Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

User plugin pro projekt Metronom4u.cz

Zdeněk Krast

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Zdeněk Krast

Informatika

Název práce

User plugin pro projekt Metronom4u.cz

Název anglicky

User plugin for the Metronom4u.cz project

Cíle práce

Cílem práce je analyzovat a formou softwarového doplňku rozšířit aplikaci Metronom4u.cz tak, aby umožnila uživatelské přihlášení (a administrátorské). Každý uživatel bude mít vlastní uživatelský účet na kterém bude moci mít kredity. Kredit uživatel bude používat k placení reklam na Metronom4u.cz, které si sám vloží do sekce reklamního panelu. Zásuvný modul bude určen k integraci s systémem Metronom buď na přímo přes GIT nebo jinou formou (REST rozhraní).

Metodika

Postupujte dle následující metodiky:

- Prostudujte literaturu z oblasti tvorby zásuvných modulů pro .Net a C# aplikace
- Analyzujte zadaný úkol a navrhněte vhodné uživatelské cíle
- Vytvořte zásuvný modul
- Modul otestujte
- Napište dokumentaci k instalaci modulu
- Definujte závěry

Doporučený rozsah práce

30-40

Klíčová slova

Zásuvný modul, Plugin, Metronom4u.cz, .Net, Návrh, Analýza

Doporučené zdroje informací

Josef Pavlíček and Petra Pavlíčková, Usability Testing Methods and Usability Laboratory Management, Submitted: June 10th, 2022 Reviewed: November 23rd, 2022 Published: December 29th, 2022 DOI: 10.5772/intechopen.109140

Předběžný termín obhajoby

2023/24 LS – PEF

Vedoucí práce

Ing. Josef Pavlíček, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 7. 3. 2024

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 11. 3. 2024

doc. Ing. Tomáš Šubrt, Ph.D.

Děkan

V Praze dne 14. 03. 2024

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci „User plugin pro projekt Metronom4u.cz“ jsem vypracoval(a) samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor(ka) uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 14. 3. 2024 _____

Poděkování

Rád(a) bych touto cestou poděkoval(a) Ing. Josefu Pavlíčkovi, Ph.D., za cenné rady, odborné vedení a trpělivost během celého procesu tvorby práce.

User plugin pro projekt Metronom4u.cz

Abstrakt

Tato bakalářská práce se zaměřuje na návrh a vytvoření user pluginu pro projekt Metronom4u.cz. V teoretické části jsou popsány zásuvné moduly, jejich výhody, historie a designové vzory. Dále se pak teoretická část věnuje technické stránce technologií a nástrojů a jejich použití pro samotnou tvorbu zásuvného modulu.

Na základě teoretických podkladů je v praktické části vytvořen zásuvný modul pro uživatele včetně jeho implementační dokumentace. Zásuvný modul, společně s dokumentací, je implementován a testován. Na základě výsledků z testování je zásuvný modul zhodnocen a následně jsou navrženy změny a vylepšení jeho funkcionalit.

Výsledkem této práce bude nasazení otestovaného a o návrhy změněného a vylepšeného zásuvného modulu pro projekt Metronom4u.cz. na jehož webu bude umístěn.

Klíčová slova: Zásuvný modul, Plugin, Metronom4u.cz, .Net, Návrh, Analýza

User plugin for the Metronom4u.cz project

Abstract

This bachelor thesis focuses on the design a creation of a user plugin for Metronom4u.cz project. The theoretical part describes the plugins, their advantages, history and design patterns. The theoretical part is also devoted to the technical aspects of technologies and tools, and their use for actual creation of plugin.

Based on the theoretical background, the practical part creates a plugin for the user, including its implementation documentation. The plugin together with the documentation is implemented and tested and based on the results from the testing, the plugin is evaluated and then changes and improvements to its functionalities are proposed.

The result of this work will bet he deployment of the testeted, modified and improved plugin for the project Metronom4u.cz on whose website it will be placed.

Keywords: Plug-in module, Plugin, Metronom4u.cz, .Net, Design, Analysis

Obsah

1 Úvod

Projekt Metronom4u.cz představuje inovativní online platformu zaměřenou na hudební sféru, která se inspiruje úspěšnými koncepty, jakými jsou např. Herohero a Patreon. Tato platforma vytváří prostor pro hudebníky a kapely, kde mohou sdílet svůj tvůrčí obsah a zároveň ho monetizovat. Klíčovým prvkem projektu je rozdělení uživatelů do dvou rovin – tvůrci obsahu, kteří nahrávají a zpeněžují své hudební produkce, a fanoušci, kteří si mohou zakoupit přístup k exkluzivnímu obsahu svých oblíbených umělců.

Platforma Metronom4u.cz nejenže poskytuje možnost financování tvůrčího procesu pro umělce, ale zároveň nabízí jedinečnou formu interakce mezi tvůrci a jejich fanoušky. S cílem podpořit komunitní duch a zvýšit viditelnost tvůrců, je součástí projektu i bezplatný metronom s prostorově cílenou reklamou. Kapely a hudební umělci mohou rezervovat reklamní sloty na metronomu, což jim umožňuje efektivní propagaci a šíření povědomí o jejich obsahu.

Projekt Metronom4u.cz se tak stává inovativním mostem mezi tvůrci obsahu a jejich publikem. Nabízí nový způsob financování tvůrčího procesu v hudebním průmyslu a zároveň podporuje komunitní interakci a spolupráci.

Tato práce se zaobírá problematikou tvorby uživatelského pluginu, její dokumentací, verifikací integrity vstupů pro přihlášení a registraci a prací s databází.

2 Cíl práce a metodika

2.1 Cíl práce

Cílem práce je analyzovat a formou softwarového doplňku rozšířit aplikaci Metronom4u.cz tak, aby umožnila uživatelské přihlášení (a administrátorské). Každý uživatel bude mít vlastní uživatelský účet, na kterém bude moci mít kredity. Kredit bude uživatel používat k placení reklam na Metronom4u.cz, které si sám vloží do sekce reklamního panelu. Zásuvný modul bude určen k integraci se systémem Metronom, a to buď na přímo přes GIT nebo jinou formou (REST rozhraní).

2.2 Metodika

Postupujte dle následující metodiky:

- prostudujte literaturu z oblasti tvorby zásuvných modulů pro .Net a C# aplikace,
- analyzujte zadaný úkol a navrhněte vhodné uživatelské cíle,
- vytvořte zásuvný modul,
- modul otestujte,
- napište dokumentaci k instalaci modulu,
- definujte závěry.

3 Teoretická část práce

3.1 Úvod do zásuvných modulů

Zásuvný modul představuje počítačový software, který rozšiřuje funkčnost hostitelského programu bez zásadních změn v jeho jádru. Tato inovativní komponenta umožňuje přidání nových schopností do již existujícího programu, což může být obzvláště užitečné v oblastech digitálního zvuku, videa a prohlížení webu. Díky zásuvným modulům mají programátoři možnost aktualizovat hostitelský program, aniž by narušili uživatelské prostředí samotného programu. Zaručuje tak plynulý a kontinuální chod a zážitek pro uživatele. (Britannica, 2022)

3.1.1 Historie a vývoj

Rozvoj zásuvných modulů datujeme do druhé poloviny 20. století a přesněji do 90. let, kdy došlo k výraznému zlepšení výkonu softwaru u mikroprocesorů. Jednou z prvních aplikací, která masivně využívala zásuvné moduly, byl program Photoshop od společnosti Adobe, který byl specializovaný na zpracování a úpravu obrázků. Prvopočátky zásuvných modulů přinášely rozšířené funkce, jako například speciální efekty, filtry a další možnosti manipulace s obrázky, čímž vytvářely nové dimenze tvorby a úpravy graficky přímo v rámci prostředí programu Photoshop. Tímto způsobem se zásuvné moduly staly klíčovým prvkem v průkopnictví v modularitě softwaru. (Britannica, 2022)

Moderní zásuvné moduly poskytují mnoha počítačovým programům značnou flexibilitu. I když byly textové procesory nebo webové prohlížeče původně navrženy pečlivě, tvůrci softwaru nemohou předvídat všechny možné funkce, které by budoucí uživatelé mohli vyžadovat. V případě, že program nemá architekturu zásuvných modulů, jsou uživatelé nuceni přepínat mezi dvěma programy, aby provedli žádanou práci, nebo se uživatelé musejí spoléhat na to, že požadovaná funkce bude zahrnuta v budoucí aktualizaci softwaru. Zásuvné moduly tímto řeší situaci integrací s uživatelským programem. (Britannica, 2022)

Sdílením architektury zásuvných modulů s jinými firmami vývojáři softwaru vytvářejí užitečné synergické efekty mezi svými vlastními produkty a různými

přidruženými produkty. Každý zásuvný modul přidává hodnotu k hostitelskému programu a úspěch hostitelského programu zase posiluje hodnotu všech zásuvných modulů. Tímto způsobem se vytváří harmonická interakce, kde vzájemné posílení a integrace mezi jednotlivými moduly a hostitelským programem vede k celkovému zdokonalení a rozšíření funkcionalit celého softwarového ekosystému. (Britannica, 2022)

3.2 Výhody zásuvných modulů

Výhody sestavování softwaru z několika modulů jsou již dobře známy a dlouho aplikovány. Zapouzdření konkrétních funkcionalit do modulů a poskytnutí definovaného rozhraní vedlo k vývoji přístupu orientovaného na komponenty v softwarovém inženýrství. Tento přístup umožňuje kombinaci komponent a tvorbu systémů. Klíčovým rozdílem mezi architekturami založenými na zásuvných modulech a jinými komponentovými architekturami je fakt, že zásuvné moduly jsou volitelné a nejsou povinnými komponentami. Systém by měl být schopen fungovat bez ohledu na přítomnost nebo nepřítomnost zásuvných modulů, ale zároveň nabízí variabilitu funkčnosti v závislosti na tom, které zásuvné moduly jsou aktuálně přidány. Využitím zásuvných modulů lze efektivně řešit následující problémy:

- zvýšení komplexity systému o další funkcionality,
- optimalizace načítání rozsáhlých systémů pouze s potřebným softwarem pro aktuální situaci,
- aktualizace dlouhodobě provozovaných aktualizací bez nutnosti restartu,
- integrace rozšíření vyvinutých externími vývojáři. (Chatley, Eisenbach, Magee, 2003)

3.3 Designové vzory pro zásuvné moduly

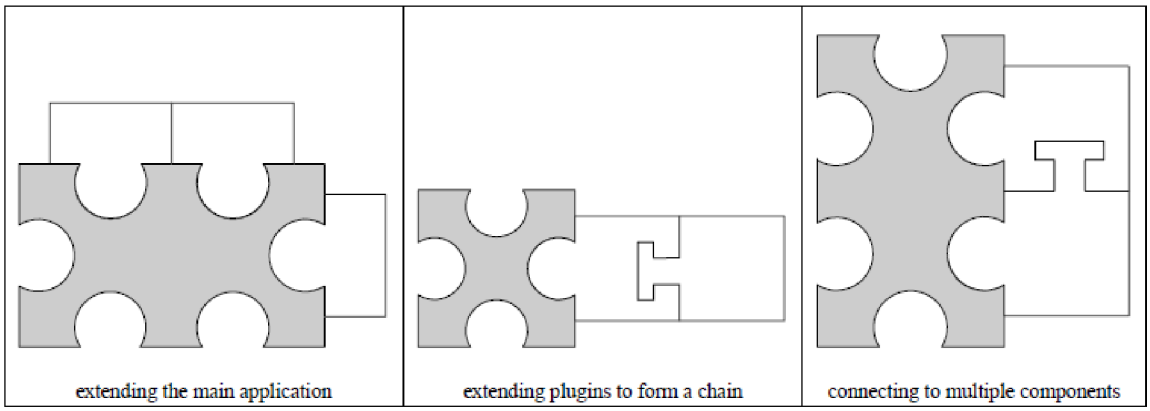
Pokud zvážíme, jak se komponenty v rámci architektury zásuvných modulů spojují, může nám to připomínat skládání dílků do skládačky. Jestliže má dílek správný tvar, můžeme ho snadno spojit s jiným dílkem, který disponuje odpovídajícím otvorem.

Základní aplikace tedy poskytuje několik otvorů, do nichž lze zapojit volitelné komponenty, jež přidávají dodatečné funkce, které obsahují soubory tříd a rozhraní. Otvory symbolizují rozhraní, které je známé hlavní aplikaci. Kolíky pak dále zastupují třídy v komponentách zásuvných modulů, jež implementují toto rozhraní. Rozhraní definuje signatury metod v dané třídě. (Chatley, Eisenbach, Magee, 2003)

Součástka s různými tvary otvorů může vést ke složitějším konfiguracím součástí, jako jsou ty, které jsou vyobrazeny na zapojení vpravo. To může být využito například v integrovaném vývojovém prostředí, kde prvním doplňkem je prohlížeč nápovědy a druhým je ladící nástroj. Tím, že se umožní vzájemné propojení těchto dvou zásuvných modulů, zapříčiní zlepšení poskytované nápovědy. (Chatley, Eisenbach, Magee, 2003)

Případ zapojení nalevo je například případ, kdy budeme chtít omezit počet připojitelných zásuvných modulů. Může to být v případě, že zásuvný modul spotřebovává zdroj, který má hlavní aplikace k dispozici a jehož množství je omezené. Omezení kardinality lze také použít k omezení tvarů, které může konfigurace nabývat. (Chatley, Eisenbach, Magee, 2003)

Prostřední zapojení funguje aplikací kardinality a zřetězení. Taková situace může být užitečná, například když chceme zřetězit video filtry, kde by každý filtr mohl přijímat video stream na vstupu a poskytovat odlišný výstupní stream. (Chatley, Eisenbach, Magee, 2003)



Obrázek 1: Možné způsoby zapojení zásuvných modulů (Chatley, Eisenbach, Magee, 2003)

3.4 Dependency Injection

Dependency Injection je programátorský koncept, který zajišťuje nezávislost třídy na svých závislostech. Tento přístup dosahuje oddělení vytvoření objektu od jeho použití, což podporuje princip inverze závislostí a jediné odpovědnosti (SOLID), jejichž hlavním cílem je zlepšit znovupoužitelnost kódu a minimalizovat tím potřebu častých změn ve třídě. Dependency Injection podporuje tyto cíle tím, že odděluje proces vytvoření a použití objektu, což umožňuje výměnu závislostí bez nutnosti změny třídy, která je používá. Tímto minimalizujeme rizika nutnosti úpravy třídy jen kvůli změně jedné z jejich závislostí. (Stackify, 2023)

3.4.1 Techniky Dependency Injection

Pro přerušení závislostí mezi třídami vyšší a nižší úrovně lze zavést rozhraní. Pokud je tak učiněno, obě třídy budou záviset na rozhraní a ne na sobě navzájem. Tento přístup je popsán principem inverze závislostí, kde přináší výhody již ve zmiňované znovupoužitelnosti kódu a minimalizuje vlnový efekt při změně tříd nižší úrovně. I přes pečlivou implementaci toho principu má tento princip svá úskalí, a to, že stále zůstává závislost na třídě nižší úrovně. Rozhraní pouze odděluje použití třídy nižší úrovně, nikoliv však její instanci. (Stackify, 2023)

Cílem techniky Dependency Injection je eliminovat tuto závislost tím, že se oddělí proces použití od vytvoření objektu, a tím se snižuje množství rutinního kódu a zvyšuje se flexibilita. (Stackify, 2023)

3.5 Inverze řízení

V softwarovém inženýrství je inverze řízení návrhovým vzorem, kdy vlastní části počítačového programu přebírají řízení. Tento vzor, historicky označovaný jako inverze, mění směr řízení ve srovnání s procedurálním programováním, kde v procedurálním programování kód programu opakovaně volá vlastní knihovny, které vykonávají obecné úlohy. Opakem při inverzi řízení je framework, který řídí volání vlastního kódu. Inverze řízení nabyla své uplatnění od doby vzniku prostředí s grafickým uživatelským rozhraním (GUI) a nadále je využíván také pro webové servery. (Sweet, 1985)

Programování řízené událostmi využívá inverzi řízení, což znamená, že kód aplikace se soustředí pouze na zpracování událostí. Běhové prostředí nebo framework se stará o obsluhu smyčky událostí a posíláním zpráv. (Laputan, 1988)

3.6 Programovací jazyk C#

Programovací jazyk C#, anglicky vyslovovaný jako „See Sharp“, je moderní, snadno použitelný, bezpečný a objektově orientovaný programovací jazyk. Pochází z rodiny jazyků C. Jazyk C# je standardizován organizací ECMA International standardem ECMA-334 a organizací ISO/IEC standardem ISO/IEC 23270. Pro prostředí .NET framework je tímto vyhovující pro implementaci obou výše zmíněných norem. (Hejlsberg a kol., 2010, s. 1)

3.6.1 Využití jazyka

Svojí stavbou je zaměřený spíše na objektové programování, ale zároveň poskytuje podporu pro programování orientované na komponenty. V dnešním světě návrhu softwaru se stává nedílnou součástí ve využití softwarových komponent, které představují nezávislé a jednoznačně popsatelné balíčky funkcí. Komponenty mají atributy poskytující deklarativní informace o komponentě a mohou obsahovat svoji dokumentaci. Podpora jazyku C# a jeho poskytované konstrukce z něho činí přirozený jazyk k vytváření a používání softwarových komponent. (Hejlsberg a kol., 2010, s. 1)

3.6.2 Klíčové funkce

Při vývoji odolných a spolehlivých aplikací přináší jazyk C# několik klíčových funkcí. Patří sem automatický Garbage collection, který efektivně uvolňuje paměť zabranou objekty, které se nepoužívají. Dále strukturovaný a rozšiřitelný přístup k detekci a zpracování výjimek umožňuje efektivní řešení chyb. Další konstrukce jazyka zabrání čtení z neinicializovaných proměnných, překročení hranic polí nebo provedení nekontrolovaných typových operací, a přispívá tak k celkové spolehlivosti a bezpečnosti aplikací. (Hejlsberg a kol., 2010, s. 1)

3.6.3 Jednotný typový systém a jeho flexibilita

Jazyk C# implementuje jednotný typový systém, ve kterém se všechny typy proměnných zdědily od jednoho kořenového objektového typu. Tato jednotná vlastnost umožňuje všem typům provádět společný soubor operací a zajišťuje konzistenci při práci s hodnotami libovolného typu. (Hejlsberg a kol., 2010, s. 1)

3.6.4 Verzování a zajištění kompatibility stabilního vývoje

S cílem zajištění kompatibilního vývoje programů a knihoven v jazyce C# v průběhu času vývojáři při návrhu tohoto jazyka kladli důraz na verzování. Opačně tomu může být v mnoha jiných programovacích jazycích, kde je otázka verzování často opomíjena. Může tak způsobovat častější poruchy programů psaných v těchto jazycích například v aktualizaci závislých knihoven na novější verze. (Hejlsberg a kol., 2010, s. 1)

3.7 .NET

.NET je multiplatformní vývojářská platforma s otevřeným zdrojovým kódem, která poskytuje bezplatné prostředí pro vytváření různých druhů aplikací. Tato platforma podporuje spouštění programů psaných v několika programovacích jazycích. Nejznámější z nich jsou právě C# a F#. Jeho základním stavebním kamenem je vysoce výkonný runtime, který je efektivně využíván mnoha rozsáhlými aplikacemi v produkčním prostředí. Byla koncipována s cílem poskytovat vývojářům produktivitu, vysoký výkon, zabezpečení a spolehlivost. Automatická správa paměti je realizována pomocí Garbage collectoru a odstraňuje tak starost o ruční správu paměti. Platforma .NET je typově a paměťově bezpečná díky striktním překladačům jazyka. (Microsoft, 2024)

Platforma dále podporuje souběžnost prostřednictvím primitiv async, wait a task, jež umožňují efektivní zpracování asynchronních operací. Má velkou sadu knihoven, které jsou optimalizovány pro výkon v různých operačních systémech a architekturách čipů. Výše zmíněné kombinace vlastností dělají z platformy .NET atraktivní nástroj pro vytváření robustních a efektivních aplikací. (Microsoft, 2024)

O .NET se stará společnost Microsoft a taktéž jeho komunita. Je pravidelně aktualizován a díky tomu mohou uživatelé nasazovat bezpečné a spolehlivé aplikace do vývoje. (Microsoft, 2024)

3.7.1 Komponenty

.NET obsahuje následující součásti:

- Runtime – který spouští kód aplikace a zajišťuje běh programů.
- Knihovny – poskytuje již napsané užitečné funkce, které lze opakovaně volat, příkladem může být parsování, aby usnadnily vývoj aplikací.
- Překladač – stará se o kompilaci zdrojového kódu do kódu spustitelného.
- SDK a další nástroje – SDK (Software Development Kit) a další nástroje umožňují vývojářům vytvářet, spravovat a monitorovat aplikace s použitím moderních pracovních postupů.
- Zásobníky aplikací – poskytují rámce pro vývoj specifických typů aplikací, například ASP.NET Core pro webové aplikace nebo Windows Forms pro desktopové aplikace. Tyto zásobníky umožňují snadný vývoj a psaní aplikací pro konkrétní potřeby. (Microsoft, 2024)

3.7.2 Free open source

.NET spravuje Microsoft a jeho komunita na GitHubu. Nachází se v několika úložištích. Jedná se bezplatný open source nástroj, který je projektem .NET Foundation. (Microsoft, 2024)

3.7.3 Ekosystém .NET

Existuje několik druhů variant .NET. Každá z nich je zaměřena na podporu různých typů aplikací. Rozdíly těchto variant vyplývají zejména z historických a technických důvodů. Druhy různých implementací můžeme dělit na:

- .NET Framework – Jedná se o původní verzi .NET, která je aktivně udržována a podporována. Nabízí širokou škálu možností pro vývoj aplikací pro prostředí Windows a Windows Server.

- Mono – Mono je komunitní open source implementace .NET. Je multiplatformní a podporuje rozhraní .NET Framework. Aktivně ji lze nalézt ve vývoji aplikací pro Android, iOS a WebAssembly.
- .NET CORE – Je multiplatformní open source a nejmodernější verzí optimalizovanou pro cloudové prostředí. Stále je kompatibilní s .NET Framework a je aktivně podporována pro běh na Linuxu, macOS a Windows (Microsoft, 2024)

3.7.4 NuGet

Klíčovým prvkem na každé moderní vývojové platformě je mechanismus umožňující vývojářům vytvářet, sdílet a využívat užitečný kód. Tento kód je seskupen pomocí balíčků, které zahrnují zkompilovaný kód spolu s dalším obsahem potřebným pro projekty, které tyto balíčky využívají. V prostředí .NET tuto funkci k podpoře sdílení kódu zastává nástroj NuGet od společnosti Microsoft. Definuje postup pro vytváření, hostování a spotřebovávání balíčků v prostředí .NET a poskytuje nástroje pro každou z těchto rolí. Zjednodušeně řečeno lze říci, že NuGet je obdoba souboru ve formátu ZIP s příponou .nupkg obsahující zkompilovaný kód a manifest s popisnými informacemi, jako je například verze balíčku. Vývojáři, kteří chtějí kód sdílet, tyto balíčky vytvoří a publikují na veřejném nebo soukromém hostiteli. Konzumenti těchto balíčků je potom implementují do svých projektů a následně volají funkce balíčku ve svém kódu. (Microsoft, 2022)

3.8 Entity Framework Core

Entity Framework Core, zkráceně EF Core, představuje knihovnu, kterou vývojáři softwaru používají k interakci s databázemi. Tato knihovna funguje jako objektově-relační mapovač (O/RM), což znamená vytváření spojení mezi relační databází s vlastním rozhraním API a světem objektově orientovaného softwaru zahrnujícího třídy a softwarový kód. Hlavním přínosem EF Core je možnost rychle psát kód v jazyce, který vývojáři znají lépe než třeba SQL. EF Core je multiplatformní, tudíž je možné ji spustit na operačních systémech Windows, Linux a také na Apple. Jeho název obsahuje přívlastek Core jako součást iniciativy .NET Core. .NET pokrývá širokou

škálu oblastí desktopu, webu, mobilních zařízení, her, IoT, cloudu a umělé inteligence. (Smith, 2021, s. 3)

Nynější EF Core vychází z dlouholetých zkušeností získaných z předchozích verzí s ohledem na zpětnou vazbu od uživatelů. Postupem času přinesl i zásadní změnu v podobě schopnosti pracovat s nerelačními databázemi, což nebylo původním záměrem. Předpokladem pro práci s Entity Framework Core jsou zkušenosti s vývojem v .NET pomocí jazyka C# a alespoň základní představa o relačních databázích. (Smith, 2021, s. 4)

3.8.1 Mapování Entity Framework a databáze

EF Core, jakožto objektově-relační mapovač (O/RM), umožňuje mapovaní mezi relačními databázemi a světem tříd a softwarového kódu v rámci .NET Frameworku. Základní principy tohoto mapování jsou přehledně zobrazeny v tabulce č. 1, která ilustruje propojení dvou světů za pomocí EF Core v prostředí .NET. (Smith, 2021, s. 7)

Relační databáze	.NET software
Tabulka	.NET třída
Sloupce tabulky	Vlastnosti třídy
Záznam	Prvky .NET kolekcí – př. List
Primární klíče: unikátní záznam	Unikátní instance
Cizí klíče: definují vztah	Odkaz na jinou třídu
SQL – př. „WHERE“	.NET LINQ – př. „Where(p =>…“

Tabulka 1: EF Core mapování mezi databází a .NET softwarem (Smith, 2021, s. 7)

3.9 Práce s databází

Efektivní správa dat je klíčovým prvkem využívání počítačových systémů. Pro tuto úlohu se v praxi nejčastěji využívají databáze. Databáze představuje sdílenou a integrovanou počítačovou strukturu, která pečlivě uchovává a spravuje soubor následujících údajů:

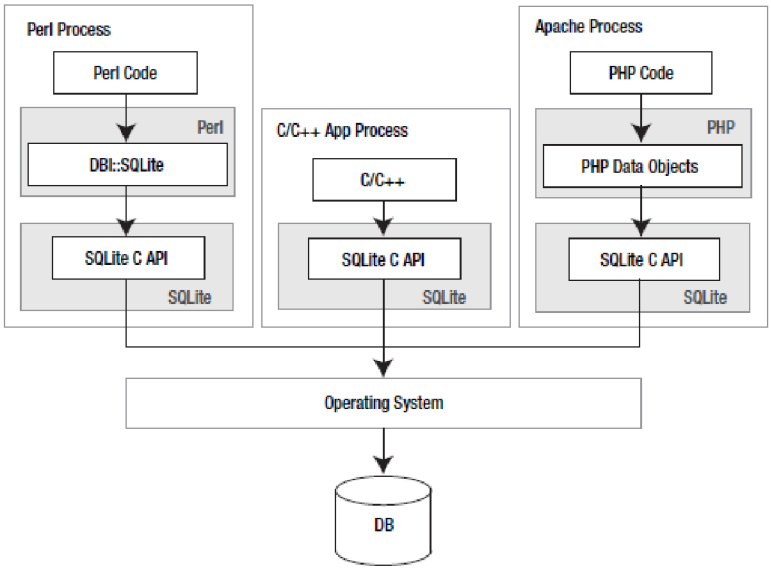
- data koncového uživatele – surová fakta, která mají přímý vliv na koncového uživatele,
- metadata – informace o datech, které integrují a spravují data koncového uživatele. (Coronel, 2019, s. 6)

Metadata hrají klíčovou roli při popisu vlastností dat a vzájemných vztahů v databázi. Ukládání informace o názvech datových prvků, typů hodnot a možnosti ponechání prvku prázdného může být typickým příkladem. Tato metadata rozšiřují hodnotu a význam dat a umožňují nám vnímat databázi jako shluk sebe popisujících se dat. (Coronel, 2019, s. 7)

Systém správy databází (DBMS) slouží jako soubor programů, který efektivně řídí přístup k datům v databázi a spravuje tak její strukturu. (Coronel, 2019, s. 7)

3.9.1 Využití SQLite pro ukládání dat

SQLite je relační databáze s otevřeným zdrojovým kódem. Byla navržena tak, aby poskytovala pohodlný způsob správy dat bez režijních nákladů, které často přináší specializované systémy pro správu relačních databází. SQLite je znám pro svou vysokou přenositelnost, snadné použití, efektivitu, kompaktnost a spolehlivost. SQLite je integrovaná databáze, která není spuštěna jako samotný proces, ale spolupracuje s aplikací v jejím vlastním prostředí. Kód je tedy přímo vložen do programu, který tuto databázi hostí. Jednou z výhod takto stavěné databáze je, že není třeba složité konfigurace sítě ani správy. Není třeba se tedy zabývat komplikovanými oprávněními a privilegii, firewallem a adresami. Klient i server sdílí společný proces, což snižuje režii spojenou se síťovým voláním a zjednodušuje správu databáze a následné nasazení aplikace. (Allen, 2010, str. 1)



Obrázek 2: SQLite v hostovském procesu (Allen, 2010, s. 2)

3.10 Využití GitHubu

GitHub vytváří prostředí, ve kterém je možné ukládat kód na vzdálený server a sdílet ho takto s dalšími lidmi. Umožňuje více uživatelům najednou přidávat, upravovat nebo mazat kód ve stejném souboru nebo projektu. Vysvětlení fungovaní GitHubu se dá přirovnat ke Google Docs, které jsou jakýmsi online místem, kde společně jeden či více lidí píší do souboru, aniž by museli posílat různé verze pomocí emailů. (Guthals, Haack, 2019, s. 7)

3.10.1 Řízení verzí

Systémy pro správu verzí, které jsou také známy jako systémy pro kontrolu zdrojů nebo SCM, jsou programy, které sledují každou verzi každého souboru v projektu, ukládají časová razítka, kdy byly tyto verze vytvořeny a zaznemenávají autora, který změny provedl. Díky těmto systémům mohou programátoři experimentovat a dělat chyby bez obav, že by museli začít uplně od začátku. Využití takového systému pro programování může vypadat následovně:

1. Vytvoření projektu, obvykle vytvořením složky v počítači.
2. Informovat vybraný systém pro správu verzí, aby sledoval změny ve vytvořené složce.

3. Vždy, když je dosažen funkční stav nebo je žádoucí podniknout další kroky, je třeba uložit projekt jako novou verzi.
4. Pokud je žádoucí vrátit se ke starší verzi, je nutné požádat systém, aby obnovil vybranou předchozí verzi. (Guthals, Haack, 2019, s. 8)

3.10.2 **Řízení verzí v GitHubu**

GitHub využívá technologie pro správu verzí Git. Git představuje druh systému pro správu verzí, který je dostupný zdarma a s otevřeným zdrojovým kódem. Produkty poskytované GitHubem usnadňují používání systému Git. (Guthals, Haack, 2019, s. 8)

3.11 **Výsledek rešerše**

Po prozkoumání historie a vývoje zásuvných modulů, analýze výhod, zkoumání designových vzorů, C# programovacího jazyka, .NET ekosystému, klíčových technik jako Dependency Injection a Inverze řízení si nyní shrňme klíčové poznatky.

Zásuvné moduly mají bohatou historii a prošly významným vývojem. Jejich úspěch spočívá v modularitě a schopnosti snadného přidávání či výměny funkcí v rámci aplikací. Využitím zásuvných modulů přináší několik výhod, a sice zlepšení modularizace, znovupoužitelnosti a možnosti dynamické konfigurace aplikace. Při implementaci zásuvných modulů je vhodné používat designové vzory, které usnadňují správu, rozšíření a údržbu aplikace. Technika Dependency Injection poskytuje elegantní způsob správy závislostí mezi komponentami aplikace, což usnadňuje testovatelnost a flexibilitu. Inverze řízení hraje klíčovou roli při oddělovaní vytvoření objektu od jeho použití, což podporuje SOLID principy a umožňuje efektivní vývoj a údržbu kódu. Jazyk C# nabízí výhody jako jednotný typový systém, silné funkcionální prvky a stabilní vývoj s podporou verzování. .NET ekosystém poskytuje široké možnosti pro vývoj aplikací, a to nejen pro platformu Windows, ale i pro další operační systémy, díky .NET Core. Entity Framework Core usnadňuje práci s databázemi a nabízí mapování mezi objektovým modelem a relační databází. Stejně tak i SQLite pro ukládání dat, což přináší jednoduchost a přenositelnost. GitHub poskytuje efektivní nástroje pro správu verzí, což je klíčový prvek pro týmovou spolupráci a sledování změn v kódu.

Tímto shrnutím lze poskytnout komplexní pohled na klíčové prvky týkající se zásuvných modulů, programovacího jazyka C#, .NET ekosystému a nástrojů využitých v práci.

4 Analýza problému

V této části práce bude podrobně popsán problém, který má být řešen prostřednictvím zásuvného modulu, stejně jako důkladná analýza současné situace a potřeb uživatelů. Analýza problému je klíčovým krokem pro úspěšné navrhování a implementaci řešení.

4.1 Uživatelské cíle

Uživatelské cíle v této kapitole jsou strukturovány do dvou hlavních skupin: předplatitelé obsahu a tvůrci obsahu. Na počátku je každý uživatel automaticky zařazen do role předplatitele obsahu, což odpovídá základnímu režimu použití platformy. Tímto způsobem mají uživatelé okamžitý přístup k obsahu a funkcím, které platforma nabízí.

Nicméně, uživatel má možnost kdykoliv přidat novou roli, a to roli na tvůrce obsahu, pokud si přeje aktivně přispívat k obsahu platformy. Tato flexibilita umožňuje uživatelům rozšířit své možnosti a využít plný potenciál platformy podle vlastních individuálních zájmů a schopností.

Každá skupina uživatelů má své specifické potřeby a očekávání, jež se liší v závislosti na jejich rolích v rámci platformy. Systém je navržen tak, aby byl schopen dynamicky reagovat na tyto potřeby a poskytoval uživatelům prostředí, které je přizpůsobeno jejich individuálním preferencím a cílům.

4.1.1 Předplatitelé obsahu

Předplatitelé obsahu v této platformě hledají především kvalitní a zajímavý hudební obsah. Jejich hlavním cílem je získat přístup k exkluzivním nahrávkám, hudebním tutoriálům, koncertům a dalším speciálním materiálům poskytovaným tvůrci obsahu. Kromě obsahu chtějí mít také pohodlný proces pro předplatné, který zahrnuje jednoduchou registraci, volbu předplatného a pohodlné platební možnosti. Interaktivita s tvůrci obsahu je pro ně důležitá, a to prostřednictvím možnosti komentování, poskytování zpětné vazby a účasti na tvorbě obsahu.

4.1.2 Tvůrce obsahu

Tvůrci obsahu na této platformě mají za cíl efektivně zpeněžit svou tvorbu. To zahrnuje nabízení placeného obsahu nebo přijímání finančních příspěvků od předplatitelů. Kromě toho potřebují nástroje pro správu a analýzu svého obsahu, což zahrnuje sledování výkonu obsahu, počet předplatitelů a další relevantní statistiky. Dále si cení flexibility v tvorbě obsahu, což znamená možnost vytvářet různorodý obsah a experimentovat s různými styly, přičemž zároveň získávají zpětnou vazbu od své komunity.

Celkově je platforma navržena tak, aby uspokojila potřeby obou skupin uživatelů. Poskytuje intuitivní prostředí pro interakci a tvorbu obsahu v oblasti hudby, podporuje transparentní proces předplatného a umožňuje tvůrcům dosáhnout svých cílů vytvářením a sdílením kvalitního hudebního obsahu.

4.2 Datové entity

V rámci tohoto projektu je důležité detailně popsat strukturu dat, která budou využívána. Pro účely této analýzy bylo identifikováno několik klíčových aspektů, které by měly být zahrnuty.

V první řadě je nutné identifikovat hlavní entity, které budou v projektu používány. To můžeš zahrnovat uživatele, produkty, objednávky a další. Každá z těchto entit má své specifické atributy a vztahy s ostatními entitami.

Dalším důležitým aspektem je stanovení vztahů mezi jednotlivými entitami. To pomůže definovat strukturu datového modelu a zajišťuje tak konzistenci dat.

Specifikace atributů každé entity, je dalším krůčkem zdárné analýzy. Každý atribut bude mít svůj datový typ a může být povinný nebo volitelný.

Normalizace datového modelu je také důležitá pro minimalizaci redundance a zajištění konzistence dat. To může zahrnovat rozdělení dat do různých tabulek a využití primárních a cizích klíčů k zachování vztahů mezi nimi.

Dodatečné informace, jako jsou omezení integrity dat, indexy atp., jsou rovněž důležité pro správné navržení databáze a zajištění optimálního výkonu.

4.3 Vztahy mezi entitami

V rámci analýzy problému je také důležité detailně zkoumat vztahy mezi jednotlivými entitami v projektu. Tyto vztahy definují způsob, jakým jsou data propojena a jak spolu v jednotlivých částí aplikace interagují. V tomto projektu je několik typů vztahů, které jsou zásadní pro pochopení struktury dat.

4.3.1 Jedna k jedné (1:1)

Tento typ vztahu označuje situaci, kdy jedna instance jedné entity je spojena s jednou instancí druhé entity. Například každý uživatel může mít pouze jednu osobní identifikaci.

4.3.2 Jedna ke mnoha (1:N)

Vztah jedna ke mnoha znamená, že jedna instance jedné entity může být propojena s více instancemi druhé entity. V tomto projektu je to například zamýšleno ve vztahu ke kategorii produktu, která může obsahovat více produktů.

4.3.3 Mnoho k mnoha (M:N)

Tento vztah indikuje, že více instancí jedné entity je spojeno s více instancemi entity druhé. Kvůli své komplexnosti vyžaduje tento typ vztahu spojovací tabulky, které mapují vztahy mezi entitami. Ku příkladu může být fakt, že mnoho uživatelů může být přiřazeno ke mnoha různým skupinám či rolím.

4.4 Atributy entit

V analýze dat je nezbytné detailně prozkoumat atributy jednotlivých entit, což jsou charakteristiky či vlastnosti, které popisují každou entitu v systému. Atributy poskytují informace o jednotlivých entitách a určují, jak jsou data strukturována. Uživatel v tomto projektu má následující atributy:

1. userId – Jednoznačný identifikátor uživatele v databázi
2. username – Uživatelské jméno, které je unikátní a slouží k identifikaci uživatele a komunikaci v systému.

3. firstName – Křestní jméno uživatele
4. lastName – Příjmení uživatele
5. email – E-mailová adresa uživatele, která slouží k přihlašování, komunikaci a obnovení údajů
6. birthDate – Datum narození uživatele, které se využívá k ověření věku a poskytuje informaci pro určité funkce systému
7. registrationDate – Datum registrace uživatele, který označuje čas, kdy uživatel vytvořil účet v systému
8. lastActiveDate – Datum poslední aktivity uživatele, který udává poslední čas, kdy byl aktivní v systému
9. userRating – Hodnocení uživatele, které vyjadřuje jeho reputaci či důvěryhodnost v systému
10. diallingCode – Telefonní předvolba uživatele, který se používá společně s telefoním číslem
11. phoneNumber – Telefonní číslo uživatele pro kontaktování

Atributy, které jsou zmíněné nejsou konečné a v budoucím sledu změn se vždy adaptují vývoji aplikace a databáze.

4.5 Dodatečné informace

Dodatečné informace, jako jsou omezení integrity dat, primární a cizí klíče, indexy atp., jsou klíčové z hlediska navržení a implementace databáze.

Omezení integrity dat definují pravidla, která zabezpečují konzistenci a platnost dat v databázi, což je zásadní pro prevenci chyb a zachování datové integrity.

Primární a cizí klíče jsou klíčové pro správné propojení tabulek a vytváření vztahů mezi nimi, které umožňují efektivní manipulaci s daty a udržení konzistence v databázi.

Indexy poté slouží k urychlení vyhledávání a dotazování nad daty, což zvyšuje výkon a efektivitu databázových operací. Správné využití těchto dodatečných informací je nezbytné pro optimalizaci výkonu databáze a zajištění efektivního fungování celého systému. Jejich pečlivé navržení a implementace přispívají k lepší správě dat a zvyšují celkovou spolehlivost a výkonnost aplikace.

4.6 Způsob ukládání dat

Ukládání dat v projektu je dalším z důležitých aspektů pro efektivní správu a přístup k datům. V této práci byla zvolena databáze jako primární úložiště dat. Pro implementaci databáze byl využit SQLite, což je lehká, snadno přenositelná relační databáze. Pro práci s daty byl použit Entity Framework, což je knihovna pro přístup k datům v .NET frameworku, která umožňuje jednoduchou a efektivní práci s relačními databázemi pomocí objektově-relačního mapování (O/RM). Tato kombinace poskytuje robustní a spolehlivý způsob ukládání dat, který odpovídá potřebám projektu.

4.7 Zálohování

Pro zajištění bezpečnosti a integrity dat je nutné implementovat pravidelné zálohování dat a mechanismy pro obnovu v případě výpadku nebo havárie. Jako hlavní prostředek pro ukládání a správu zdrojových kódů byla zvolena platforma GitHub, která poskytuje robustní nástroje pro verzování a správu kódu, což umožňuje pravidelné ukládání a synchronizaci dat mezi lokálním prostředím a cloudovou platformou. Díky tomu jsou data vždy chráněna a je možné je obnovit v případě ztráty nebo poškození.

4.8 Funkce

V rámci analýzy funkčních požadavků bylo identifikováno několik klíčových funkcí, které je nutné implementovat pro správné fungování zásuvného modulu. Mezi ty nejzákladnější požadované funkce můžeme zařadit registraci uživatele, přihlášení uživatele, hashování hesel, validace údajů a jejich verifikace.

4.8.1 Registrace uživatele

Registrace uživatele představuje důležitou funkcionalitu zásuvného modulu, která umožňuje novým uživatelům vytvořit si účet a získat tím přístup na platformu. Implementace tohoto mechanismu zahrnuje vytvoření uživatelského rozhraní, které umožní uživatelům zadat potřebné informace pro správnou registraci. Dále je součástí

této implementace zpracování a validace těchto informací, aby byla zajištěna bezpečnost a konzistence dat.

4.8.2 Přihlášení uživatele

Přihlášení je dalším důležitým faktorem zásuvného modulu, která umožňuje registrovaným uživatelům přístup k jejich účtu a funkcím. Implementace tohoto mechanismu zahrnuje v budoucnu vytvoření uživatelského rozhraní pro zadání přihlašovacích údajů. Dále je součástí implementace ověření těchto údajů pomocí správných autentizačních metod například pro ověření hesla pomocí hashovací funkce. Po úspěšném ověření je uživateli poskytnut přístup k jeho účtu.

4.8.3 Hashování hesel

Proces hashování hesel je klíčovým bezpečnostním aspektem při ukládání, verifikování či porovnávání hesel. Implementaci této funkcionality zahrnuje použití kryptografických hashovacích funkcí, které převedou heslo na unikátní řetězec znaků, tzv. hash. Tento hash je následně uložen v databázi místo samotného hesla, nebo je použit při komparaci a verifikování. Tímto způsobem není možné z databáze získat původní heslo, což zvyšuje bezpečnost uživatelských účtů. K tomuto mechanismu je dobré zvolit si takový typ hashovací funkce a nastavit délku a pravidla požadovaného hesla tak, aby systém byl odolný vůči útoku využívající Rainbow tables, což jsou předem vypočítané hashe z nejpoužívanějších hesel.

Implementace hashování zajišťuje ochranu před neoprávněným přístupem k heslům a snižuje riziko zneužití v případě úniku citlivých dat (data leak).

4.8.4 Validace a verifikace údajů

Verifikace a validace údajů jsou dalšími klíčovými procesy při manipulaci s uživatelskými informacemi v zásuvném modulu. Verifikace se zaměřuje na ověření správnosti formátu a struktury poskytnutých údajů, zatímco validace kontroluje, zda tyto údaje odpovídají stanoveným kritériím.

V zásuvném modulu je nutné provádět verifikaci a validaci údajů napříč celou aplikací, od registračního procesu až po manipulaci s uživatelskými profily.

To zahrnuje například ověřování formátu e-mailových adres, správnost telefonních čísel a další podobné mechanismy.

Při verifikaci se kontroluje, zda jsou zadané údaje v požadovaném formátu, například, zda e-mail obsahuje zavináč a tečku nebo zda telefonní číslo obsahuje pouze čísla. Validace pak zahrnuje kontrolu platnosti údajů, jako například ověření, že e-mailová adresa, či heslo v systému skutečně existují.

Tímto způsobem verifikace a validace údajů zajišťují, že zásuvný modul pracuje s pouze platnými a spolehlivými informacemi, což přispívá k bezpečnosti, integritě a efektivitě aplikace.

4.8.5 **Změna dat**

Změny citlivých dat, jako je například změna hesla uživatele, jsou důležitou součástí funkcionalit, kterými by měl být zásuvný modul vybaven. Tyto změny vyžadují zvláštní zacházení a procesy z důvodu zachování bezpečnosti a integrity uživatelských účtů.

Proces změny citlivých dat, jako je heslo, by měl být navržen tak, aby minimalizoval riziko neoprávněnému přístupu k těmto údajům. To obvykle zahrnuje ověření identity pomocí stávajících údajů a následné potvrzení nových údajů.

Zásuvný modul by měl poskytovat uživatelům intuitivní a bezpečný způsob změny citlivých dat.

5 Vlastní práce

V této části práce bude popsán průběh vývoje zásuvného modulu pro projekt Metronom4u.cz. Při vývoji budou použity technologie představené v teoretické části.

5.1 Návrh zásuvného modulu

Při návrhu zásuvného modulu bylo třeba zvážit několik klíčových aspektů, aby byl zajištěn efektivní a integrovaný prvek do celé struktury projektu Metronom4u.cz. Nejprve bylo nutné se zamyslet a zanalyzovat potřeby obou rovin uživatelů, tj. tvůrci obsahu a předplatitelé. Inspirace byla čerpána na portálech, které jsou založené na podobné, leč velmi všeobecné podstatě, přičemž tato platforma je určena přímo tvůrcům z hudebního odvětví. Tím je myšleno již zmíněné Herohero a Patreon. Byly identifikovány podstatné atributy a funkcionality, které odpovídají požadovaným očekáváním obou stran.

Dále bylo potřeba vybrat programovací jazyk. V potaz bylo bráno více programovacích jazyků, ale po důkladném zvážení byl vybrán programovací jazyk, který efektivně odpovídá potřebám tohoto projektu a má dostatečnou podporu vývojové komunity. Vzhledem k multiplatformní povaze projektu a jeho zaměření na webové technologie byl vybrán programovací jazyk C#.

Pro tento projekt byl programovací jazyk C# doplněn o .NET Framework, který, po zvážení všech výhod svého použití, byl vybrán především zejména pro jeho schopnost poskytovat stabilní a výkonný základ pro vývoj webových aplikací. .NET Framework nabízí také integraci s Entity Framework Core, který je také klíčovým prvkem projektu.

Pro návrh modulu bylo rozhodnuto také využít jeden z designových vzorů návrhu, který podporuje modularitu a rozšiřitelnost. Zohledněny byly také koncepty jako Dependency Injection a Inverze řízení (IoC), které přispějí k flexibilitě a udržitelnosti kódu.

Jelikož projekt využívá Entity Framework Core pro práci s databází, bylo zajištěno, aby návrh zásuvného modulu byl plně kompatibilní s tímto O/RM frameworkem. To umožňuje v projektu snadnou integraci nového modulu s existující databázovou strukturou.

Celkově byl kladen důraz na robustnost, flexibilitu a snadnou integraci modulu do celkové struktury projektu Metronom4u.cz. Tímto postupem je zajištěno, že návrh zásuvného modulu je dobře připraven pro praktickou implementaci.

5.2 Třída InputValidators.cs

V prostředí moderního internetu, kdy digitální platformy a online služby tvoří nedílnou součást každodenního života, je důležitost kontroly uživatelských vstupů klíčová pro zajištění bezpečnosti, integrity dat a kvality uživatelského zážitku. Každý, kdo navrhuje a implementuje webové aplikace, jako je projekt Metronom4u.cz, musí klást důraz na správnou validaci a bezpečnost vstupů od uživatelů.

V této kapitole je pozornost věnována třídě InputValidators.cs, která představuje klíčovou součást našeho bezpečnostního rámce. Bezpečnostní otázky, jako jsou SQL injection, cross-site scripting a další, jsou v dnešní době neustále vyvíjející se hrozby, které mohou zpochybnit celkovou integritu a spolehlivost aplikací.

5.2.1 Nastavení pravidel pro heslo

Bezpečnost uživatelských účtů je klíčovým prvkem každé webové platformy, a proto je nutné věnovat pozornost nastavení pravidel pro heslo v rámci projektu. Heslo je často první linií obrany proti neoprávněnému přístupu a jeho správná konfigurace může významně přispět k celkové bezpečnosti uživatelských účtů.

Nastavení pravidel pro heslo vychází ze zásad silného hesla a optimální ochrany účtů. Uživatelé jsou povinni zvolit heslo, které splňuje následující kritéria:

- Délka alespoň 10 znaků – zajišťuje dostatečnou komplexnost a odolnost proti hrubým útokům.
- Alespoň jedno velké písmeno – zvýrazňuje rozmanitost znaků a ztěžuje odhadnutí hesla.
- Alespoň jedno číslo – přidává číselný prvek pro zvýšení bezpečnosti.
- Alespoň jeden speciální znak – poskytuje další vrstvu ochrany tím, že zahrnuje speciální znaky.

Tato pravidla pro heslo nejen poskytují robustní ochranu uživatelských účtů, ale také reflektují současné normy pro bezpečnost hesel.

5.2.2 **Metoda int ValidateNewPasswordByRules**

Tato metoda slouží k validaci nového hesla podle určených pravidel. Jako svůj parametr má od uživatele zadané heslo. Zde je popis jednotlivých částí metody:

1. Podmínka délky hesla:
    a. if (password.Length < 10) return -1; // BAD_LENGTH
    b. Vrací hodnotu -1, pokud délka hesla není alespoň 10 znaků.
2. Počítání velkých písmen, čísel a speciálních znaků:
    a. int capital = 0;, int number = 0;, int specialChar = 0;
    b. Pomocí cyklu prochází všechny znaky v hesle a počítá počet velkých písmen, čísel a speciálních znaků.
3. Podmínka splnění pravidel pro heslo:
    a. if (capital > 0 && number > 0 && specialChar > 0) return 0; // OK
    b. Vrací hodnotu 0, pokud heslo splňuje požadovaná pravidla.
4. Výsledek, pokud pravidla nejsou splněna:
    a. return -2; // MISMATCH_RULES
    b. Vrací hodnotu -2, pokud heslo nedosahuje požadovaných pravidel (chybí alespoň jedno z požadovaných kritérií).

```
/// <summary>
/// Validace nového hesla podle definovaných pravidel.
/// </summary>
/// <param name="password">Nové heslo k validaci.</param>
/// <returns>
/// 0, pokud je heslo v souladu s pravidly.
/// -1, pokud je délka hesla mimo povolený rozsah.
/// -2, pokud heslo nesplňuje požadovaná pravidla.
/// </returns>
2 usages
public int ValidateNewPasswordByRules(string password)
{
    if (password.Length < 10) return -1; // BAD_LENGTH
    int capital = 0;
    int number = 0;
    int specialChar = 0;
    for (int i = 0; i < password.Length; i++){
        char c = password[i];
        if (char.IsUpper(c)){
            capital++;
        }
        else if (char.IsDigit(c)){
            number++;
        }
        else if (char.IsPunctuation(c) || char.IsSymbol(c)){
            specialChar++;
        }
    }

    if (capital > 0 && number > 0 && specialChar > 0){
        return 0; // OK
    }

    return -2; // MISMATCH_RULES
}
```

Obrázek 3: Metoda int ValidateNewPasswordByRules (Autor, 2024)

5.2.3 Metoda bool ValidateMail

Metoda ValidateMail slouží k jednoduché validaci e-mailové adresy. Skládá se z jednotlivých částí:

1. Podmínka pro obsah „@“ a „.“:
    a. if (mail.Contains('@') && mail.Contains('.'))
    b. Kontroluje, zda e-mail obsahuje uvedené znaky.
2. Rozdělení e-mailu na části:
    a. string[] parts = mail.Split('@');
    b. Rozděluje e-mail na dvě části uložené do pole o dvou prvcích podle znaku „@“.
3. Získání domény a následná kontrola správnosti:
    a. string domain = parts.Length > 1 ? parts[1] : null;
    b. Získává druhou část e-mailové adresy (domény).
    c. if (!string.IsNullOrEmpty(domain) && domain.Contains('.')) { return true; }
    d. Kontroluje, zda doména není prázdná, pokud není, vrací hodnotu true. V opačném případě vrací hodnotu false.

```
/// <summary>
/// Validace formátu e-mailové adresy.
/// </summary>
/// <param name="mail">E-mailová adresa k validaci.</param>
/// <returns>
/// True, pokud je formát e-mailové adresy platný.
/// False, pokud formát e-mailové adresy není platný.
/// </returns>
1 usage
public bool ValidateMail(string mail) {
    if (mail.Contains('@') && mail.Contains('.')) {
        string[] parts = mail.Split( separator: '@');
        string domain = parts.Length > 1 ? parts[1] : null;
        if (!string.IsNullOrEmpty(domain) && domain.Contains('.')){return true;}
    }
    return false;
}
```

Obrázek 4: Metoda bool ValidateMail (Autor, 2024)

5.2.4 Metody bool Is{attribute}FreeToUse

Metody Is{attribute}FreeToUse, jak již anglickým názvem napovídají, jsou metody, které na serveru ověřují, zda jsou jednotlivé atributy dostupné k vytvoření nového uživatele. Tímto se zamezí chybám na serveru v duplicitních záznamech, které mají být unikátní. Tyto metody využívají Entity Framework Core a SQLite pro práci s databází. AppDbContext představuje instanci DbContextu, která je konfigurována pro práci s databází. Metoda využívá LINQ (Language Integrated Query) a metody EF pro jednoduchý způsob dotazování nad databází a kontrolu dostupnosti požadovaných atributů. Každá metoda pak vrací parametr true nebo false, který určuje jejich dostupnost.

```
/// <summary>
/// Kontrola dostupnosti uživatelského jména.
/// </summary>
/// <param name="username">Uživatelské jméno k ověření.</param>
/// <returns>
/// True, pokud je uživatelské jméno dostupné k použití.
/// False, pokud uživatelské jméno již existuje v databázi.
/// </returns>
1 usage
public static bool IsUserameFreeToUse(string username){
    using (var context = new AppDbContext())
    {
        bool isUsernameFree = !context.Users.Any(u:Users => u.Username == username);
        return isUsernameFree;
    }
}
```

Obrázek 5:Metoda bool IsUsernameFreeToUse (Autor, 2024)

```
/// <summary>
/// Kontrola dostupnosti e-mailové adresy.
/// </summary>
/// <param name="mail">E-mailová adresa k ověření.</param>
/// <returns>
/// True, pokud je e-mailová adresa dostupná k použití.
/// False, pokud e-mailová adresa již existuje v databázi.
/// </returns>
1 usage
public static bool IsEmailFreeToUse(string mail){
    using (var context = new AppDbContext())
    {
        bool isEmailFree = !context.Users.Any(u:Users => u.Email == mail);
        return isEmailFree;
    }
}
```

Obrázek 6: Metoda bool IsEmailFreeToUse (Autor, 2024)

5.2.5 Metoda int ChangePassword

Funkce změny hesla hraje v systému důležitou roli z hlediska zabezpečení a poskytuje uživatelům možnost pravidelně aktualizovat svá hesla. Existuje několik důvodů, proč je tato funkcionalita důležitá, a to především kvůli bezpečnosti, prevenci proti opakovanému použití hesel a reakce na bezpečnostní hrozby.

Metoda ChangePassword slouží v zásuvném modulu k provedení změny uživatelského hesla. Přijímá několik vstupních parametrů včetně původního hesla, dvě nová hesla, uživatelské jméno a instanci databázového kontextu.

Metoda nejprve zkontroluje, zda jsou nová hesla shodná a splňují bezpečnostní pravidla. Tato kontrola probíhá pomocí již zmíněné metody ValidateNewPasswordByRules. Pokud kontrola proběhne v pořádku, nové heslo se zahashuje pomocí kryptografické hashovací funkce a ověří se, zda se původní staré heslo, které uživatel taktéž zadává, shoduje s uloženým starým heslem v databázi. Dále se kontroluje shoda nového požadovaného hesla s heslem původním.

Pokud všechny kontroly projdou úspěšně, provede se změna hesla v databázi pomocí aktualizace hodnoty HashedPassword uživatele a uložení změn do databáze.

```
/// <summary>
/// Metoda pro změnu uživatelského hesla.
/// </summary>
/// <param name="oldPassword">Původní heslo uživatele pro ověření identity.</param>
/// <param name="newPassword1">Nové heslo, které bude nastaveno.</param>
/// <param name="newPassword2">Potvrzení nového hesla pro kontrolu shody.</param>
/// <param name="username">Uživatelské jméno uživatele, jehož heslo se mění.</param>
/// <param name="context">Instance <see cref="AppDbContext"/> pro přístup k databázi.</param>
/// <returns>
/// 0, pokud změna hesla proběhla úspěšně.
/// -1, pokud nová hesla nejsou shodná.
/// -2, pokud nové heslo má nedostatečnou délku.
/// -3, pokud nové heslo nesplňuje bezpečnostní pravidla.
/// -4, pokud původní heslo nesouhlasí s uloženým heslem uživatele.
/// -5, pokud nové heslo je stejné jako původní heslo.
/// </returns>
public int ChangePassword(string oldPassword, string newPassword1,
    string newPassword2, string username,AppDbContext context) {
    if (!(newPassword1 == newPassword2)) {
        return -1; //NOT_MATCHING
    }
    int evaluateNewPass = ValidateNewPasswordByRules(newPassword1);
    if (evaluateNewPass == -1)
    {
        return -2; //BAD_LENGTH
    }else if (evaluateNewPass == -2) {
        return -3;//MISMATCH_RULES
    }
    string hashNewPassword = Users.HashPassword(newPassword1);
    var user:Users? = context.Users.FirstOrDefault(u:Users => u.Username == username); // Checking based on email
    if (Users.HashPassword(oldPassword) != user.HashedPassword) {
        return -4; //MISMATCH_OLD_PASSWORD
    }
    if (!(hashNewPassword == user.HashedPassword)) {
        return -5; //SAME_PASSWORD
    }
    user.HashedPassword = hashNewPassword;
    context.SaveChanges();
    return 0; //OK
}
```

Obrázek 7: Metoda int ChangePassword (Autor, 2024)

5.3 Třída RegisterViaMail.cs

V dnešním digitálním světě je proces registrace klíčovým prvkem pro správné fungování online komunit, aplikací a webových platforem. Umožňuje uživatelům vytvářet své individuální účty, získávat personalizovaný přístup a využívat nabízené funkce. Tato kapitola se zaměří na implementaci třídy RegisterViaMail.cs, která představuje svůj mechanismus registrace uživatelů prostřednictvím e-mailu.

5.3.1 Personalizovaný přístup

Registrace poskytuje uživatelům možnost vytvořit si své vlastní účty, což otevírá dveře k personalizovanému přístupu ke službám a obsahu. Personalizované účty umožňují uživatelům nastavovat preference, ukládat data a interagovat s platformou tak, jak jim nejlépe vyhovuje.

5.3.2 Budoucnost registrace

S ohledem na neustálý vývoj digitálních technologií a uživatelských preferencí je plánováno do budoucna rozšířit možnosti registrace. Je zvažována implementace přihlášení prostřednictvím sociálních sítí, jako jsou Facebook nebo Google a využití příslušných API. Tímto způsobem lze poskytnout uživatelům ještě pohodlnější a rychlejší způsob vstupu do naší platformy.

Dalším stupněm ochrany, který je zvažován, je přidání dvoufaktorové autentizace. Dvoufaktorová autentizace představuje další vrstvu zabezpečení, která vyžaduje kromě běžného hesla i další ověřovací prvek, například kód z mobilní aplikace nebo textovou zprávu. Tato metoda poskytuje efektivní obranu proti neoprávněným přístupům a zvyšuje celkovou bezpečnost platformy. Před prvním vydáním projektu se na implementaci dvoufaktorové autentizace, jako součásti rozšíření bezpečnostních opatření, bude pracovat.

5.3.3 Metoda int validateAndCreateNewUser

Tato metoda slouží k validaci a vytvoření nového uživatelského účtu v systému. Ze všeho nejdříve probíhá kontrola dostupnosti žádaného uživatelského jména. Pokud je již používáno, metoda vrátí hodnotu -1, což signalizuje, že takový uživatel již existuje.

Následuje kontrola shody mezi zadanými hesly (password1 a password2). Pokud nejsou identická, metoda vrátí hodnotu -2. Dále provádí validaci hesla dle stanovených pravidel, jako je délka a formát. V případě nesouladu s pravidly vrátí odpovídající hodnoty -3 (špatná délka), nebo -4 (nesoulad s pravidly).

Další krok zahrnuje kontrolu formátu zadaného e-mailu. Pokud neodpovídá očekávanému formátu, metoda vrátí hodnotu -5. Následuje kontrola dostupnosti

daného e-mailu. Pokud je e-mail již používán jiným uživatelem, dostaneme od metody návratovou hodnotu -6.

Poslední dvě kontroly se týkají telefonního čísla s jeho předvolbou. Probíhá zde kontrola dostupnosti, pokud je číslo s danou předvolbou již registrováno v systému, metoda vrátí -7.

Pokud celá validace až do tohoto bodu proběhla bez sebemenší chyby, zásuvný modul se pokusí vykomunikovat s databází vytvoření nového uživatelského účtu s příslušnými zvalidovanými atributy včetně hashování hesla a pokusí se jej uložit do databáze pomocí Entity Framework Core a SQLite.

V případě neúspěchu při práci s databází se vyvolá výjimka, která bude předána na front end a v programové části vrátí návratovou hodnotu -8. V opačném případě nového, úspěšně zaregistrovaného uživatele vrátí hodnotu 0.

```
/// <summary>
/// Ověří, validuje a vytvoří nového uživatele na základě poskytnutých informací.
/// </summary>
/// <param name="username">Uživatelské jméno nového uživatele.</param>
/// <param name="password1">První zadávané heslo.</param>
/// <param name="password2">Druhé zadávané heslo pro potvrzení.</param>
/// <param name="firstName">Křestní jméno nového uživatele.</param>
/// <param name="lastName">Příjmení nového uživatele.</param>
/// <param name="mail">E-mailová adresa nového uživatele.</param>
/// <param name="birthDate">Datum narození nového uživatele.</param>
/// <param name="dialingCode">Telefonní předvolba nového uživatele.</param>
/// <param name="phoneNumber">Telefonní číslo nového uživatele.</param>
/// <param name="favouriteInstruments">Pole oblíbených hudebních nástrojů nového uživatele.</param>
/// <returns>
/// 0, pokud je registrace úspěšná.
/// -1, pokud je zadané uživatelské jméno již obsazené.
/// -2, pokud se zadaná hesla neshodují.
/// -3, pokud je délka hesla mimo povolený rozsah.
/// -4, pokud heslo neodpovídá požadovanému formátu.
/// -5, pokud je e-mailová adresa ve špatném formátu.
/// -6, pokud je zadaný e-mail již obsazený.
/// -7, pokud je zadané telefonní číslo již obsazené.
/// -8, pokud dojde k chybě při vytváření nového uživatele.
/// </returns>
```

Obrázek 8: XML komentáře k metodě int ValidateAndCreateNewUser (Autor, 2024)

```
public int validateAndCreateNewUser(string username,string password1, string password2, string firstName,
    string lastName,string mail,DateTime birthDate,int dialingCode, int phoneNumber,
    string [] favouriteInstruments)
{
    if (!InputValidators.IsUserameFreeToUse(username)) { return -1;} //CHECK FOR USAGE
    if (!password1.Equals(password2)) { return -2; } //CHECK IF PASS ARE IDENTICAL
    int validate = new InputValidators().ValidateNewPasswordByRules(password1); //OUTPUT of validator
    if (validate == -1) { return -3; //BAD_LENGTH
    } else if (validate == -2){ return -4; //FORMAT_MISMATCH
    }
    if (new InputValidators().ValidateMail(mail) == false) { return -5;} //FORMAT CHECK
    if (!InputValidators.IsEmailFreeToUse(mail)) { return -6; } //CHECK FOR USAGE
    if (!InputValidators.IsNumberFreeToUse(dialingCode, phoneNumber)) { return - 7;}
    try {
        using (var context = new AppDbContext()) {
            var newUser = new Users{
                Username = username,
                HashedPassword = Users.HashPassword(password1),
                FirstName = firstName,
                LastName = lastName,
                Email = mail,
                BirthDate = birthDate,
                RegistrationDate = DateTime.Now,
                LastActiveDate = DateTime.Now,
                UserRating = 0.0,
                DialingCode = dialingCode,
                PhoneNumber = phoneNumber,
                FavoriteInstruments = favouriteInstruments.ToList(),
                BoughtContent = [],
                SellingContent = [], };
            context.Users.Add(newUser);  // ADD NEW USER
            context.SaveChanges();
        }
    }
    catch (Exception ex) {
        return -8; //execption with DB
    }
    return 0; //ALL OK
}
```

Obrázek 9: Metoda int ValidateAndCreateNewUser (Autor, 2024)

5.4 Třída Users.cs

Třída Users představuje entitu uživatele systému. Její struktura reflektuje klíčové informace o uživateli. Jedná se o třídu, která vytváří instanci objektu User.

5.4.1 Konstruktory

V projektu jsou používané tři definice konstruktorů, přičemž první z nich je konstruktor parametrický, který přijímá všechny potřebné parametry pro vytvoření instance třídy Users. Druhý bezparametrický slouží k zajištění existence konstruktoru

bez parametrů a je využíván například při vytvoření nového uživatele, protože v databázi nejsou všechny atributy povinné a některé ani nově registrovaný uživatel nemůže mít, nebo je nemusí vyplňovat. Třetí definicí je konstruktor parametrický, takový, který je vyvolán po úspěšném přihlášení se do systému a je očištěn o hashované heslo, které kvůli bezpečnosti po dobu relace není nikde v této konkrétní

```
/// <summary>
/// Inicializuje novou instanci třídy Users s kompletním souborem atributů.
/// </summary>
/// <param name="userId">Unikátní identifikátor uživatele.</param>
/// <param name="username">Uživatelské jméno.</param>
/// <param name="hashedPassword">Hashované heslo.</param>
/// <param name="firstName">Křestní jméno.</param>
/// <param name="lastName">Příjmení.</param>
/// <param name="email">E-mailová adresa.</param>
/// <param name="birthDate">Datum narození.</param>
/// <param name="registrationDate">Datum registrace.</param>
/// <param name="lastActiveDate">Datum poslední aktivity.</param>
/// <param name="userRating">Hodnocení uživatele.</param>
/// <param name="dialingCode">Telefonní předvolba.</param>
/// <param name="phoneNumber">Telefonní číslo.</param>
/// <param name="favoriteInstruments">Seznam oblíbených hudebních nástrojů.</param>
/// <param name="boughtContent">Seznam zakoupeného obsahu.</param>
/// <param name="sellingContent">Seznam prodávaného obsahu.</param>
public Users(int userId,string username, string hashedPassword, string firstName, string lastName,
    string email,DateTime birthDate,DateTime registrationDate, DateTime lastActiveDate, double userRating,
    int dialingCode, int phoneNumber,List<string> favoriteInstruments , List<int> boughtContent,List<int> sellingContent)
{
    UserId = userId;
    Username = username;
    HashedPassword = HashPassword(hashedPassword);
    FirstName = firstName;
    LastName = lastName;
    Email = email;
    BirthDate = birthDate;
    RegistrationDate = registrationDate;
    LastActiveDate = lastActiveDate;
    UserRating = userRating;
    DialingCode = dialingCode;
    PhoneNumber = phoneNumber;
    FavoriteInstruments = favoriteInstruments;
    BoughtContent = boughtContent;
    SellingContent = sellingContent;
}
```

Obrázek 10: První konstruktor (Autor, 2024)

instanci specifikováno.

```
/// <summary>
/// Inicializuje novou instanci třídy Users bez parametrů.
/// </summary>
1 usage
public Users()
{
    throw new NotImplementedException();
}
```

Obrázek 11: Druhý konstruktor (Autor, 2024)

```
/// <summary>
/// Inicializuje novou instanci třídy Users s omezeným souborem atributů.
/// </summary>
/// <param name="userId">Unikátní identifikátor uživatele.</param>
/// <param name="username">Uživatelské jméno.</param>
/// <param name="firstName">Křestní jméno.</param>
/// <param name="lastName">Příjmení.</param>
/// <param name="email">E-mailová adresa.</param>
/// <param name="birthDate">Datum narození.</param>
/// <param name="registrationDate">Datum registrace.</param>
/// <param name="lastActiveDate">Datum poslední aktivity.</param>
/// <param name="userRating">Hodnocení uživatele.</param>
/// <param name="dialingCode">Telefonní předvolba.</param>
/// <param name="phoneNumber">Telefonní číslo.</param>
/// <param name="favoriteInstruments">Seznam oblíbených hudebních nástrojů.</param>
/// <param name="boughtContent">Seznam zakoupeného obsahu.</param>
/// <param name="sellingContent">Seznam prodávaného obsahu.</param>
1 usage
public Users(int userId, string username, string firstName, string lastName, string email, DateTime birthDate,
    DateTime registrationDate, DateTime lastActiveDate, double userRating, int dialingCode, int phoneNumber,
    List<string> favoriteInstruments, List<int> boughtContent, List<int> sellingContent)
{
    UserId = userId;
    Username = username;
    FirstName = firstName;
    LastName = lastName;
    Email = email;
    BirthDate = birthDate;
    RegistrationDate = registrationDate;
    LastActiveDate = lastActiveDate;
    UserRating = userRating;
    DialingCode = dialingCode;
    PhoneNumber = phoneNumber;
    FavoriteInstruments = favoriteInstruments;
    BoughtContent = boughtContent;
    SellingContent = sellingContent;
}
```

Obrázek 12: Třetí konstruktor (Autor, 2024)

5.4.2 Popis vlastností

Vlastnosti v této třídě jsou propojeny s databází pomocí SQLite a Entity Framework pomocí anotací. Tato struktura zajišťuje efektivní ukládání a načítání dat z databáze. Nyní budou popsány jednotlivé vlastnosti a jejich význam.

1. UserId – Identifikátor
    a. [Key] – Tato anotace označuje UserId jako primární klíč v databázi, což znamená, že je unikátní identifikátor pro každého uživatele.
    b. [DatabaseGenerated(DatabaseGeneratedOption.Identity)] – specifikuje, že hodnota UserId bude automaticky generována databází při vytváření nového záznamu.
    c. [Column("UserID")] – určuje název sloupce v databázové tabulce, který odpovídá této vlastnosti.
2. Username, HashedPassword, FirstName, …
    a. Každá z těchto vlastností je opět označena anotací [Column], která specifikuje odpovídající název sloupce v databázové tabulce.
    b. Tyto vlastnosti obsahují informace o uživateli.
3. FavouriteInstruments, BoughtContent, SellingContent
    a. Tyto vlastnosti jsou specifické tím, že obsahují kolekce dat.

5.4.3 Metoda HashPassword

Metoda HashPassword je implementací zabezpečeného mechanismu pro hashe hesla uživatele. Metoda využívá SHA-256 algoritmus, což je kryptografická hashovací funkce k převedení uživatelského hesla na bezpečný a jedinečný řetězec.

Při navrhování systému byla zvolena tato kryptografická funkce z několika důvodů. Prvním z nich byla bezpečnost, jelikož SHA-256 je navržena tak, aby byla odolná vůči různým útokům. Proces hashování je jednostranný, což znamená, že není možné matematicky spočítat původní heslo z jeho hashové hodnoty. Díky správně nastavené délce hesla lze tvrdit, že je i odolné vůči rainbow tabulkám pro již provařené, spočítané hashe. V neposlední řadě SHA-256 nabízí dostatečnou efektivitu a rychlost pro běžné použití, což minimalizuje zpoždění při ověřování uživatelských hesel a jejich šifrování.

Samotná metoda dostane v parametru surové heslo v podobě řetězce a postupuje následujícím způsobem:

1. Vytvoření instance SHA-256 – neboli inicializace nové instance, což je kryptografický objekt pro provedení hashování.
2. Převod hesla na Byte Array – konvertuje vstupní heslo z řetězce do pole bytů ve formátu UTF-8.
3. Výpočet hashe – provede hashování převedeného hesla a vrátí pole bytů obsahující hash hodnotu.
4. Konverze zpět na řetězec – převede pole bytů zpět na řetězec a odstraní pomlčky a převede všechny znaky na malá písmena.
5. Návratové hashové hodnoty – návratová hodnota metody je výsledný hash řetězec, který může být bezpečně uložen nebo porovnán s již existujícím hashem při ověření hesla.

```
/// <summary>
/// Získá hash hesla pomocí kryptografické funkce SHA-256.
/// </summary>
/// <param name="password">Heslo, které má být zahashováno.</param>
/// <returns>Hashované heslo.</returns>
5 usages
public static string HashPassword(string password)
{
    using (SHA256 sha256 = SHA256.Create())
    {
        byte[] passwordBytes = Encoding.UTF8.GetBytes(password); // Conversion

        byte[] hashBytes = sha256.ComputeHash(passwordBytes); // calculating HASH

        return BitConverter.ToString(hashBytes).Replace( oldValue: "-", newValue: "").ToLower();// conversion to string
    }
}
```

Obrázek 13: Metoda string HashPassword (Autor, 2024)

5.5 Třída LoginViaMail.cs

V digitálním světě, kde kybernetická bezpečnost a ochrana uživatelských účtů před neoprávněným přístupem hrají klíčovou roli, je třeba implementovat bezpečné a spolehlivé mechanismy pro ověření uživatelů. Třída LoginViaMail.cs představuje součást zásuvného modulu, která se zaměřuje na ověřování uživatelů prostřednictví e-mailu a hesla. Tato kapitola se detailně zabývá strukturou, účelem a implementací této třídy, která má na starost proces přihlášení uživatelů.

5.5.1 Metoda AuthenticateUser

Tato metoda slouží k ověření uživatele na základě poskytnutého e-mailu a hesla. Přijímá taktéž instanci AppDbContext pro přístup k databázi. Před porovnáním hesel se vyvolá metoda HashPassword ze třídy Users, která zajistí, že porovnání hesel bude komunikováno ve stejném formátu. Tento postup zvyšuje bezpečnost uživatelských účtů tím, že uchovává pouze hash hesla v databázi, což ztěžuje potenciálním útočníkům získání původního hesla. Její návratové hodnoty se liší výstupem podmínek, které jsou předem nadefinovány. Uživatel je buď úspěšně přihlášen, nenalezen v databázi, či při poskytnutí špatných přihlašovacích údajů zamítnut v autentizačním procesu.

```
/// <summary>
/// Ověří uživatele na základě e-mailu a hashovaného hesla.
/// </summary>
/// <param name="context">Instance <see cref="AppDbContext"/> pro přístup k databázi.</param>
/// <param name="email">E-mail uživatele, který se snaží přihlásit.</param>
/// <param name="password">heslo uživatele, které je následně ve funkci zahashované, aby se mohly porovnávat jen hashe.</param>
/// <param name="loggedUser">Výstupní parametr obsahující instanci uživatele.</param>
/// <returns>
/// 0, pokud je uživatel úspěšně přihlášen.
/// -1, pokud uživatel není nalezen v databázi.
/// 1, pokud jsou poskytnuty špatné přihlašovací údaje (e-mail neexistuje nebo heslo nesouhlasí).
/// </returns>
public static int AuthenticateUser(AppDbContext context, string email, string password, out Users loggedUser)
{
    string hashedPassword = Users.HashPassword(password);
    loggedUser = null;
    var user Users? = context.Users.FirstOrDefault(u Users => u.Email == email); // Checking based on email
    if (user == null) {
        return -1; //NOT_FOUND
    }
    if (user.HashedPassword == hashedPassword) { // PASS_CHECK
        loggedUser = new Users(user.UserId,
            user.Username,
            user.FirstName,
            user.LastName,
            user.Email,
            user.BirthDate,
            user.RegistrationDate,
            user.LastActiveDate,
            user.UserRating,
            user.DialingCode,
            user.PhoneNumber,
            user.FavoriteInstruments,
            user.BoughtContent,
            user.SellingContent);
        return 0; //USER_LOGGED
    }
    return 1; //BAD_CREDENTIALS (= EMAIL EXISTS ; BAD_PASSWORD)
}
```

Obrázek 14: Metoda int AuthenticateUser (Autor, 2024)

5.6 Třída AppDbContext.cs

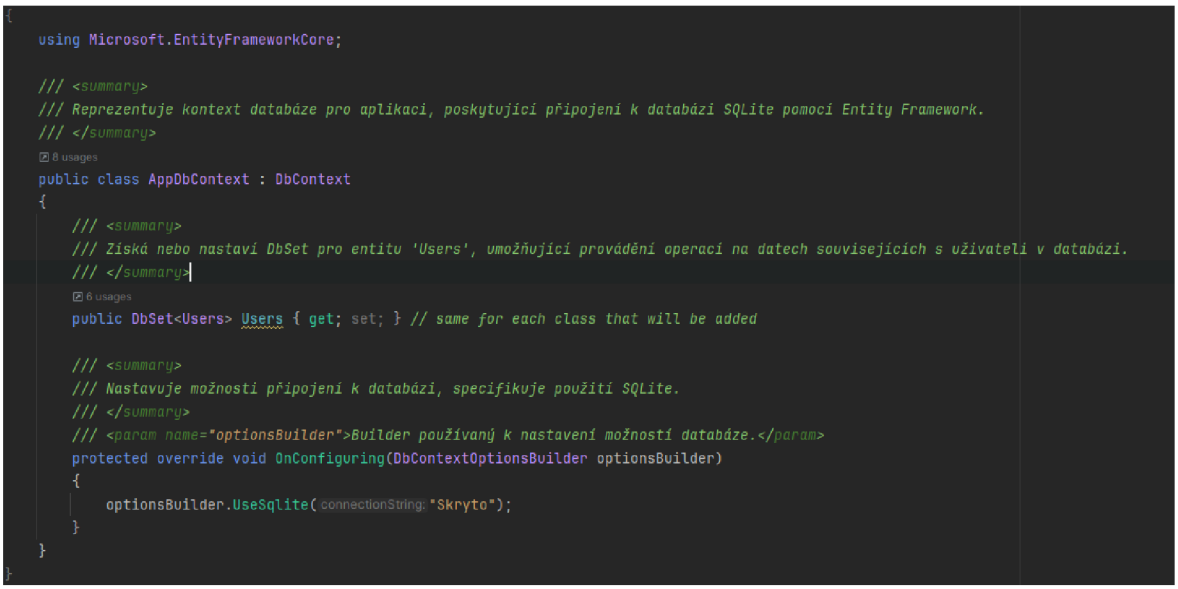
K zajištění spolehlivé a efektivní správě dat je důležité mít solidní a bezpečný přístup k databázi. Tato třída představuje součást zásuvného modulu, která zajišťuje komunikaci s databází a využívá k tomu technologie Entity Framework (EF) a databázový systém SQLite. Tato kapitola se zabývá strukturou této třídy, jejím účelem a způsobem, jakým propojuje modul s databází.

5.6.1 Klíčové prvky třídy

Tato třída vytváří prostředí, ve kterém mohou ostatní části aplikace efektivně interagovat s databází, přičemž zároveň poskytuje abstrakci, která umožňuje snadné přizpůsobení konkrétním databázovým systémům nebo změnám v modelu dat.

Využívání kombinace SQLite a Entity Framework přináší výhody efektivní správy dat a snadné rozšíření, které je pro moderní aplikace s nároky na spolehlivost a flexibilitu.

1. Definice DbSet – Vlastnost Users v typu DbSet<User> definuje sadu operací, které lze provádět s objekty třídy Users v rámci databáze. Podobné vlastnosti mohou být přidány pro další třídy, které budou reprezentovat různé entity v databázi.
2. Konfigurace připojení k databázi – Metoda OnConfiguring slouží k nastavení způsobu, jakým se aplikace připojuje k databázi. V tomto případě je použito SQLite, který je lehkým, přenositelným a efektivním relačním databázovým systémem. Z důvodu bezpečnosti je řádek, na kterém je definováno napojení na databázi, přepsán na hodnotu skryto, jelikož by umožňoval komukoliv zjistit, na jaké adrese se server nachází.



```
using Microsoft.EntityFrameworkCore;

/// <summary>
/// Reprezentuje kontext databáze pro aplikaci, poskytující připojení k databázi SQLite pomocí Entity Framework.
/// </summary>
8 usages
public class AppDbContext : DbContext
{
    /// <summary>
    /// Získá nebo nastaví DbSet pro entitu 'Users', umožňující provádění operací na datech souvisejících s uživateli v databázi.
    /// </summary>
    6 usages
    public DbSet<Users> Users { get; set; } // same for each class that will be added

    /// <summary>
    /// Nastavuje možnosti připojení k databázi, specifikuje použití SQLite.
    /// </summary>
    /// <param name="optionsBuilder">Builder používaný k nastavení možností databáze.</param>
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlite( connectionString: "Skryto");
    }
}
```

Obrázek 15: Třída AppDbContext.cs (Autor, 2024)

5.7 Komentáře v kódu

Komentáře v kódu při vývoji softwaru a jejich správné použití přináší několik výhod. XML komentáře jsou v tomto projektu využívány jak u metod, tak u tříd. Pomáhají při automatickém generování dokumentace a poskytují užitečné informace pro vývojáře i uživatele kódu.

Jednořádkové komentáře jsou rovněž důležité pro lepší čitelnost kódu a rychlejší orientaci ve funkcionalitě. Nadále poskytují vysvětlení účelu nebo principu daného řádku kódu. Tímto způsobem zvyšují srozumitelnost a usnadňují údržbu kódu v průběhu času.

Použití komentářů je klíčové z hlediska týmové spolupráce, protože umožňují rychlejší porozumění kódu i pro nové vývojáře nebo členy týmu. Vývojáři mohou snáze pochopit účel funkcí, parametrů a návratových typů a hodnot.

Celkově je důležité přikládat pozornost ke správnému a smysluplnému komentování kódu, a to jak v jednořádkové formě pro rychlý přehled, tak za pomoci strukturovaných XML komentářů.

5.8 Dokumentace zásuvného modulu

Tato část popisuje proces vytvoření dokumentace pro tento projekt za pomocí nástroje Doxygen.

5.8.1 Doxygen

Doxygen je nástroj pro generování dokumentace z komentářů ve zdrojovém kódu. Podporuje několik formátů dokumentace včetně HTML. Výhodou Doxygenu je možnost automatického generování dokumentace ze zdrojových kódů a umožnění snadné navigace skrze vytvořené odkazy. (Doxygen.nl, 2024)

5.8.2 Postup generování dokumentace

1. Instalace Doxygenu.
2. Konfigurace Doxygenu – specifikace vstupního, výstupního souboru, tříd, které mají být zahrnuty.
3. Editace konfiguračního souboru – po zvážení všech hledisek, bylo rozhodnuto o volbě výstupu ve formátu HTML, které vytvoří internetovou stránku, kterou se lze proklikat do všech bodů dokumentace.
4. Vygenerování dokumentace.
5. Otevření dokumentace – v přílohách naleznete vygenerovanou dokumentaci ve formátu HTML. Otevřením souboru index.html zajistí přechod na kořenovou stránku dokumentace.

6 Výsledky

6.1 Testování

V této části budou popsány metody a postupy, které byly použity pro testování zásuvného modulu a jeho funkcionalit. Testování hraje klíčovou roli v zajištění kvality softwarového produktu a ověření jeho správného fungování.

6.1.1 Přístup k testování

Pro testování byl zvolen přístup založený na použití předdefinovaných assertů a jednotkových testů. Tento přístup umožňuje systematicky ověřovat jednotlivé části kódu a zajišťuje, že nově přidané funkce nebo změny neovlivní existující funkcionality.

6.1.2 Testovací prostředí

Pro testování bylo vytvořeno speciální testovací prostředí, které obsahovalo provizorní databázi. Tato databáze byla použita k simulaci reálných scénářů a ověření správné integrace modulu s daty.

6.1.3 Metody testování

V rámci testování byly použity různé metody, včetně:

- Jednotkové testy – Testy, které ověřují správnost jednotkových metod a funkcí v izolaci od zbytku kódu.
- Integrační testy – Testy, které ověřují, jak se jednotlivé části systému integrují a spolupracují mezi sebou.
- Testy komunikace s databází – Testy, které ověřují správnost implementace SQLite a EF v kódu a jejich dopad na databázi.

6.1.4 Výsledky testování

Výsledky testování potvrdily správnou funkčnost zásuvného modulu a jeho schopnost plnit stanovené požadavky a specifikace. Identifikované chyby byly

opraveny a ověřeny opětovným testováním. Testování zásuvného modulu hrálo klíčovou roli v procesu vývoje a zajištění jeho kvality. Díky systematickému přístupu k testování a použití různých metod byla ověřena správnost a spolehlivost modulu, což přispívá k jeho úspěšnému nasazení a používání uživateli.

Testování také poskytlo vhled do budoucího vývoje modulu včetně návrhů na vylepšení a plánování rozšíření.

6.2 Porovnání s jinými zásuvnými moduly

Zásuvný modul, který byl vytvořen v rámci bakalářské práce, se odlišuje od mnoha existujících zásuvných modulů především v přístupu a implementaci. Na rozdíl od některých modulů, které poskytují specifická API a hotová řešení pro konkrétní účely, tento nabízí přístup k jednoduchému kódu, který lze snadno začlenit do existujících projektů bez nutnosti navazování na externí rozhraní.

Absence specifického API znamená, že tento modul nenabízí předdefinované rozhraní, které by uživatelé museli implementovat ve svých projektech, místo toho poskytuje kód, který může být snadno přizpůsoben a integrován podle potřeb konkrétního projektu.

Důležitým aspektem používání zásuvného modulu je důkladné prostudování přiložené dokumentace, která obsahuje nezbytné informace o implementaci a používání. Oproti modulům s vlastním API, kde je rozhraní předem definováno, tato dokumentace slouží jako klíčový průvodce pro správnou integraci do projektu.

Všechny implementace a funkcionality zásuvného modulu byly pečlivě otestovány, aby byla zajištěna spolehlivost. Zadavatel má možnost stáhnout si tento modul přímo ze soukromého repozitáře na platformě GitHub a snadno jej implementovat do svého projektu.

6.3 Budoucnost zásuvného modulu

Tento zásuvný modul představuje prvotní návrh, který lze v budoucnu rozšířit o řadu funkcionalit a vylepšení. V dalším vývoji modulu je plánováno postupné doplnění o další třídy, které budou reflektovat potřeby a rozšíření databáze. Přidání

nových tříd povede k zajištění komplexnější správy uživatelských dat a zvýší flexibilitu modulu.

Jedním z klíčových rozšíření bude implementace funkcionality pro změnu e-mailu uživatele. To umožní uživatelům aktualizovat své kontaktní údaje a větší svobodu, přičemž budou zachovány bezpečnostní standardy.

Dalším nedílným krokem bude přidání komunikace s mailovým serverem v souvislosti se změnami osobních a citlivých údajů, kterými jsou například e-mail nebo heslo. Tím bude zajištěna bezpečná a ověřená cesta pro uživatele, kteří chtějí upravit svá osobní data.

Dlouhodobým cílem vývoje je implementace dvoufaktorové autentizace, která poskytne vrstvu dalšího zabezpečení prostřednictvím mobilního ověření. Tato funkce přidá další vrstvu bezpečnosti k již existujícím postupům ověření.

V rámci rozšíření funkcí, se předpokládá také přidání možnosti odstranění a dočasného zmražení uživatelského účtu. Tato funkcionalita umožní uživatelům snadno spravovat svůj účet a přizpůsobovat svoji přítomnost v systému podle vlastních potřeb.

V další fázi rozvoje modulu se plánuje přidání možnosti nákupu kreditů, což otevře cestu k implementaci platebních funkcí a poskytne uživatelům další možnosti využití služeb.

Vývoj modulu bude flexibilně reflektovat a reagovat na potřeby uživatelů této komunity. Modul se bude neustále vyvíjet a inovovat, aby poskytoval optimální zážitek a splňoval nejnovější bezpečnostní standardy.

7 Závěr

Tato bakalářská práce měla za cíl vytvořit prvotní verzi zásuvného modulu pro uživatele a sepsat k němu dokumentaci, a to v rámci projektu Metronom4u.cz.

V teoretické části byly vymezeny klíčové teoretické pojmy a standardní postupy spojené s vývojem zásuvných modulů. Důraz byl kladen na pochopení výhod, designových vzorů a historie, což poskytuje solidní základ pro tvorbu zásuvných modulů. Programovací jazyk C# byl zvolen pro své výhody v oblasti objektového programovaní, jednotného typového systému a stabilní podpory v rámci platformy .NET. V kontextu .NET bylo zdůrazněno využití NuGet, který je správcem balíčků pro .NET platformu a umožňuje snadné přidávání externích knihoven a nástrojů, což zrychluje proces vývoje a zajišťuje aktuálnost používaných komponent. Entity Framework byl představen jako klíčový nástroj pro mapování databáze, což usnadňuje práci s relačními daty. V kombinaci se SQLite, které bylo zvoleno pro jednoduchost a přenositelnost, tak tvoří spolehlivý základ pro ukládání uživatelských dat. Další součástí teoretické části bylo seznámení s verzovacím systémem GitHub, který poskytuje prostředí pro sdílení, revize a správu kódu.

Praktická část úspěšně implementovala nabyté teoretické poznatky při vytvoření zásuvného modulu uživatele. Bylo vytvořeno pět tříd, které pokrývají všechny základní funkce, které měly být vyvinuty. Byla provedena analýza uživatelských cílů dvou rovin uživatelských přístupů – tvůrce obsahu a předplatitel obsahu. Díky této analýze byly navrhnuty funkce a třídy tak, aby reflektovaly potřeby obou skupin a budou sloužit jako podklad do budoucna při rozšiřování a dalším vývoji tohoto modulu. Vytvořena byla také dokumentace, která usnadní implementaci zadavateli, se kterým bude následně pokračovat vývoj jak zásuvného modulu, tak i celé webové aplikace.

Mezi důležité body budoucího vývoje po společné diskuzi se zadavatelem lze zařadit postupné doplnění dalších tříd, které budou reflektovat rozšíření databáze, které povede ke zvýšení komplexnosti a konzistence budoucnosti zásuvného modulu. Klíčovým faktorem bude také rozšíření bezpečnostní vrstvy o dvoufaktorovou autentizaci a přidání komunikace s mailovým serverem, díky kterému bude možné změnu citlivých údajů provádět s větší mírou bezpečnosti z hlediska

potvrzovacího e-mailu. Dále je nutné tento modul rozšířit o funkce, jako jsou dočasné zmražení či smazání uživatelského účtu, aby uživatelé měli možnost přizpůsobovat svoji přítomnost v systému dle svých potřeb. Implementace platebních funkcí poskytne uživatelům další možnosti využití služeb v rámci nákupu kreditů a odemykání obsahu. Modul bude i nadále flexibilně reflektovat a reagovat na potřeby komunity uživatelů a jejich zpětnou vazbu.

Tato bakalářská práce může pomoci zadavateli s ulehčením vývoje jeho projektu z hlediska uživatele a může se tak věnovat větší pozornost jiným částem projektu.

8 Seznam použitých zdrojů

1. ALLEN, Grant a Michael OWENS. The definitive guide to SQLite. 2nd ed. [New York]: Apress, c2010. Expert's voice in open source (Apress). ISBN 978-1-4302-3225-4.
2. An introduction to NuGet. Microsoft [online]. 2022 [cit. 2024-02-27]. Dostupné z: https://learn.microsoft.com/en-us/nuget/what-is-nuget
3. CORONEL, Carlos a Steven MORRIS. Database systems: design, implementation, and management. 13e. Australia: Cengage, [2019]. ISBN 978-1-337-62790-0.
4. Design Patterns Explained – Dependency Injection with Code Examples. , Thorben. Stackify [online]. 2023 [cit. 2024-02-23]. Dostupné z: https://stackify.com/dependency-injection/
5. Designing Reusable Classes. Laputan [online]. 1988 [cit. 2024-02-24]. Dostupné z: http://www.laputan.org/drc/drc.html
6. Doxygen [online]. 1997 [cit. 2024-03-08]. Dostupné z: https://www.doxygen.nl/
7. GUTHALS, Sarah a Phil HAACK. GitHub for dummies. John Wiley, 2019. ISBN 978-1-119-57265-7.
8. HEJLSBERG, Anders, Mads TORGERSEN, Scott WILTAMUTH a Peter GOLDE. C# Programming Language. Fourth Edition. Addison-Wesley Professional, [2010]. ISBN 978-0-321-74176-9.
9. CHATLEY, Robert, Susan EISENBACH a Jeff MAGEE. Modelling a Framework for Plugins [online]. 180 Queen's Gate, London SW7 2AZ, 2003 [cit. 2024-03-09]. Dostupné z: https://www.researchgate.net/publication/2882633_Modelling_a_Framework_for_Plugins. Vědecký článek. Imperial College London.
10. SMITH, Jon P. Entity Framework Core in action. Second edition. Shelter Island: Manning, [2021]. ISBN 978-161-7298-363.
11. STERNE, Jonathan. Plug-in. Britannica [online]. 2022 [cit. 2024-02-23]. Dostupné z: https://www.britannica.com/technology/plug-in
12. SWEET, Richard. The Mesa programming environment. ACM SIGPLAN Notices [online]. 1985, 1985(20), 14 [cit. 2024-02-24]. Dostupné z: doi:https://doi.org/10.1145/17919.806843

13. Úvod do NET. Microsoft [online]. 2024 [cit. 2024-02-22]. Dostupné z: https://learn.microsoft.com/cs-cz/dotnet/core/introduction?WT.mc_id=dotnet-35129-website

9 Seznam obrázků, tabulek, grafů a zkratek

9.1 Seznam obrázků

9.2 Seznam tabulek

Seznam použitých zkratek

EF – Entity Framework
ORM – objektově-relační mapovač
IoC – Inversion of Control
.NET - Network Enabled Technologies

Přílohy

Příloha 1 – složka se zdrojovým kódem

Příloha 2 – složka s dokumentací