožňuje vyvýjet aplikace na MacOS,

Windows i Linux. ASP.NET podporuje pro vývoj webových aplikací mnoho programovacích jazyků, například C#, VB, F#, a další. [2]

2.2.5 Blazor

Je open-sourceový framework pro tvorbu Single-page webových aplikací, vytvořený společností Microsoft v roce 2018. Jméno Blazor je kombinací slov Browser a Razor. Z toho plyne, že místo vytváření vytvářet Razor Views na serveru, aby bylo možné prezentovat HTML prohlížeči, je Blazor schopen spouštět tyto Views na straně klienta.

Aplikace vytvořené v tomto frameworku jsou postavené na klasických věcech, které se používají pro tvorbu webových stránek, HTML, CSS a C#. Pro tvorbu aplikací využívá Blazor framework Razor komponenty, které jsou součástí ASP.NET, a které se skládají z kombinace HTML a C# kódu. Blazor aplikace se dělí na dva typy, podle typu hostingu: [2]

Blazor WebAssembly

Který je také nazývaný jako client-side hosting model, je model, při kterém aplikace díky WebAssembly běží přímo v prohlížeči. To znamená, že vše, co aplikace potřebuje (zkompilovaný kód, jeho závislosti a .NET), jsou staženy do prohlížeče.

Blazor Server

Který je také nazývaný jako server-hosting model, je model, při kterém aplikace běží na serveru. Mezi klientem a serverem je vytvořeno spojení pomocí knihovny SignalR. Ve chvíli, kdy nastane nějaká událost na straně klienta (například kliknutí na tlačítko), informace o události je poslána na server pomocí SignalR spojení. Server zpracuje událost a vytvoří změnu pro HTML soubor, zpět klientovi se neposílá celý HTML soubor ale právě jen tato změna. Prohlížeč poté zaktualizuje uživatelské rozhraní webové aplikace. Díky tomu, že se pro aktualizace uživatelského rozhraní používá pouze změna, tak je aplikace responsivnější.

3 TECHNOLIGIE PRO VÝVOJ DESKTOPOVÝCH A WEBOVÝCH APLIKACÍ

V následujících odstavcích jsou popsány vybrané technologie pro vývoj desktopových a webových aplikací.

3.1 Entity Framework

Je open-sourceový ORM framework, vydaný v roce 2008, určený pro .NET aplikace, který umožňuje vývojáři pracovat s databází pomocí .NET objektů, bez nutnosti věnovat se databázovým tabulkám. Stará se o připojování k databázím, provádění commandů, získávání výsledků dotazů a o automatické převádění těchto výsledků na objekty aplikace. [14]

Výhody/Vlastnosti Entity Frameworku:

- umožňuje používat LINQ dotazy pro získání dat z databáze. LINQ dotazy jsou převedeny do databázového dotazovacího jazyka (například SQL). Také umožňuje nepoužívat LINQ, a dotazovat se na databázi přímo pomocí SQL dotazů.
- umožňuje provádět migrace databáze jen za pomocí příkazů provedených v Package Manager Console, nebo v příkazovém řádku.

V následujících odstavcích jsou posány metody použití Entity Frameworku. Při popisu bylo vycházeno ze článku [15].

3.1.1 Code-First

Je způsob použití, při kterém vývojář vytvoří nejprve třídy v aplikaci na základě, kterých potom Entity Framework vygeneruje databázové tabulky. Výhodou této metody je snadné synchronizování prostředí a upravování entit, za pomocí Migrací.

Je to vhodný způsob práce s Entity Frameworkem pro vývojáře, kteří současně s aplikací vytváří i databázi.

3.1.2 Database-First

je způsob použití při kterém je nejprve vytvořena databáze se všemi tabulkami, na základě, které Entity Framework za pomocí nástroje Entity Framework Wizard vytvoří třídy v aplikaci.

Tento případ je vhodný například pokud je nějaká již existující databáze, pro kterou se najde nový způsob využití.

3.1.3 Model-First

je způsob použití založený na tvorbě databáze v uživatelském rozhraní. Jediné, co je potřeba je vytvořit model databáze za pomocí EDMX designu a poté jen nechat vygenerovat třídy a databázi.

Výhodou tohoto přístupu je, že vytvoření databáze je snadné rychlé a nejsou k němu potřeba žádné velké zkušenosti.

3.2 Programovací jazyky využívané pro vývoj desktopových/webových aplikací

V následujících odstavcích jsou popsány některé programovací jazyky využívané pro vývoj desktopových a webových aplikací.

3.2.1 C#

je open-sourceový objektově orientovaný programovací jazyk vytvořený společností microsoft v roce 2001. C# jako takový je také multiplatformní, aplikace vytvořené v .NETu mohou běžet v podstatě na jakémkoliv operačním systému (Windows, Linux, Mac, IOS, Android nebo dokonce na cloudu). Dalo by se říci, že je to univerzální jazyk, jelikož možnosti jeho využití jsou téměř neomezené, od klasických konzolových a windows aplikací, přes videohry, webové aplikace a nativní mobilní aplikace až po Azure cloudové aplikace, umělou inteligenci a strojové učení, a to není zdaleka vše. [16][17]

3.2.2 XAML

XAML je relativně jednoduchý a univerzální deklarativní programovací jazyk, původně navržený společností Microsoft pro framework WPF, schopný vytvářet a inicializovat objekty. XAML je v podstatě XML, ale se sadou pravidel.

Jelikož XAML je pouze mechanismus pro použití rozhraní .NET API, pokoušet se jej porovnat s HTML, Scalable Vector Graphics (SVG) nebo jinými jazyky je zavádějící. XAML se skládá z pravidel pro to, jak mají kompilátory zacházet s XML a z vlastních klíčových slov.[9][18]

3.2.3 HTML

Hypertext Markup Language neboli HTML je jedním ze základních jazyků pro tvorbu webových stránek. Skládá se z množství elementů, které říkají prohlížeči, jak zobrazit obsah webové stránky. [19]

3.2.4 CSS

Cascading Style Sheet neboli CSS je designový jazyk vytvořený s cílem zjednodušit proces designování webových stránek. CSS určuje to, jak budou vypadat HTML elementy na webové stránce. CSS je také úsporou času, jelikož můžete mít jen jeden CSS soubor pro více HTML stránek, zajistí se tím, že všechny stránky budou mít stejný styl. [20]

3.2.5 JavaScript

Je programovací jazyk umožňující implementaci dynamických prvků, jako jsou například animace, interaktivní prvky a další, na webové stránky. Byl vytvořen v roce 1995 pod jménem LiveScript (později byl přejmenován na JavaScript). Jednou z výhod JavaScriptu je také například možnost validovat údaje zadané uživatelem předtím, než jsou odeslány na server, což ušetří čas serveru, jelikož na něj putuje méně dotazů. [21]

3.2.6 TypeScript

Je silně typový objektově orientovaný programovací jazyk, vytvořený Andersem Hejlsbergem v Microsoftu. TypeScript je založený na JavaScriptu a jeho kód se kompiluje do JavaScriptu, což je jedna z výhod TypeScriptu, jelikož díky kompilaci do JavaScriptu může bez problémů využívat všechny JavaScriptové knihovny, nástroje a rozšíření. [22]

3.2.7 Swift

Je open-sourceový programovací jazyk vyvinutý v roce 2014 společností Apple, určený pro vývoj aplikací pro operační systémy od společnosti Apple (MacOS, iOS, iPadOS, tvOS a watchOS) a Linux. Swift je založený na objektovém C. Swift se snaží soustředit se na výkon, a rychlost aplikace, byl vyvinutý tak aby překonal svého předchůdce. [23]

3.2.8 SQL

Structured Query Language neboli SQL je databázový jazyk vytvořený pro získávání a upravování dat v relačních databázích. Všechny Relační databázové systémy (MySQL, MS Access, Oracle, MSSQL, …) využívají jazyk SQL. SQL umožňuje uživateli k datům v databázi, definovat tato data nebo s nimi různě manipulovat, také umožňuje uživateli vytvářet a mazat tabulky v databázi, a mnoho dalšího. [24]

4 NÁVRHOVÉ VZORY

Návrhové vzory nám pomáhají vytvořit přehledné, znovupoužitelné, upravitelné, a především obecné řešení našeho projektu. Většinou nám říkají, jak by na sobě měli být závislé jednotlivé objekty, jak by na sobě měli být závislé anebo jak by se měli chovat. Návrhové vzory nejsou striktně dané, není nutné se jimi řídit, ačkoliv je to vhodnější.

4.1 Typy návrhových vzorů

V následujícím textu jsou uvedeny typy návrhových vzorů podle knihy [25].

- **Vzory týkající se tvorby objektů (Creational)**
    - Jsou vzory zabývající se tvorbou objektů a tvorbou instance. Např Singleton Pattern.
- **Vzory týkající se struktury programu (Structural)**
    - Jsou vzory skládáním objektů a tříd do struktur, které by měli být flexibilní a efektivní.
- **Vzory týkající se chování (Behavioral)**
    - Jsou vzory zabývající se chováním tříd a tím, jak třídy mezi sebou komunikují.

4.2 MVVM

MVVM neboli Model – View – ViewModel je návrhový vzor navržený tak, aby oddělil programovou logiku od uživatelského rozhraní. Slouží k rozdělení kódu do modulů, se kterými se lépe pracuje, snadno se upravují nebo nahrazují jinými, kód je díky tomu přehlednější. [16][25]

Tento návrhový vzor se používá často pro WPF (Windows Presentation Foundation) aplikace.

V následujíchích odstavcích jsou popsány jednotlivé části tohoto návrhového vzoru. Při popisu těchto částí bylo vycházeno z článků [16][25].

4.2.1 Model

Jedná se o tzv. business logiku, která obsahuje data a jednotlivé funkce a výpočty. Model nesmí znát žádné informace o ovládacích prvcích z View.View reprezentuje uživatelské rozhraní v jazyce XAML.

4.2.2 ViewModel

Spojuje Model a View a drží si stav aplikace. Implementuje property a commandy na které se může View nabindovat, a upozorňuje View na změny pomocí notify eventů.

Command

Jedná se o metodu ViewModelu, kterou volá View, pokud uživatel stiskne tlačítko (nebo libovolný jiný ovládací prvek implementující rozhraní ICommandSource).

PropertyChange

ViewModel může pomocí eventu NotifyPropertyChange z rozhraní INotifyPropertyChange notifikovat View o tom, že se změnila jeho properta

4.2.3 View

Reprezentuje uživatelské rozhraní v jazyce XAML. Zodpovídá za definování struktury, rozložení a vzhledu, co uživatel vidí na obrazovce. Neměl by obsahovat logiku programu.

Bindování

Umožňuje upravovat vlastnosti objektu pomocí elementu v uživatelském rozhraní. Pomocí bindování můžeme snadno propojit view s modelem, aniž by a sebe museli mít přímou referenci.

4.3 MVC

MVC neboli Model-View-Controller je návrhový vzor navržený tak aby oddělil datový model aplikace (Model), uživatelské rozhraní (View) a řídící logiku (Controller). Slouží k rozdělení kódu do modulů, se kterými se lépe pracuje, snadno se upravují nebo nahrazují jinými, kód je díky tomu přehlednější.

Model

je hlavní část návrhového vzoru MVC a obsahuje data a jednotlivé funkce a výpočty. Model by neměl mít žádné údaje sloužící k zobrazování dat uživateli a měl by být nezávislý na uživatelském rozhraní.

View

slouží pro zobrazování dat modelu uživateli. Pro zobrazování dat může využívat grafů, tabulek, diagramů a spousty dalších věcí. Zodpovídá za vzhled aplikace. [25]

Controller

je článek mezi View a Modelem.Pokud View vyvolá nějakou událost, Controller na ni vhodně zareaguje (například zavoláním metody z modelu). [16]

4.4 SOLID

Jedná se o akronym počátečních písmen pěti základních principů, které definoval Robert C. Martin. Jedná se o tyto principy[26]:

S – Single responsibility principle – Každá třída má právě jednu zodpovědnost

O – Open/Closed principle – Funkcionalitu třídy lze rozšířit bez nutnosti její modifikace

L – Liskov substitution principle – Třídy musí být plně nahraditelné svými potomky

I – Interface segregation principle – Používat malá a úzce zaměřená rozhraní

D – Dependency inversion principle – Závislost na abstrakcích, nikoliv na implementacích

V následujících odstavcích jsou popsány jednotlivé principy, jak jsou definovány v těchto zdrojích [15][26][27].

4.4.1 Single responsibility principle

Jinými slovy, každá třída by měla mít zodpovědnost a tato zodpovědnost by měla být danou třídou či modulem plně pokryta. Za zodpovědnost se zpravidla považuje nějaká rozumně jednoduchá a oddělená funkcionalita. Díky tomu je pro vývojáře jednodušší při úpravách software najít místa vyžadující změnu.

Častým argumentem proti SRP je, že systém složený z více tříd je složitější. Ve výsledku však takovýto systém neobsahuje více metod a vlastností než systém složený z několika velkých tříd – jsou pouze rozděleny do menších celků. Samozřejmě je však důležité, jako vždy, nic nepřehánět.

4.4.2 Open/Closed principle

Třídy by měly být psány tak, aby jejich funkčnost šla rozšířit bez nutnosti je modifikovat. Rozšíření funkčnosti by mělo být možné pouze tím, že přidáme nový kód bez nutnosti zasahovat do kódu existujícího. Díky tomu minimalizujeme možnost, že při úpravách neúmyslně poškodíme jiné části aplikace spoléhající na původní kód. Většina realizací tohoto principu spočívá v použití dědičnosti.

4.4.3 Liskov substitution principle

Pokud nějaký kód používá bázovou třídu, musí fungovat i v případě, že místo bázové třídy použijeme jakoukoliv z jejích podtříd. Kódu používajícímu bázovou třídu tedy musí být jedno, zda dostane bázovou třídu nebo její podtřídu. Všechny podtřídy musí být z pohledu uživatele bázové třídy vzájemně zaměnitelné. Uživatel bázové třídy také nesmí být nucen měnit svoje chování podle typu obdržené podtřídy.

4.4.4 Interface segregation principle

Princip oddělení rozhraní říká, že každé rozhraní by mělo být co nejmenší možné a třídy by neměly být nuceny používat rozhraní, která nepoužívají. Pokud tedy nějaké rozhraní přesáhne rozumnou velikost, musí se rozdělit do několika dalších a užších rozhraní. Potom se touto změnou zasažené třídy přepracují tak, aby implementovaly jen minimální potřebnou podmnožinu původních rozhraní.

4.4.5 Dependency inversion principle

Jeho dodržování vede k výrazné redukci závislostí v kódu. Mnoho závislostí na konkrétních implementacích je nahrazeno závislostí na společném abstraktním rozhraní. Konkrétní

implementace se mění mnohem častěji než abstraktní rozhraní. Nikdy bychom neměli záviset na něčem, co se často mění, protože to pak může vyžadovat, aby se závislý kód také měnil.

Pokud třídy závisí na abstraktních rozhraních a třídách, je velice snadné nahradit jednu konkrétní implementaci jinou. Kód je tedy velice flexibilní a mnoho změn lze provést pouze výměnou jedné konkrétní implementace za jinou. Tato výhoda je pak velmi výrazná v případě, že třídy testujeme a potřebujeme sledovat jejich chování. Můžeme jim pak jednoduše předat takové implementace, které budou jejich chování kontrolovat.

Pokud třída závisí pouze na abstraktních rozhraních, její znovupoužitelnost jinde je mnohem jednodušší. Třídu je možné bez úprav použít v prostředí tvořeném odlišnými implementacemi rozhraní, na kterých závisí.

Existují místa, kde se závislostí na konkrétní implementaci zbavuje těžko. Typickým případem je vytváření instancí. Pokud chceme vytvořit instalaci, nelze použít pouze abstraktní rozhraní, musíme sáhnout po konkrétní implementaci. Řešením je v tomto případě návrhový vzor Abstract Factory. Třída bude záviset pouze na abstraktní továrně vytvářející instance, aniž bychom museli znát jejich konkrétní implementaci.

5 BEZPEČNOST

Zabezpečení aplikací je důležité, obzvláště u webových aplikací připojených k internetu, jelikož jsou vystaveny možnému útoku odkudkoliv.

5.1 Nejčastěji vyskytující se bezpečnostní rizika

V následujících odstavcích jsou popsány některé z nejčastěji se vyskytující se bezpečnostních rizik podle článku ze stránky OWASP [28]. Při popisu bezpečnostních rizik je vycházeno ze článků [29][30].

5.1.1 Cross site scripting (XSS)

Je bezpečnostní riziko, při kterém útočník vloží do webové aplikace na straně klienta scripty, díky kterým se může dostat k citlivým údajům, nebo se díky nim může pokusit o to aby mu jiný uživatel sdělil své citlivé informace.

Tomuto bezpečnostnímu riziku se dá předcházet používáním modernějších frameworků, které v sobě nabízejí nějakou úroveň ochrany, jako například kódování výstupů.

5.1.2 SQL Injection (SQi)

Je metoda, při které útočník hledá slabiny ve způsobu, jakým databáze vykonává vyhledávací dotazy, a následně se snaží dostat k citlivým informacím.

SQi se dá předcházet použitím parametrizovaných SQL dotazů, které využívají parametry namísto stringů.

5.1.3 Denial of service (DoS) a Distributed denial of service (DDoS) útoky

Je útok, při kterém se útočníci snaží přetížit server, při přetížení serveru následně může dojít se zpomalení odpovědí serveru nebo dokonce ke znemožnění přístupu uživatelům.

DDoS útokům se dá předcházet zabezpečením serverů, kontrolou vstupů (omezení maximálního počtu dotazů, omezení maximální velikosti nahrávaných souborů), kontrolou přístupu a dalšími způsoby.

5.1.4 Broken access control

Je bezpečnostní riziko, při kterém je útočníkovi umožněno získat neautorizovaný přístup a práva (útočník se tváří jako administrátor).

Tomuto riziku se dá předcházet využíváním přidělování přístupu na základě uživatelských rolí, pravidelným kontrolováním a testováním autorizace.

5.1.5 Server side request forgery

Je bezpečnostní riziko, které může nastat, pokud webová aplikace nevaliduje URL vložené uživatelem dříve, než jsou stažena data ze zdroje.

Tomuto riziku se dá předcházet například validací URL vložených uživatelem.

II. PRAKTICKÁ ČÁST

6 NÁVRH APLIKACE

V následující části jsou popsány funkční/nefunkční požadavky a případy užití.

6.1 Funkční požadavky

- Systém bude uživateli umožňovat se přihlásit do aplikace.
- Systém bude umět komunikovat se čtečkou RFID karet.
- Systém bude uživateli umožňovat zaznamenávat počet vypitých káv.
- Systém bude zobrazovat tabulku Top 10 uživatelů s nejvyšším počtem vypitých káv.
- Systém bude umožňovat zobrazení údajů o vypitých kávách.
- Systém bude umožňovat filtrování dat o vypitých kávách.
- Systém bude umožňovat generování reportů vybraných dat.
- Systém bude umožňovat vypočítání ceny vypité kávy uživatele za zvolený měsíc.
- Systém bude umožňovat rozesílání e-mailů s vyúčtováním.

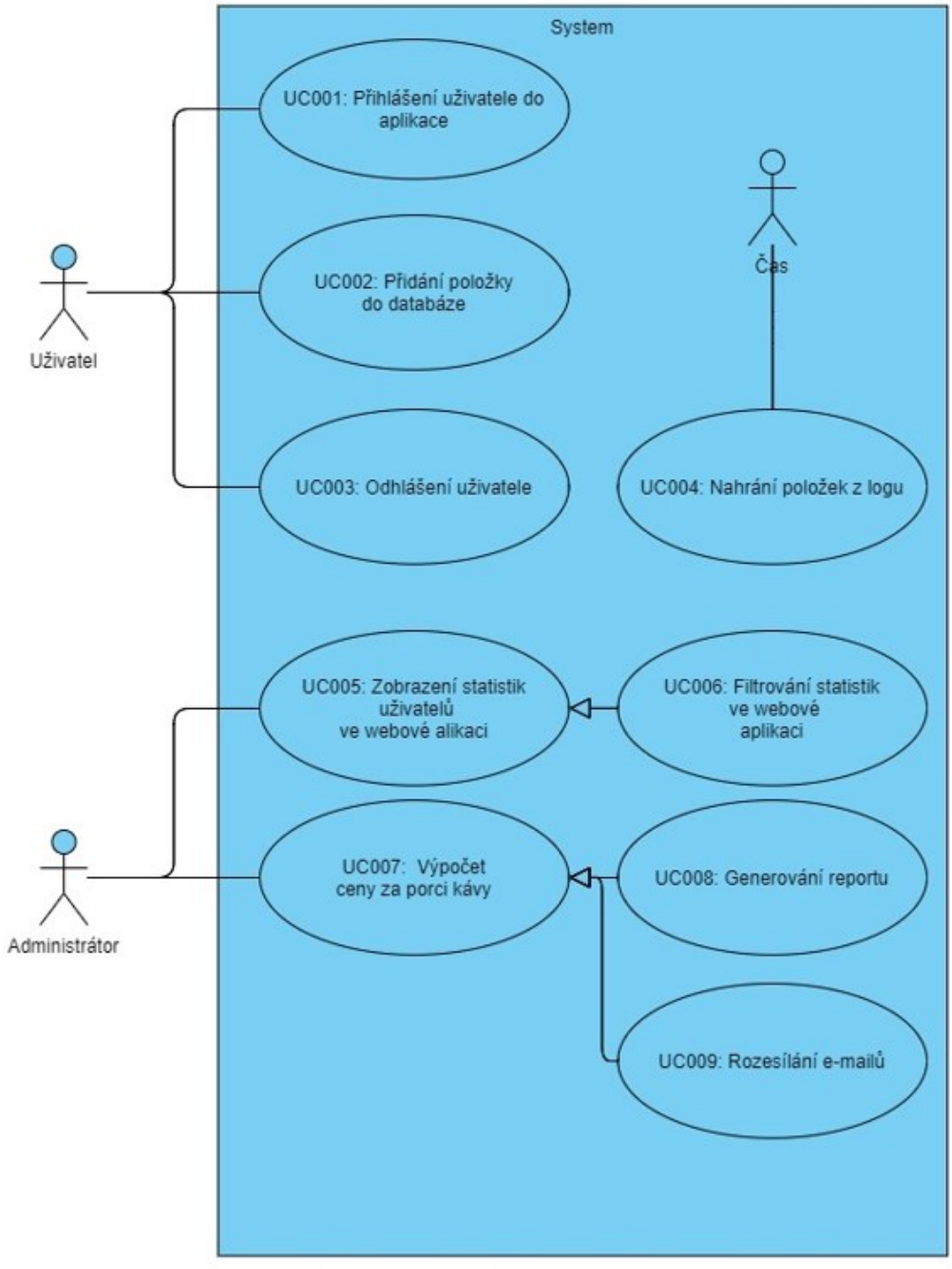
6.2 Nefunkční požadavky

- Aplikace musí umět pracovat offline.
- Systém by měl mít intuitivní a uživatelsky přívětivé uživatelské rozhraní.
- Systém musí být plynulý a rychle reagovat.
- Odezva serveru musí být menší než 2 sekundy.

6.3 Případy užití

V následující části tohoto dokumentu se nachází diagram případů užití a také jednotlivé případy užití aplikace.

6.3.1 Diagram Případů užití



Obrázek 1 – Digram případů užití

Tabulka 1 – Scénář pro přihlášení uživatele

Název: Přihlášení uživatele do aplikace		
ID: UC001		
Charakteristika: Případ popisující přihlášení uživatele do aplikace		
Primární aktér: Uživatel		
Vedlejší aktéři: Žádní		
Vstupní podmínky: Uživatel má kartu zaměstnance, nachází se na výchozí obrazovce, systém je připojený k databázi.		
Výstupní podmínky: Uživatel bude přihlášen a přesměrován na hlavní stránku aplikace.		
Hlavní scénář:		
Krok	Aktér/Systém	Popis
1	Uživatel	Uživatel přiloží kartu zaměstnance ke čtečce karet
2	Systém	Aplikace vyhledá uživatele v databázi pomocí kódu karty -> Alternativní scénář UC001a.
3	Systém	Po úspěšném nalezení uživatele v databázi aplikace uživatele přesměruje na hlavní obrazovku.
Alternativní scénáře: UC001a – Nedostupná databáze		

Tabulka 2 – Alternativní scénář pro přihlášení uživatele

Přihlášení uživatele do aplikace – Alternativní scénář:		
ID: UC001a		
Charakteristika: Alternativní případ popisující přihlášení uživatele v případě že je databáze nedostupná		
Alternativní scénář:		
Krok	Aktér/Systém	Popis
1	Systém	Aplikace přesměruje uživatele na offline verzi úvodní obrazovky.

Tabulka 3 – Scénář popisující přidání položky do databáze

Název: Přidání položky do databáze		
ID: UC002		
Charakteristika: Případ popisující přidání položky do databáze		
Primární aktér: Uživatel		
Vedlejší aktéři: Žádní		
Vstupní podmínky: Uživatel se nachází na hlavní obrazovce, systém je připojený k databázi.		
Výstupní podmínky: Uživatel bude odhlášen a přesměrován na úvodní stránku.		
Hlavní scénář:		
Krok	Aktér/Systém	Popis
1	Uživatel	Klikne na tlačítko s položkou, kterou chce přidat do databáze
2	Systém	Aplikace uloží položku do databáze -> Alternativní scénář UC002a.
3	Systém	Po uložení přesměruje aplikace uživatele na úvodní obrazovku.
Alternativní scénáře: UC002a – Nedostupná databáze		

Tabulka 4 – Alternativní scénář popisující přidání položky do databáze

Přihlášení uživatele do aplikace – Alternativní scénář:		
ID: UC002a		
Charakteristika: Alternativní případ popisující přidání položky do databáze v případě že je databáze nedostupná		
Alternativní scénář:		
Krok	Aktér/Systém	Popis
1	Systém	Aplikace místo uložení položky do databáze zaloguje informace o uživateli a položce do souboru pro pozdější nahrání do databáze.
2	Systém	Po zalogování přesměruje aplikace uživatele na úvodní obrazovku.

Tabulka 5 – Scénář popisující odhlášení uživatele

Název: Odhlášení uživatele		
ID: UC003		
Charakteristika: Případ popisující odhlášení uživatele		
Primární aktér: Uživatel		
Vedlejší aktéři: Žádní		
Vstupní podmínky: Uživatel se nachází na hlavní obrazovce.		
Výstupní podmínky: Uživatel bude odhlášen a přesměrován na úvodní stránku.		
Hlavní scénář:		
Krok	Aktér/Systém	Popis
1	Uživatel	Klikne na tlačítko „Log Out“
2	Systém	Aplikace uživatele přesměruje na úvodní obrazovku.

Tabulka 6 – Scénář popisující nahrání položek z logu

Název: Nahrání položek z logu		
ID: UC004		
Charakteristika: Případ popisující nahrání položek z logu do databáze		
Primární aktér: Čas		
Vedlejší aktéři: Žádní		
Vstupní podmínky: Systém je připojený k databázi.		
Výstupní podmínky: Data v databázi budou aktuální.		
Hlavní scénář:		
Krok	Aktér/Systém	Popis
1	Systém	Systém zkontroluje, jestli jsou v log souboru nějaké záznamy.
2	Systém	Pokud jsou v log souboru nějaké záznamy, tak je aplikace nahraje do databáze.
3	Systém	Aplikace vymaže záznamy z log souboru.

Tabulka 7 – Scénář pro zobrazení statistik

Název: Zobrazení statistik uživatelů ve webové aplikaci		
ID: UC005		
Charakteristika: Případ popisující zobrazení statistik všech uživatelů ve webové aplikaci		
Primární aktér: Administrátor		
Vedlejší aktéři: Žádní		
Vstupní podmínky: Administrátor se nachází na úvodní obrazovce webové aplikace.		
Výstupní podmínky: Administrátorovi je zobrazena stránka se záznamy.		
Hlavní scénář:		
Krok	Aktér/Systém	Popis
1	Administrátor	Klikne na tlačítko „Calculator“
2	Systém	Aplikace přesměruje administrátora na obrazovku se statistikami uživatelů.

Tabulka 8 – Scénář pro filtrování statistik

Název: Filtrování statistik ve webové aplikaci		
ID: UC006		
Charakteristika: Případ popisující filtrování statistik všech uživatelů ve webové aplikaci		
Primární aktér: Administrátor		
Vedlejší aktéři: Žádní		
Vstupní podmínky: Administrátor se nachází na obrazovce se statistikami uživatelů.		
Výstupní podmínky: Administrátorovi jsou zobrazeny záznamy podle zvolených kritérií.		
Hlavní scénář:		
Krok	Aktér/Systém	Popis
1	Administrátor	Nastaví hodnoty jednotlivých filtrů.
2	Administrátor	Klikne na tlačítko „Refresh“
3	Systém	Aplikace zobrazí vybrané statistiky.

Tabulka 9 – Scénář pro výpočet ceny

Název: Výpočet ceny za porci kávy		
ID: UC007		
Charakteristika: Případ popisující výpočet ceny za porci kávy		
Primární aktér: Administrátor		
Vedlejší aktéři: Žádní		
Vstupní podmínky: Administrátor se nachází na obrazovce se statistikami uživatelů, má zobrazeny záznamy, pro které chce vypočítat cenu.		
Výstupní podmínky: Systém vypočítá cenu za porci, a následně k jednotlivým uživatelům vypočítá cenu odpovídající počtu vypitých káv.		
Hlavní scénář:		
Krok	Aktér/Systém	Popis
1	Administrátor	Nastaví celkovou částku utracenou za nákup kávy.
2	Administrátor	Klikne na tlačítko „Refresh“
3	Systém	Aplikace vypočítá cenu za 1 porci, a celkovou útratu jednotlivých uživatelů.
4	Systém	Aplikace zobrazí cenu za porci kávy, a u jednotlivých uživatelů zobrazí výši jejich útraty.

Tabulka 10 – Scénář Generování reportu

Název: Generování reportu		
ID: UC008		
Charakteristika: Případ popisující generování reportu.		
Primární aktér: Administrátor		
Vedlejší aktéři: Žádní		
Vstupní podmínky: Administrátor se nachází na obrazovce se statistikami uživatelů, má zobrazeny záznamy, a má vypočítanou výši útraty jednotlivých uživatelů.		
Výstupní podmínky: Systém vytvoří excel soubor se jmény uživatelů a výší jejich útraty.		
Hlavní scénář:		
Krok	Aktér/Systém	Popis
1	Administrátor	Klikne na tlačítko „Generate“
2	Systém	Aplikace vygeneruje report.

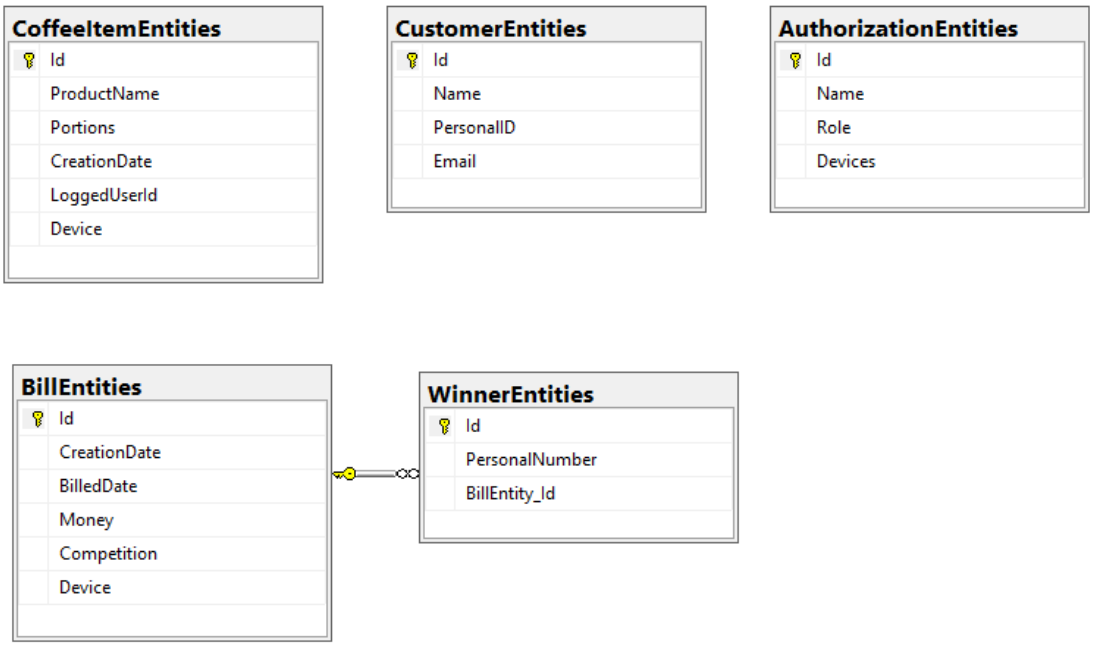
Tabulka 11 – Scénář rozesílání e-mailů

<table>
<tr><td colspan="3">Název: Rozesílání e-mailů</td></tr>
<tr><td colspan="3">ID: UC009</td></tr>
<tr><td colspan="3">Charakteristika:
Případ popisující rozesílání e-mailů.</td></tr>
<tr><td colspan="3">Primární aktér:
Administrátor</td></tr>
<tr><td colspan="3">Vedlejší aktéři:
Žádní</td></tr>
<tr><td colspan="3">Vstupní podmínky:
Administrátor se nachází na obrazovce se statistikami uživatelů, má zobrazeny záznamy, a má vypočítanou výši útraty jednotlivých uživatelů.</td></tr>
<tr><td colspan="3">Výstupní podmínky:
Systém rozešle emaily uživatelům se žádostí o zaplacení útraty a s výší jejich útraty.</td></tr>
<tr><td colspan="3">Hlavní scénář:</td></tr>
<tr><td>Krok</td><td>Aktér/Systém</td><td>Popis</td></tr>
<tr><td>1</td><td>Administrátor</td><td>Klikne na tlačítko „Send“.</td></tr>
<tr><td>2</td><td>Systém</td><td>Aplikace zobrazí uživateli vyskakovací okno s textboxem pro zadání zprávy emailu a s textboxy pro vyplnění dalších potřebných údajů.</td></tr>
<tr><td>3</td><td>Administrátor</td><td>Administrátor vyplní všechny potřebné údaje a kliknkne na tlačítko „Send“.</td></tr>
<tr><td>4</td><td>Systém</td><td>Aplikace odešle email všem uživatelům, kteří se nacházejí ve vyfiltrovaných statistikách.</td></tr>
<tr><td>5</td><td>Systém</td><td>Po dokončení odesílání systém zobrazí administrátorovi hlásku o úspěšném odeslání mailů.</td></tr>
<tr><td colspan="3"></td></tr>
</table>

7 DATABÁZE

Databáze jsou v tomto projektu dvě. První databází je databáze zaměstnanců, která byla vytvořena již dlouho před začátkem vývoje této aplikace, a jediné co bylo potřeba udělat bylo se k této databázi připojit. Databáze zaměstnanců se v aplikaci využívá za účelem získání informací o uživateli (číslo zaměstnance, jméno, email), aby mohli být později použité v aplikaci, například při rozesílání emailů.

Druhá databáze je databáze určená pro ukládání dat ze systému a pro tyto účely byla vytvořena za pomocí Entity Frameworku metodou Code-First, což je způsob tvorby databáze, při kterém vývojář vytvoří nejprve třídy v aplikaci na základě, kterých potom Entity Framework vygeneruje databázové tabulky. Následující odstavce jsou věnovány popisu této databáze a jejím entitám.



Obrázek 2 – Diagram databáze

7.1 CoffeeItemEntity

Je bezpochyby nejdůležitější entita databáze, jedná se o entitu, do níž se ukládají záznamy o jednotlivých vypitých kávách, každý záznam se má následující parametry: unikátní *Id*, *ProductName* (název produktu), *Portions* (na přípravu některých druhů kávy je spotřebováno více zrnkové kávy a z toho důvodu je u nich větší hodnota porce, při vyúčtování se potom platí za vypitý počet porcí nikoliv káv), *CreationDate* (Datum vytvoření), *LoggedUserId* (číslo zaměstnance který položku přidal) a *Device* (zařízení na kterém byla položka přidána).

7.2 CustomerEntity

Někdy může nastat situace, že je potřeba provést čištění kávovaru a poté udělat jednu kávu na propláchnutí kávovaru, nebo si přijde na kávu člověk z jiné firmy, aby bylo možné zaznamenat tyto vypité kávy do systému, byly vytvořeny karty pro servis a karty pro hosty, kterými je možno zaznamenávat vypité kávy do systému, jelikož ale tyto karty nepatří žádnému zaměstnanci, nedají se najít v databázi zaměstnanců, a to je důvod proč vznikla tabulka CustomerEntity. Jedná se o Entitu, v níž jsou uloženy čísla karet pro hosty a servis a každý záznam má následující parametry: unikátní *Id*, *Name* (název karty například Host1), *PersonalID* (číslo karty) a *Email*, na který bude odesláno vyúčtování.

7.3 BillEntity

Je entita určená pro ukládání informací o vyúčtování. Má následující parametry: unikátní *Id*, *CreationDate* (datum vytvoření), *BilledDate* (období pro které bylo provedeno vyúčtování), *Money* (náklady na nákup kávy v daném měsíci), *Competition* (udává druh soutěže zvolený pro dané vyúčtovací období) a *Device* (zařízení pro které je vyúčtování prováděno).

7.4 WinnersEntity

Je entita sloužící k ukládání informací o vítězích soutěže v daném vyúčtování. Má následující parametry: unikátní *Id*, *PersonalNuber* (číslo zaměstnance) a *BillEntityId* (cizí klíč, určuje, ke kterému vyúčtování se vztahuje daný vítěz).

7.5 AuthorizationEntity

Je entita navržená pro přidělování pravomocí ve webové aplikaci pro vyúčtování. Má následující parametry: unikátní *Id*, *Name* (jméno uživatele), *Role* (role uživatele, například admin), *Devices* (udává pro která zařízení může daný správce provádět vyúčtování).

8 TVORBA WPF APLIKACE

Nejdůležitější částí aplikace pro evidenci spotřeby kávy je i desktopová aplikace vytvořená ve frameworku WPF, jejíž hlavním využitím je ukládání a dat o vypitých kávách zaměstnanců, aplikace umí komunikovat se čtečkou RFID karet, komunikovat z databází, a má offline režim pro případy kdy je databáze nedostupná. V následujících odstavcích jsou popsány nejdůležitější funkce a třídy této webové aplikace.

8.1 Komunikace se čtečkou karet

Komunikaces RFID čtečkou karet je realizována ve třídě *COMCardReader*, kde nejdůležitější je metoda *SendPeriodicalQuery*, která periodicky kontroluje seriovou komunikaci se čtečkou karet. Pokud dostane nějaké bajty k přečtení, tak skočí do smyčky while, ve které čte bajty. Po přečtení všech bajtů upraví výsledný string do potřebného formátu pro další použití a vyvolá event *OnCardLoaded*, na který poté reaguje třída *AppModel*. Níže je vložený kód metody *SendPeriodicalQuery*.

```
protected override void SendPeriodicalQuery()
{
    try
    {
        if (serialCommunication.CommunicationPort.BytesToRead > 0)
        {
            StringBuilder sb = new StringBuilder();
            Thread.Sleep(200);
            while (serialCommunication.CommunicationPort.BytesToRead > 0)
            {
                var data = serialCommunication.CommunicationPort.ReadLine();
                sb.Append(data);
                var packet = sb.ToString();
                if (packet.Contains("Detected") && packet.Contains(":"))
                    break;
            }
            Debug.WriteLine($"{sb.ToString()}");
            var card = sb.ToString().Split(':');
            if (card.Length < 2) return;
            var code = card[1].RemoveLineEndings();
            Debug.WriteLine($"{code}");
            if (!string.IsNullOrEmpty(code))
                this.OnCardLoaded?.Invoke(this, code);

        }
    }
    catch (Exception ex)
    {
        Logger.Instance.Log.Error(ex);
    }
}
```

Kód 1 – Komunikace se čtečkou karet

8.2 Získávání informací o uživatelích

Informace o uživatelích se získávají ve třídě *RealPersonProvider* z databáze zaměstnanců, ze které tato třída získává data dvěma metodami. Metodou *GetPersonsInformations* anebo metodou *GetPersonsInformationsByID*.

8.2.1 Získávání informací o uživateli na základě kódu karty

Pro získávání informací o uživateli pomocí kódu karty se využívá metoda *GetPersonsInformations*, která vyhledá uživatele na základě kódu jeho karty. Tato metoda se používá ve chvíli, kdy se uživatel přihlašuje do aplikace pomocí své karty. Tato metoda pracuje tak, že číslo karty předá metodě *GetEmployeeByCardCode*, což je metoda třídy *AdministrationClient*, která pokud nalezne uživatele vrátí informace o uživateli, ze kterých je poté vytvořena instance třídy Person, pokud není uživatel nalezen vrátí metoda null. Metoda současně odchytává výjimku *EndpointNotFoundException*, která nastane, pokud databáze není z nějakého důvodu dostupná, v takovém případě metoda vrátí instanci třídy *OfflinePerson*, které předá číslo karty zaměstnance jako parametr.

```
  public IPerson GetPersonsInformations(string number)
  {
       try
       {
              using (var admin = new AdministrationClient())
              {
                     var user = admin.GetEmployeeByCardCode(number);
                     return (user is null || user.EmplId is null) ? null : new
Person(user.FirstName, user.LastName, user.EmplId);
              }
       }
       catch (System.ServiceModel.EndpointNotFoundException)
       {
              return new OfflinePerson("Offline", number);
       }
  }
```

Kód 2 – Získávání informací o uživateli na základě čísla karty

8.2.2 Získávání informací o uživateli na základě osobního čísla

Pro získávání informací o uživateli pomocí osobního čísla zaměstnance se využívá metoda *GetPersonsInformationsByID*. Tato metoda se používá například při aktualizaci statistik dat na úvodní obrazovce, a kdy je potřeba získat informace o uživateli na základě jeho čísla zaměstnance. Tato dělá to, že číslo zaměstnance předá metodě *GetEmployeeByEmplId*, což je metoda třídy *AdministrationClient*, která pokud nalezne uživatele, tak vrátí informace o uživateli, ze kterých je poté vytvořena instance třídy Person.

```
public IPerson GetPersonsInformationsByID(string number)
{
	using (var admin = new AdministrationClient())
	{
		var user = admin.GetEmployeeByEmplId(number);
		return (user is null || user.EmplId is null) ? null : new
	Person(user.FirstName, user.LastName, user.EmplId);
	}
}
```

Kód 3 - Získávání informací o uživateli na základě čísla zaměstnance

8.3 Offline režim

V následujících odstavcích je popsána funkcionalita offline režimu aplikace, který nastane při ztrátě spojení se serverem, a ve kterém aplikace serializuje a ukládá data do lokální paměti a po opětovném navázání spojení se serverem data deserializuje a uloží do databáze.

8.3.1 Serializace a ukládání dat

K serializaci dat dochází v této aplikaci v případě, kdy z nějakého důvodu není dostupná databáze a aplikace je nucena běžet v offline režimu. V takovém případě je nutné ukládat data o vypitých kávách do souboru, za kterého bude možné, po připojení aplikace k databázi, nahrát data do databáze. O serializaci dat se stará třída *OfflineJsonSerialization*, která serializuje a ukládá data do souboru pomocí metody *Write*. Což je metoda, která nejprve načte aktuální položky ze souboru do listu, poté do tohoto listu přidá požadovaná data a následně list opět serializuje do formátu json a zapíše do souboru.

```
        public void Write(SerializableItem serializableItem)
        {
            var serializableItems = Read();
            serializableItems.Add(serializableItem);
            File.WriteAllText(Filename,
JsonConvert.SerializeObject(serializableItems, Formatting.Indented));
        }
```

Kód 4 – Zapisování dat do souboru

8.3.2 Deserializace a načítání dat

K deserializaci dat dochází při načítání dat z offline souboru. O načítáni dat ze souboru se stará třída *OfflineJsonSerialization*, která načítá data pomocí metody Read, která nejprve ověří, zda soubor, ze kterého chce uživatel číst existuje, pokud ne tak vrátí prázdný list, a pokud soubor existuje tak načte a deserializuje jeho data.

```
        public List<SerializableItem> Read()
        {
            if (!File.Exists(Filename))
                return new List<SerializableItem>();
            string json = File.ReadAllText(Filename);
            return JsonConvert.DeserializeObject<List<SerializableItem>>(json);
        }
```

Kód 5 – Čtení dat ze souboru

8.3.3 Kontrola aktuálnosti dat a ukládání offline dat do databáze

O kontrolu toho, zda jsou data aktuální se stará třída ActualityControl, která v pravidelném časovém intervalu pomocí property OfflineDispatcherTimer vyvolá událost a ta spustí metodu OfflineDispatcherTimer_Tick. OfflineDispatcherTimer_Tick je metoda, která nejprve otestuje, zda je databáze dostupná a následně nahraje pomocí metody Read, ze třídy OfflineJsonSerialization.cs, kolekci dat. Pokud tato kolekce obsahuje alespoň jednu položku, tak jsou postupně všechny položky nahrané do databáze. Po nahrání všech dat do databáze dojde k odstranění offline souboru.

```
        private void OfflineDispatcherTimer_Tick(object sender, EventArgs e)
        {
            try
            {
                if (appData.TestConnection())
                {
                    var reader = new OfflineJsonSerialization();
                    var collection = reader.Read();
                    if (collection.Count() > 0)
```

```
                    {
                        foreach (var serializableItem in collection)
                        {
                            var coffeeRepository = appData.CoffeeItemsRepository;
                            var coffeeItem =
serializableItem.GetCoffeeItemDTO(personProvider);
                            if(coffeeItem.LoggedUserId!=null)
                                coffeeRepository.Insert(coffeeItem);
                        }
                        var coffeeRace = new CoffeeRace();
                       coffeeRace.SortCoffeeRaceStandings(this.settings.DeviceName);
                    }
                    reader.DeleteFile();
                }
            }
            catch (Exception ex)
            {
                Logger.Instance.Log.Error(ex);
            }
        }
```

Kód 6 – Kontrola aktuálnosti dat

8.4 Vzhled aplikace a zobrazování dat

O vzhled uživatelského rozhraní aplikace se starají tři UserControly, *DeafaultUserControl.xaml*, *CoffeeUserControl.xaml* a *LoadingUserControl.xaml*. Každý UserControl má také vlastní ViewModel. Jelikož aplikace je určena k tomu, aby běžela na dotykovém displeji, bylo při návrhu jejího uživatelského rozhraní usilováno o to, aby se aplikace z dotykového displeje dobře ovládala, aby byla tlačítka dost velká, a aby bylo uživatelské rozhraní přehledné a uživatelsky přívětivé.

8.4.1 Výchozí obrazovka

Je obrazovka je zobrazená ve chvíli, kdy uživatel není přihlášený ani se zrovna nepřihlašuje nebo neukládá položku do databáze. V levé části této obrazovky je zobrazeno datum a čas a v pravé části je zobrazena tabulka deseti uživatelů s nejvyšším počtem vypitých káv v aktuálním měsíci. Ve spodní části je zobrazena informace o celkovém počtu vypitých káv v daném měsíci. Uživatelské rozhraní této obrazovky je realizováno v UserControlu *DefaultUserControl*. O aktuálnost dat v uživatelském rozhraní se stará třída *DefaultViewModel*, která zodpovídá za aktuálnost uživatelského rozhraní. O aktualizaci dat se stará metoda *UpdateMonthStats*, která je vyvolána při změně uživatele, když se uživatel přihlásí nebo odhlásí.

8.4.2 Metoda UpdateMonthStats

Metoda starající se o aktualizaci dat v uživatelském rozhraní defaultní obrazovky. Metoda je volána událostí vyvolanou změnou aktuálního uživatele. Metoda nejprve otestuje, zda je aplikace připojena k databázi, pokud není tak aktualizace neproběhne. V případě že kontrola spojení z databází proběhne v pořádku tak se zavolá metoda *SortCoffeeRaceStandings*, což je metoda třídy *CoffeeRace*, která vrátí seřazený list, se zaměstnaneckými čísly a počty vypitých káv jednotlivých uživatelů. Poté se pro prvních deset uživatelů získají z databáze jejich jména, která jsou poté i s počty vypitých káv zobrazena v tabulce defaultní obrazovky. Metoda také zavolá metodu *DrinkedCoffees*, což je metoda třídy *CoffeeRace*, která vrátí celkový počet káv vypitých v daném měsíci na aktuálním zařízení.

```
public void UpdateMonthStats()
        {
            if (appModel.LoggedUser is OfflinePerson ||
appModel.appData.TestConnection() == false)
                return;
            var coffeeRace = new CoffeeRace();
            var result =
coffeeRace.SortCoffeeRaceStandings(this.settings.DeviceName).ToList();
            this.DrinkedCoffees =
coffeeRace.DrinkedCoffees(this.settings.DeviceName);
            this.Result.Clear();

            int position = 0;
            var counter = 10;
            for (int i = 0; i < counter; i++)
            {
                if (i >= result.Count()) return;

                var user =
this.personProvider.GetPersonsInformationsByID(result[i].ID);
                if (user is null)
                {
                    counter++;
                    continue;
                }
                position++;
                this.Result.Add(new RaceResult(position, user.Name, user.Surname,
result[i].CoffeeCount));
            }
        }
```

Kód 7 – Aktualizace dat defaultní obrazovky

Obrázek 3 – Výchozí obrazovka aplikace

8.4.3 Hlavní obrazovka

Je obrazovka zobrazující se po přihlášení uživatele do systému. Tato obrazovka je horizontálně rozdělena na dvě části. V dolní části se nachází jednotlivá tlačítka pro volbu kávy, přičemž pořadí tlačítek je ovlivněno počtem káv, které daný uživatel vypil, typ kávy, který si dává nejčastěji se nachází na největším tlačítku v levé části obrazovky, ostatní druhy jsou seřazeny po řádcích z leva do prava. V horní části se nachází informace o uživateli, v levé části je jméno uživatele, a v pravé části statistické údaje uživatele, pořadí v soutěži, počet vypitých káv v aktuálním měsíci a počet vypitích káv v aktuální den. O aktuálnost dat v uživatelském rozhraní se stará třída *MainViewModel*, která zodpovídá za aktuálnost uživatelského rozhraní. O aktualizaci dat se stará metoda *AppModel_OnPersonChanged*, která je vyvolána při změně uživatele, když se uživatel přihlásí. Třída MainViewModel také obsahuje Commandy pro odhlášení a pro zvolení kávy na které binduje jednotlivá tlačítka.

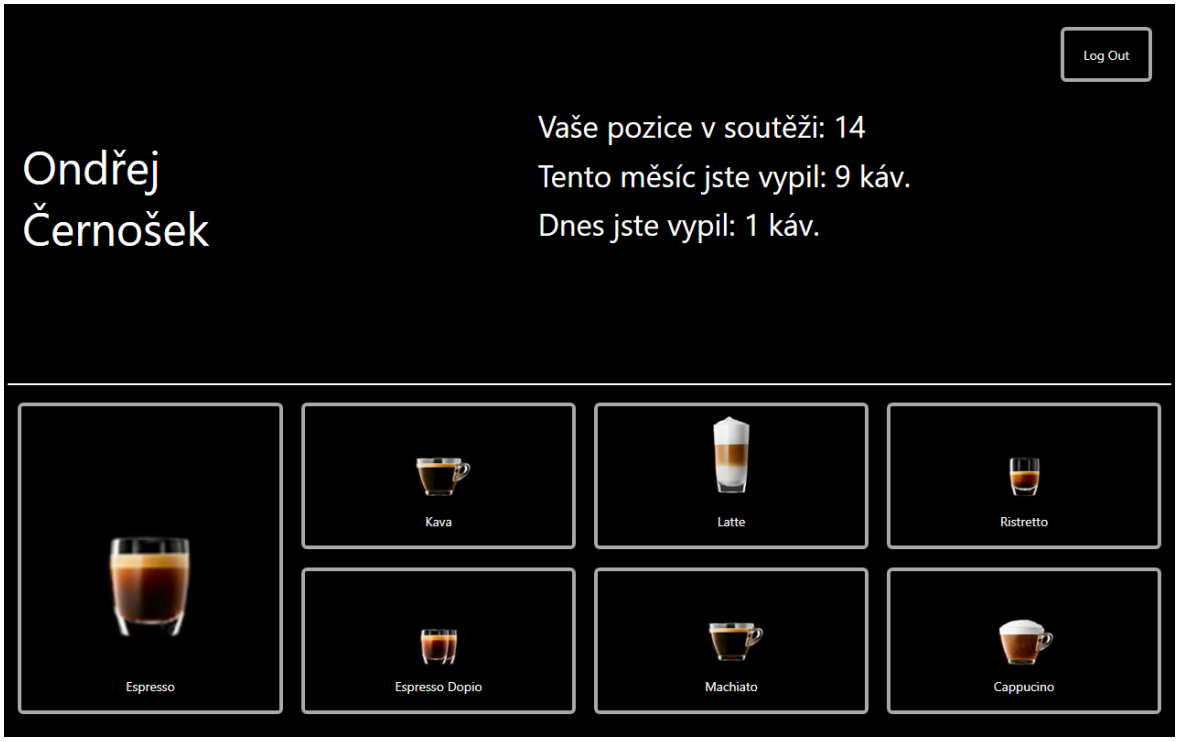
8.4.4 Metoda AppModel_OnPersonChanged

Metoda starající se o aktualizaci dat v uživatelském rozhraní hlavní obrazovky. Metoda je volána událostí vyvolanou změnou aktuálního uživatele. Metoda nejprve otestuje, zda je

aplikace připojena k databázi. Pokud není aplikace připojena k databázi, zobrazí se pouze tlačítka s výběrem káv a na místě, určeném pro jméno uživatele se zobrazí informace o tom že aplikace je v offline režimu. Pokud kontrola připojení proběhne v pořádku potom si metoda získá data o vypitých kávách pomocí metod třídy *AppModel* a data o pořadí v soutěži pomocí metody třídy *CoffeeRace*. Uspořádání tlačítek s kávami podle preferencí uživatele probíhá díky metodě *SortCoffeeList* třídy *CoffeeList*.

```
private void AppModel_OnPersonChanged(object sender, IPerson e)
        {
            SelectedPerson = e;
            if (SelectedPerson != null && !(SelectedPerson is OfflinePerson) &&
appModel.appData.TestConnection()==true)
            {
                var tmp = appModel.FindByUserID(SelectedPerson.PersonalID);
                CoffeeCount = tmp.Count();
                tmp = appModel.FindByUserIDDaily(SelectedPerson.PersonalID);
                DailyCoffeeCount = tmp.Count();
                var coffeeList = new CoffeeList();
                var coffeeRace = new CoffeeRace();
                this.Coffees = coffeeList.SortCoffeeList(tmp);
                coffeeRace.SortCoffeeRaceStandings(this.setings.DeviceName);
                this.CurrentPosition =
coffeeRace.FindMyRacePosition(SelectedPerson.PersonalID);
            } else {
                var coffeeList = new CoffeeList();
                this.Coffees = coffeeList.FillCoffeeList();
            }
        }
```

Kód 8 – Aktualizace dat hlavní obrazovky



Obrázek 4 – Hlavní obrazovka aplikace

8.4.5 Načítací obrazovka

Je obrazovka zobrazující se během přihlašování a odhlašování uživatele nebo při ukládání dat do databáze. Vzhled načítací obrazovky je realizován pouze textem uprostřed, který říká, jaká operace aktuálně probíhá. Uživatelské rozhraní této obrazovky je realizováno v user controlu *LoadingUserControl* a o obsah textu se stará třída *LoadingViewModel*, která reaguje na různé události vyvolané ve třídě AppModel a v závislosti na tom o jakou událost se jedná, zobrazí vhodný text.

```
private void AppModel_OnCardReaded(object sender, string e)
{
    if (e == null) this.IsDialogShown = false;
    else this.IsDialogShown = true;
    LoadingText = "Probíhá přihlašování...";
}
```

Kód 9 – Zobrazení načítací obrazovky

8.4.6 Získávání statistických údajů

Statistické údaje o vypitých kávách jednotlivých uživatelů i o vypitých kávách celkově se získávají za pomocí třídy *CoffeeRace*, která má tři metody, metodu SortCoffeeRaceStandings, jejímž získat a seřadit údaje o vypitých kávách v aktuálním měsíci, metodu *FindMyRacePosition*, která vrací aktuální pozici daného uživatele, v pořadí měsíční soutěže a metodu *DrinkedCoffees*, která vrací celkový počet káv vypitých v aktuálním měsíci.

8.5 Model aplikace

Tato aplikace je tvořena podle návrhového vzoru Model-View-ViewModel a v tomto případě zastává třída AppModel funkci modelu. Obsahuje metody *Confirm*, *OnLoadedCard* a další metody pro práci s databází.

8.5.1 Metoda Confirm

Je metoda sloužící pro uložení uživatelem zvolené kávy do databáze, a následné aktualizování pořadí ve statistice.

```
        public async Task Confirm(Coffee coffee)
        {
            if (this.LoggedUser == null)
                return;

            var coffeeItem = new CoffeeItemDTO();
            coffeeItem.ProductName = coffee.Name;
            coffeeItem.Portions = coffee.Portions;
            coffeeItem.LoggedUserId = this.LoggedUser.PersonalID;
            coffeeItem.CreationDate = DateTime.Now;
            coffeeItem.Device = this.settings.DeviceName;
            bool completed=false;
            try
            {
                this.OnSavingToDB?.Invoke(this, EventArgs.Empty);
                await Task.Delay(1000);

                if (LoggedUser is OfflinePerson || appData.TestConnection() ==
false)
                {
                    this.offlineSerializer.Write(new SerializableItem(coffeeItem,
LoggedUser));
                    completed = true;
                }
                else
```

```
                {
                    var coffeeRepository = appData.CoffeeItemsRepository;
                    var sw = Stopwatch.StartNew();
                    coffeeRepository.Insert(coffeeItem);
                    sw.Stop();
                    this.SavingTime = sw.ElapsedMilliseconds;
                    completed = true;
                    CoffeeRace.SortCoffeeRaceStandings(this.settings.DeviceName);
                }
                Logger.Instance.Log.Info($"logging time: {LoggingTime}ms, saving
time: {SavingTime}ms. Conected:
{System.Net.NetworkInformation.NetworkInterface.GetIsNetworkAvailable()}");
                this.LoggedUser = null;
                this.dispatcherTimer.Stop();
            }
            catch (Exception ex)
            {
                Logger.Instance.Log.Error(ex);
            }
            finally
            {
                if (!completed)
                    this.offlineSerializer.Write(new SerializableItem(coffeeItem,
LoggedUser));
                }
        }
```

Kód 10 – Ukládání do databáze

8.5.2 Metoda OnLoadedCard

Je metoda, kterou vyvolá událost *OnCardLoaded*, informující o načtení RFID karty, třídy *COMCardReader*. Úkolem této metody je za pomocí jiných tříd a metod získat údaje o uživateli na základě přiložené karty zaměstnance a přihlásit jej do systému.

```
private async void OnLoadedCard(object sender, string e)
        {
            if (LoggedUser != null) return;

            try
            {
                this.OnCardReaded?.Invoke(this, e);
                await Task.Delay(1000);

                var sw = Stopwatch.StartNew();
                var person = this.personProvider.GetPersonsInformations(e);
                sw.Stop();
                this.LoggingTime = sw.ElapsedMilliseconds;
                Logger.Instance.Log.Info($"Only Logging Time: {LoggingTime}");
                if (person == null)
                {
                    person = this.FindInDb(e);
                }
                if (person == null)
```

```
                    throw new MeoException("Eror 404 Card not found, Cheating is
forbidden");
                this.OnUserLogged?.Invoke(this, person);
                await
Application.Current.Dispatcher.BeginInvoke(DispatcherPriority.Background,
                new Action(() => this.LoggedUser = person));
                dispatcherTimer.Start();
            }
            catch (Exception ex)
            {
                this.OnCardNotFound.Invoke(this, EventArgs.Empty);
                Logger.Instance.Log.Error(ex);
                await Task.Delay(2000);
                await
Application.Current.Dispatcher.BeginInvoke(DispatcherPriority.Background,
                new Action(() => this.LoggedUser = null));
            }
        }
```

Kód 11 – Získávání dat o uživateli

9 TVORBA WEBOVÉ APLIKACE

Součástí aplikace pro evidenci spotřeby kávy je i webová aplikace vytvořená v Blazoru, jejíž hlavním využitím je usnadnění práce při pravidelném vyúčtování kávy, aplikace umí generovat reporty vypočítat cenu jedné porce kávy, rozesílat maily s vyúčtováním a další. V následujících odstavcích jsou popsány nejdůležitější funkce a třídy této webové aplikace.

9.1 Generování reportů

O generování reportů v excelovském formátu se stará třída ExcelExporter.cs a její metody.

Metoda Export je metoda sloužící pro vyexportování dat do excelovského souboru. Metoda nejprve převede list dat určených pro export do formátu třídy DataTable pomocí metody ToDataTable. Následně za pomocí instance třídy *XLWorkbook* která je součástí NuGet knihovny *ClosedXML* [31] vytvoří excelovský soubor.

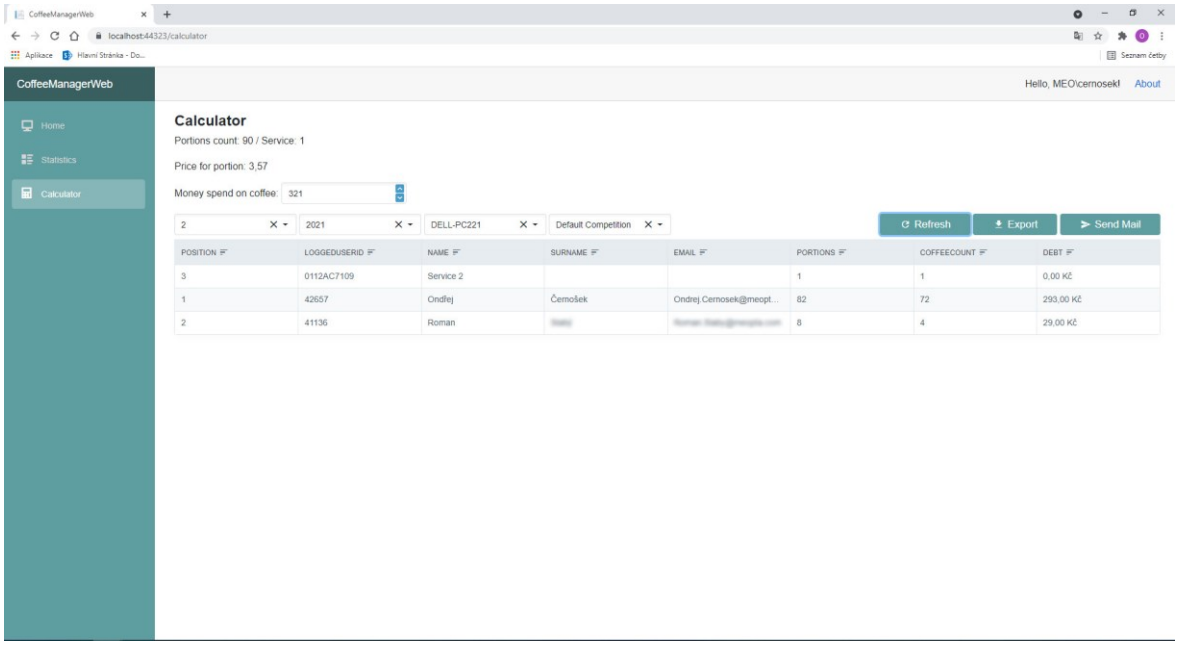
```
public void Export(List<CompleteCoffeeItem> completeList, IJSRuntime jSRuntime)
        {
            string woksheet = "Exported";
            DataTable dt = ToDataTable(completeList);
            XLWorkbook wb = new XLWorkbook();
            string filename = woksheet;
            wb.Worksheets.Add(dt, woksheet);
            if (filename.Contains("."))
            {
                int IndexOfLastFullStop = filename.LastIndexOf('.');
                filename = filename.Substring(0, IndexOfLastFullStop) + ".xlsx";
            }
            filename = filename + ".xlsx";
            var workbookBytes = new byte[0];
            using (var ms = new MemoryStream())
            {
                wb.SaveAs(ms);
                workbookBytes = ms.ToArray();
            }
            jSRuntime.InvokeAsync<object>("saveAsFile", "Exported.xlsx",
Convert.ToBase64String(workbookBytes));

        }
```

Kód 12 – Exportování dat

9.2 Vzhled aplikace a zobrazování dat

O vzhled webové aplikace se aplikace se stará několik razor souborů, což jsou soubory, které v sobě kombinují HTML a C# kód. Nejdůležitějším razor souborem v této aplikaci je *Calculator.razor* který se stará o vzhled stránky, na které probíhá generování reportů, výpočet ceny kávy nebo rozesílání emailů s vyúčtováním. V razor souborech se používají open-sourceové komponenty *RadzenComponents* [32], které usnadňují práci s designem jednotlivých komponent.



Obrázek 5 – Vzhled stránky pro vyúčtování

9.3 Rozesílání emailů

O rozesílání emailů s vyúčtováním uživatelům se stará třída *MailSender* a její metoda *Send*, která pomocí třídy *SmtpClient* vytvoří pro každého uživatele email, ve kterém jsou uvedeny údaje o počtu vypitých káv uživatele za daný měsíc, cena za porci kávy a výše dluhu uživatele.

9.4 Soutěž a vyúčtování

Při tvorbě vyúčtování je možnost zvolit soutěž pro aktuální měsíc, kdy tři uživatelé s nejvyšším počtem zaznamenaných káv za daný měsíc získají slevu na vypitou kávu. O vyúčtování se stará buď třída *DefultCompetition* (v případě že se soutěž nevyhlašuje) anebo třída *ClassicCompetition* (v případě soutěže kdy první tři mají nárok na slevu).

9.4.1 Třída DefaultCompetition

Je třída, která jejíž metoda *GetBill* se používá pro tvorbu vyúčtování. Tato metoda na základě uživatelem zvolených dat

```
        public List<CompleteCoffeeItem> GetBill( int selectedMonth, int
selectedYear, string Device, int money)
        {
            var actual = new DateTime(selectedYear, selectedMonth, 1);
            var lastDate = actual.AddMonths(-1);

            var actualCoffeeItems =
this.sortService.SortCoffeeRaceStandings(actual.Month, actual.Year, Device);
            var oldCoffeeItems = sortService.SortCoffeeRaceStandings(lastDate.Month,
lastDate.Year, Device);


            var completeItem = new CompleteCoffeeItem();
            var completeCoffeeItems = completeItem.GetCoffeeItem(actualCoffeeItems,
sortService);

            var portions = CalculatePortions(completeCoffeeItems);

            Portions = portions.X;
            ServicePortions = portions.Y;
            PriceForPortion = CalculatePriceForPortion(Portions, money);

            return CalculateDebt(completeCoffeeItems, PriceForPortion);
        }
```

Kód 13 – Vyúčtování

9.4.2 Třída ClasicCompetition

Jedná se o třídu, která využívá funkcionality třídy DefultCompetition, jen s tím rozdílem, že nejprve vyhodnotí výsledek soutěže a odečte vítězům počet porcí k zaplacení podle pořadí, před tím ale ještě vyloučí z aktuální soutěže vítěze z předchozího měsíce.

```
public List<CompleteCoffeeItem> GetBill(int selectedMonth, int selectedYear, string
Device, int money)
        {
            var actualCoffeeItems = RemoveLastWinners(selectedMonth, selectedYear,
Device);

            actualCoffeeItems = LowerPortionsForWinners(actualCoffeeItems);

            var completeItem = new CompleteCoffeeItem();

            var completeCoffeeItems = completeItem.GetCoffeeItem(actualCoffeeItems,
sortService);

            var portionsPoint =
this.defaultCompetition.CalculatePortions(completeCoffeeItems);

            Portions = portionsPoint.X;
            ServicePortions = portionsPoint.Y;
            PriceForPortion =
this.defaultCompetition.CalculatePriceForPortion(Portions, money);

            return this.defaultCompetition.CalculateDebt(completeCoffeeItems,
PriceForPortion);
        }
```

Kód 14 – Vyúčtování se soutěží

10 BEZPEČNOST

Bezpečnost aplikace pro evidenci spotřeby kávy je zajištěna především tím, že aplikace je připojena na interní síti společnosti, která je zabezpečená. Ve webové aplikaci se také pro zvýšení bezpečnosti využívá autorizace uživatelů, která se provádí za pomocí knihovny *Microsoft.AspNetCore.Authorization* od Microsoftu, která provádí autorizaci uživatele na základě informací o aktuálně přihlášeném uživateli ve Windows.

Zabezpečení databáze je realizováno opět připojením pouze k interní síti, a connection stringy opatřenými hesly. Databáze neukládá žádné citlivé informace, ale pouze číslo zaměstnance a vypité kávy a z toho důvodu z ní není možné získat citlivé údaje.

Za další bezpečnostní prvek lze také považovat to, že uživatel nezadává žádné vstupy do aplikace.

11 DEMONSTRACE VÝSLEDKŮ

V následující kapitole jsou demonstrovány výsledky vývoje aplikace pro evidenci spotřeby kávy.

11.1 Funkcionalita WPF aplikace

V této kapitole je popsána funkcionalita WPF aplikace z pohledu běžného uživatele.

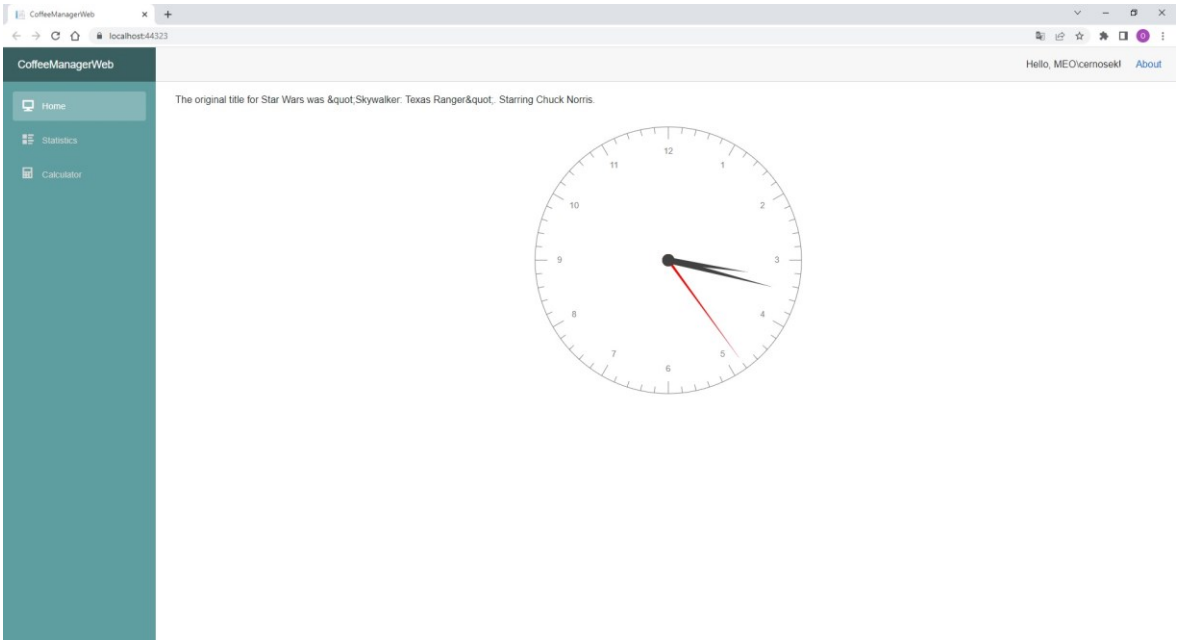
Na začátku se uživatel nachází na výchozí obrazovce viz. Obrázek 3 v kapitole 8, kde může vidět aktuální datum a čas, celkový počet zaznamenaných káv v aktuálním měsíci nebo tabulku s deseti uživateli s nevyšším počtem zaznamenaných káv v aktuálním měsíci. V této fázi může uživatel pouze přiložit svou kartu zaměstnance ke čtečce RFID karet a tím se přihlásit do aplikace a zobrazit hlavní obrazovku viz. Obrázek 4 v kapitole 8.

Poté co se uživatel úspěšně přihlásí se mu zobrazí hlavní obrazovka aplikace, na které může vidět své jméno, svou aktuální pozici v soutěži, počet vypitých káv v aktuálním měsíci a počet vypitých káv v aktuálním dni. V této fázi se může uživatel buď odhlásit za pomocí tlačítka Log Out, nebo si zaznamenat kávu stisknutím tlačítka, které odpovídá jeho zvolené kávě. Tlačítka se seřazují podle preferencí každého uživatele, druh kávy, který si uživatel dává nejčastěji se nachází vždy na největším tlačítku.

11.2 Funkcionalita webové aplikace

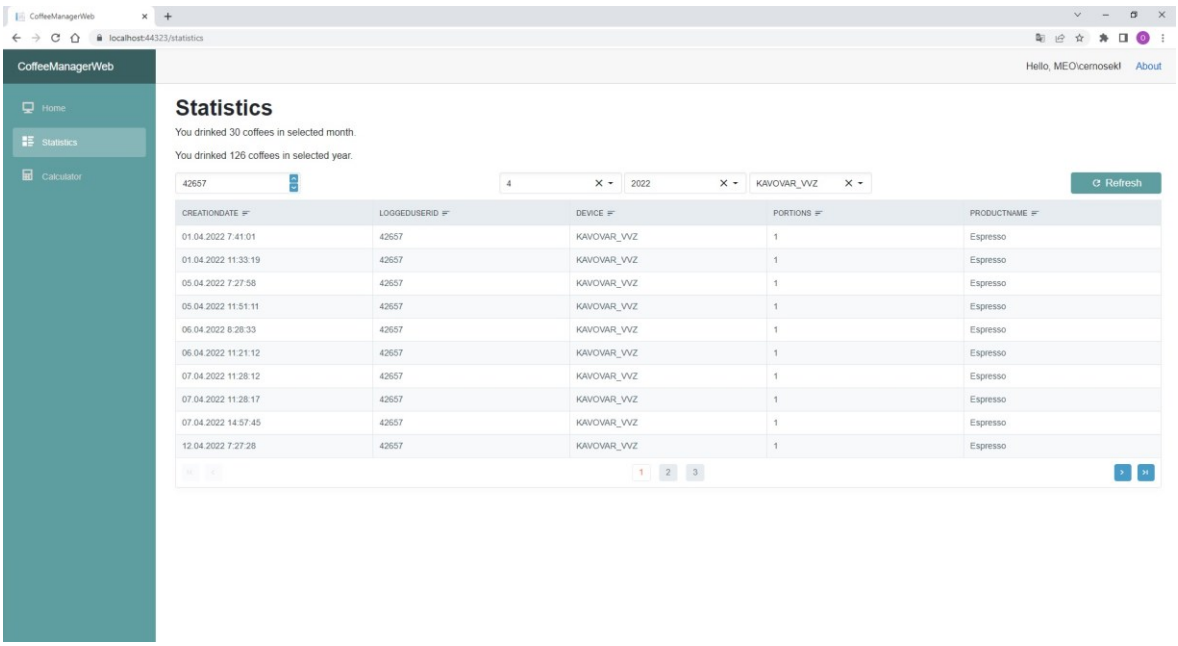
V této kapitole je popsána funkcionalita webové aplikace nejprve z pohledu uživatele a poté z pohledu administrátora.

Na začátku se uživatel nachází na úvodní stránce webové aplikace, na které může vidět tikající ručičkové hodiny ukazující aktuální čas a v levé části může vidět navigační panel s dostupnými stránkami. Uživatel, který není administrátor má zobrazeny pouze možnosti Home (úvodní stránka) a Statistics (stránka se statistikou).



Obrázek 6 – Úvodní stránka webové aplikace pro vyúčtování

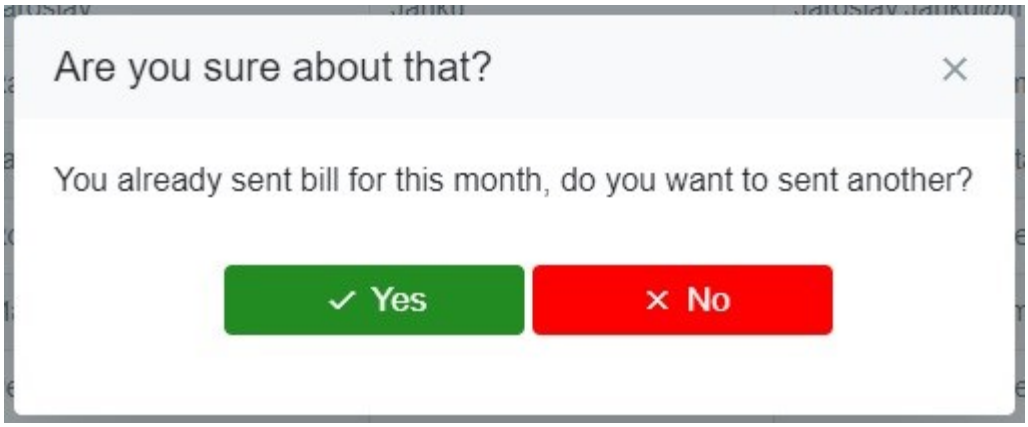
Na stránce se statistikou si může uživatel zobrazit informace o svých zaznamenaných kávách, data se zobrazují podle uživatelem zvolených kritérií, kterými jsou měsíc, rok a zařízení na kterém uživatel zaznamenal své kávy.



Obrázek 7 – Vzhled stránky se statistikou webové aplikace pro vyúčtování

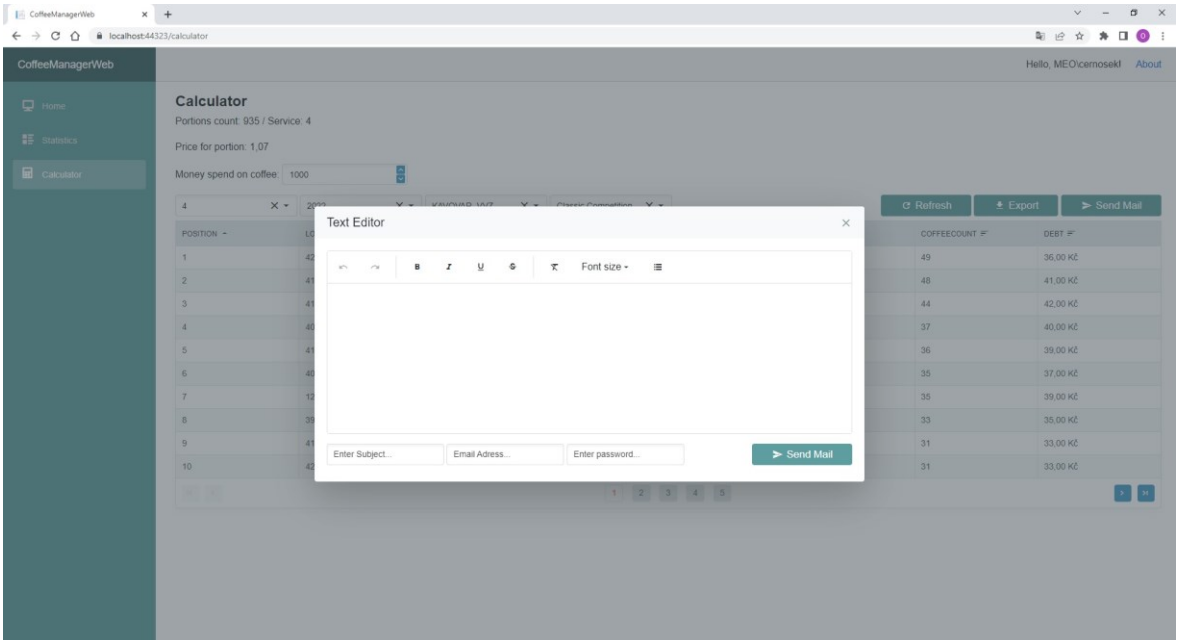
Z pohledu administrátora je aplikace o něco zajímavější, na stránce úvodní stránce i na stránce se statistikou má administrátor stejné možnosti jako běžný uživatel, v čem se ale liší je navigační menu, ve kterém má administrátor navíc možnost zvolit tlačítko s nápisem Calculator (stránka pro vyúčtování viz. Obrázek 5 v kapitole 9)

Na stránce pro vyúčtování může administrátor zobrazovat data o zaznamenaných kávách uživatelů na základě zvoleného měsíce, roku a zařízení. Další možností je vypočítat cenu jedné porce kávy a výší dluhů jednotlivých uživatelů anebo například vyhlásit měsíční soutěž, zvolením položky v ComboBoxu. Administrátor také může kliknutím na tlačítko Export vyexportovat soubor ve formátu .xls ve kterém se vytvoří tabulka s daty aktuálně zobrazenými daty na stránce pro vyúčtování. Poslední věcí, kterou administrátor ještě může udělat na stránce pro vyúčtování je kliknout na tlačítko Send Mail, které ověří, zda již pro tyto parametry nebylo provedeno vyúčtování, a pokud bylo tak zobrazí uživateli vyskakovací okno s upozorněním a dotazem, zda chce maily odeslat znovu.



Obrázek 8 – Dialogové okno s upozorněním

Pokud administrátor zvolí, že chce opakovaně poslat emaily nebo pokud pro zvolené parametry odeslání ještě neproběhlo, pak se administrátorovi zobrazí dialogové pro odesílání emailů, ve kterém může napsat zprávu emailu, předmět. Pokud chce administrátor odeslat emaily uživatelům je nucen také vyplnit svůj email, ze kterého se emaily následně odešlou, a heslo k tomuto emailu.



Obrázek 9 – Dialogové okno pro tvorbu emailů

ZÁVĚR

Cílem bakalářské práce bylo vytvoření aplikace pro evidenci spotřeby kávy, pro usnadnění evidování káv a následné vyúčtování. Aplikace byla vytvořena a nasazena do testovacího provozu a ze strany uživatelů získala kladné ohlasy.

Při vývoji aplikace bylo dbáno na jednoduchost a uživatelskou přívětivost aplikace. Aplikace umožňuje uživateli se přihlásit za pomocí RFID karty zaměstnance, a následně vybrat druh zvolené kávy, který se spolu s číslem uloží do databáze. Aplikace také nabízí offline režim, pro případ ztráty spojení, při kterém se do logovacího souboru logují čísla karet zaměstnanců a jimi zvolené kávy, aby se mohli po opětovném připojení mohli nahrát do databáze. Webová aplikace nabízí možnost zobrazení statistik za dané období, vypočítání ceny porce kávy, a dokonce i vyhlášení kávové soutěže, při které tři nejaktivnější spotřebitelé kávy, kteří současně nejsou vítězové z minulého vyúčtování, získají slevu při vyúčtování. Webová aplikace umožňuje také vygenerování souboru ve formátu .xls obsahujícího tabulku se jmény uživatelů a jejich útratami, a také umožňuje hromadné rozesílání emailů všem uživatelům, kdy každý email obsahuje žádost o zaplacení útraty uživatele a výši útraty daného uživatele.

Aplikace by bylo možné rozšířit zakoupením rozšiřujícího modulu kávovaru, díky kterému by mohla aplikace pro evidenci spotřeby komunikovat přímo s kávovarem, takže by aplikace dokázala uživateli dokonce i uvařit kávu.

SEZNAM POUŽITÉ LITERATURY

[1] *Perficient.com* [online]. Perficient, 2021 [cit. 2022-03-24]. Dostupné z: https://www.perficient.com/insights/research-hub/mobile-vs-desktop-usage

[2] Microsoft technical documentation. *Microsoft technical documentation* [online]. Microsoft, 2022 [cit. 2022-03-28]. Dostupné z: https://docs.microsoft.com/en-us/

[3] Best frameworks for desktop application development. Solace Infotech Pvt. Ltd. [online]. Solace Infotech Pvt., 2021 [cit. 2022-03-02]. Dostupné z: https://solaceinfotech.com/blog/best-frameworks-for-desktop-application-development/

[4] Qt [online]. The Qt Company, 2022 [cit. 2022-05-10]. Dostupné z: https://www.qt.io

[5] Documentation archive. *Documentation Archive* [online]. Apple, 2013 [cit. 2022-03-30]. Dostupné z: https://developer.apple.com/library/archive/documentation/Cocoa/

[6] SwiftUI. *Apple developer* [online]. Apple, 2022 [cit. 2022-05-10]. Dostupné z: https://developer.apple.com/xcode/swiftui/

[7] Electron Documentation. *Electron* [online]. OpenJS Foundation, 2022 [cit. 2022-03-30]. Dostupné z: https://www.electronjs.org/docs/latest

[8] Advancement of Swing. *Section* [online]. Section, 2021 [cit. 2022-04-02]. Dostupné z: https://www.section.io/engineering-education/advancement-of-swing-frameworks-in-java/

[9] YOSIFOVICH, Pavel. Windows Presentation Foundation 4.5 Cookbook. Packt Publishing, 2012. ISBN 9781849686228.

[10] 2021 Developer survey. Stack owerflow [online]. Stack Exchange, 2021 [cit. 2022-04-27]. Dostupné z: https://insights.stackoverflow.com/survey/2021

[11] React Documentation. *React* [online]. Meta Platforms, 2022 [cit. 2022-04-03]. Dostupné z: https://reactjs.org/

[12] Angular Documentation. *Angular* [online]. Google, 2022 [cit. 2022-04-03]. Dostupné z: https://angular.io/

[13] Vue.js Documentation. *Vue.js* [online]. Evan You, 2022 [cit. 2022-04-08]. Dostupné z: https://v2.vuejs.org/

[14] Entity Framework Documentation. *Entity Framework* [online]. ZZZ Projects, 2022 [cit. 2022-04-08]. Dostupné z: https://entityframework.net/

[15] Entity Framework. *EntityFrameworkTutorial* [online]. EntityFrameworkTutorial.net, 2022 [cit. 2022-04-12]. Dostupné z: https://www.entityframeworktutorial.net/

[16] BUONANNO, Enrico. Functional Programming in C#: How to write better C# code. Manning, 2017. ISBN 978-1617293955.

[17] WATSON, Ben. Writing High-Performance .NET Code. 2nd. 2018. ISBN 978-0990583431.

[18] XU, Jack. Practical WPF Charts and Graphics. 2010th edition. Apress, 2011. ISBN 9781430223160.

[19] HTML Introduction. *W3schools* [online]. W3Schools, 2022 [cit. 2022-04-12]. Dostupné z: https://www.w3schools.com/html/html_intro.asp

[20] CSS Introduction. *W3schools* [online]. W3Schools, 2022 [cit. 2022-04-12]. Dostupné z: https://www.w3schools.com/css/css_intro.asp

[21] JavaScript Overview. *Tutorialspoint* [online]. tutorialspoint, 2022 [cit. 2022-04-12]. Dostupné z: https://www.tutorialspoint.com/javascript/javascript_overview.htm

[22] TypeScript Documentation. *TypeScript* [online]. Microsoft, 2022 [cit. 2022-04-15]. Dostupné z: https://www.typescriptlang.org/

[23] Swift Documentation. *Swift* [online]. Apple, 2022 [cit. 2022-04-15]. Dostupné z: https://www.swift.org/

[24] SQL. *Tutorialspoint* [online]. tutorialspoint, 2022 [cit. 2022-04-15]. Dostupné z: https://www.tutorialspoint.com/sql/index.htm

[25] GAMMA, Erich. Design patterns: elements of reusable object-oriented software. Boston: Addison-Wesley, 1995. ISBN 978-0201633610.

[26] FOWLER, Martin. Patterns of enterprise application architecture. Boston: Addison-Wesley, c2003. The Addison-Wesley Signature Series. ISBN 9780321127426.

[27] SKIENA, Steven. The Algorithm Design Manual. 2nd. Springer, 2008. ISBN 978-1849967204.

[28] OWASP Top Ten. OWASP [online]. OWASP Foundation, 2021 [cit. 2022-04-02]. Dostupné z: https://owasp.org/www-project-top-ten/

[29] OWASP Cheat Sheet Series. OWASP Cheat Sheet Series [online]. Attribution 3.0 Unported, 2021 [cit. 2022-04-15]. Dostupné z: https://cheatsheetseries.owasp.org/

[30] What is server-side request forgery. Acunetix [online]. Acunetix 2022, by Invicti, 2022 [cit. 2022-04-15]. Dostupné z: https://www.acunetix.com/blog/articles/server-side-request-forgery-vulnerability/

[31] ClosedXML. Github [online]. 2022 [cit. 2022-04-20]. Dostupné z: https://github.com/ClosedXML/ClosedXML

[32] Radzen Components. Radzen [online]. Radzen, 2022 [cit. 2022-05-10]. Dostupné z: https://blazor.radzen.com/

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

CSS	Cascading Style Sheet
DDoS	Distributed denial of Service
DoS	Denial of Service
EDMX	Entity Data Model XML
HTML	Hypertext Markup Language
LINQ	Language-Integrated Query
MVC	Model-View-Controller
MVP	Model-View-Presenter
MVVM	Model-View-ViewModel
ORM	Objektově relační mapování
RFID	Radio Frequency Identification
SQi	SQL Injection
SQL	Structured Query Language
UWP	Universal Windows Platform
VB	Visual Basic
WinUI	Windows UI Library
WPF	Windows Presentation Foundation
XAML	Extensible Application Markup Language
XML	Extensible Markup Language
XSS	Cross Site Scripting

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM KÓDŮ

SEZNAM PŘÍLOH