VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Webová platforma pro ověření znalostí účastníků kurzu

Web Platform for Verifying Knowledge of Training Course Participants

Bc. Adam Šárek

Semestrální projekt

Vedoucí práce: Ing. Jakub Štolfa, Ph.D.

Ostrava, 2022

Abstrakt

Cílem této práce je analyzovat stávajících řešení webových aplikací pro poskytování vzdělávacích kurzů. Práce se zaměřuje na vývoj webové platformy nazvané TaoEnhancer, která slouží jako rozšíření pro vybrané řešení TAO Core. Vyvíjená aplikace doplňuje stávající řešení o některé očekávané a užitečné funkce, které jsou u jiných platforem běžné a ve vybrané aplikaci chybí. Informační systém je navržen v jazyce C#, který dále doplňuje framework ASP.NET Core využívající architekturu MVC. Jednotlivé role v systému zajišťuje přihlašování přes Google. Výsledkem práce je funkční řešení, které učitelům umožňuje správu bodování testů a individuální ohodnocení studentů. Studenti mají možnost zobrazit výsledek vykonaných testů až na úroveň jednotlivých otázek. Součástí práce je kromě popisu implementace daného řešení také návod k použití a výsledky testování.

Klíčová slova

Vzdělávací platforma, test, C#, TAO

Abstract

The goal of this work is to analyse existing web application solutions for providing training courses. The work focuses on the development of a web platform called TaoEnhancer, which serves as an extension for a selected solution TAO Core. The developed application complements the existing solution with some expected and useful functions that are common on other platforms and are missing in the selected application. The information system is designed in C#, which further complements the ASP.NET Core framework using the MVC architecture. Individual roles in the system are provided by the Google sign-in. The result of the work is a functional solution that allows teachers to manage test points and individual student evaluation. Students have the opportunity to display the result of their solved tests up to the level of the individual questions. In addition to the description of the implementation of the solution, the work also includes user guide and testing results.

Keywords

Educational platform, test, C#, TAO

Obsah

Seznam použitých zkratek a symbolů

HTML	–	Hyper Text Markup Language
N/A	–	Not Available
PHP	–	PHP: Hypertext Preprocessor
XML	–	Extensible Markup Language
RDF	–	Resource Description Framework
TXT	–	Textový soubor
TAO	–	Test Assisté par Ordinateur
XAMPP	–	X – meziplatformní, A – Apache, M – MariaDB, P – PHP, P – Perl
IoT	–	Internet of Things
ID	–	Identification
SW	–	Software
UC	–	Use Case
SQL	–	Structured Query Language
MVC	–	M – Model, V – View (pohled), C – Controller (řadič)
2FA	–	Dvoufaktorové ověření
API	–	Application Programming Interface
URL	–	Uniform Resource Locator
URI	–	Uniform Resource Identifier

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Motivací práce bylo získat náhled nad aktuální situací v oblasti vzdělávacích platforem. Úvodní kapitola se zaměřuje na analýzu několika již existujících řešení včetně jednotlivých prvků a funkcionalit, které tyto řešení běžně používají. Tento průzkum je důležitý především proto, jelikož je potřeba vědět, jakým směrem by práce měla směřovat, aby obsahovala všechny důležité funkce, které uživatel od podobné aplikace běžně očekává. V rámci této kapitoly byly také představeny některé konkrétní frameworky, které by mohly být potenciálními kandidáty na možné vylepšení.

Pro další rozšíření byl vybrán nástroj TAO Core, jehož hlavním nedostatkem je, že nenabízí možnost zobrazení výsledku vyplněného testu, a také u něj schází komplexnější nastavení bodů za jednotlivé otázky testu. Chybí také možnost přidělení bodů studentovi např. za otevřenou odpověď, kterou není možné bez ruční kontroly ohodnotit. Kolega Bc. Josef Micak tento nástroj včetně jeho nedostatků blíže popisuje ve svém textu práce, jelikož má v plánu jej dále rozvíjet v rámci své diplomové práce. V jeho textu je možné nalézt rovněž návod, jak s tímto nástrojem pracovat či jak je možné jej využít společně s vyvíjeným řešením.

Další kapitola se zaměřuje na jednotlivé požadavky uživatelů vyvíjeného systému, jelikož každý uživatel od tohoto systému očekává něco jiného. Případy užití poté popisují jednotlivé role a jejich přístup k daným funkcím systému, a také konkrétní scénář pro vykonání uvedené akce.

Následující kapitola se poté již podrobně zaměřuje na samotnou implementaci jednotlivých částí vyvíjené aplikace TaoEnhancer. Nejprve je zmíněné přihlašování přes Google a jeho implementace pomocí Google API. Dále pak správa uživatelů a jejich uživatelské role využívané při autorizaci stránek. Součástí implementace je také popis použití MVC architektury, zajištění responzivního designu a podpory tmavého režimu. Implementace dále pojednává o práci se soubory a podpoře pro testování.

Součástí práce je také kapitola, která se zaměřuje na testování vyvíjené aplikace. Testování komponent kontroluje, zda jsou vybrané funkce správně ošetřeny proti možným chybám v systému a zda se chovají korektně. Automatizované testování rozhraní poté zkoumá, zda systém funguje správně i z venku, tedy z pohledu uživatelského rozhraní.

Předposlední kapitola obsahuje technologie použité při vývoji, testování a nasazení dané aplikace. Vzhledem k tomu, že byla aplikace vyvíjena v programovacím jazyce C#, tak nedílnou součástí použitých technologií byla platforma .NET, která umožňuje vývoj právě v tomto jazyce. ASP.NET Core byl poté využit především díky úzké integraci s vývojovým prostředím Visual Studio, které bylo díky své robustnosti při vývoji daného řešení stěžejní. Veškeré změny v kódu je možné zpětně dohledat díky využití služby GitHub, což při vývoji práce zefektivnilo a zrychlilo dohledávání chyb mezi jednotlivými verzemi aplikace.

Kapitola 2

State of the art

Testovací platformy slouží k vytváření formulářů či průzkumů, jejichž účelem je především zhodnotit znalosti účastníků určitého kurzu či studia. Rovněž mohou poskytovat důležitou zpětnou vazbu pro lektora z hlediska efektivity dané výuky. V této kapitole je čtenář seznámen se současným state of the art – s v dnešní době nejpoužívanějšími nástroji nebo technikami používanými v této oblasti.

2.1 Stávající řešení

2.1.1 Google Forms

Google Forms [1] je jednoduchý nástroj pro tvorbu dotazníků. Umožňuje velmi rychle vytvářet formuláře propojené s účtem Google na jejichž tvorbě se může podílet i více uživatelů. Pro svou jednoduchost je tato platforma mnohdy využívána nejen studenty, ale i většími společnostmi. Dotazníky je možné propojit s dalšími aplikacemi Google, jako jsou Drive, Calendar, Docs, Sheets či Gmail.

2.1.2 ProProfs Survey Maker

ProProfs [2] umožňuje vytvářet dotazníky, formuláře a kvízy, které lze využít např. pro kontaktní formulář, zpětnou vazbu či žádost o práci. Sdílení probíhá buď skrze sociální sítě, nebo pomocí odkazu či vložení přímo na cílovou webovou stránku. Stránka rovněž nabízí více než 100 různých šablon, systém vyhodnocení testů a umožňuje otázky doplnit o obrázky či videa. Nechybí ani zabezpečení pomocí hesla.

2.1.3 Quiz Maker

Quiz Maker [3] slouží k vytváření kvízů, osobnostních testů, hlasování či dotazníků. Obsahuje více než 40 typů otázek, které mohou mít více možných typů odpovědí. Stránka poskytuje také automatické vyhodnocování testů certifikáty dle dosaženého skóre. Ve verzi premium či platinum je na tvorbě testů dokonce možné spolupracovat ve více lidech. Služba se odlišuje mimo jiné tím, že je přístup k jednotlivým kvízům možné zpoplatnit.

2.1.4 ClassMarker

ClassMarker [4] nabízí profesionální tvorbu testů, které jsou určené pro online testování v obchodním či vzdělávacím procesu. Po vyplnění testu jsou jeho výsledky automaticky vyhodnoceny a je možné je ihned zobrazit. Otázky je možné vygenerovat v náhodném pořadí a uživatel mezi nimi může během testu libovolně přecházet. Test je možné během vyplňování přerušit a dokončit později. Přístup může být omezen heslem, poplatkem či nedostatečným počtem zbývajících pokusů. Celkově se tedy jedná o velmi propracované řešení.

2.2 Prvky testovacích platforem

2.2.1 Formulářové prvky

Zatímco některé otázky mohou mít pouze jednu odpověď, tak nad jinými je vhodnější se podrobněji rozepsat či vybrat více možných odpovědí. Od toho se tedy odvíjí požadavek na možnost přidání více formulářových prvků.

Mezi obvyklé formulářové prvky patří zaškrtávací políčko, přepínač, číselník či pole se seznamem. Využití ovšem může najít také prvek nahrání souboru pro případné připojení externího dokumentu či výběr data pro otázky, které s tímto údajem pracují. Samozřejmostí je také tlačítko, které slouží pro přesouvání mezi jednotlivými otázkami a také samotné odeslání testu.

2.2.2 Doplňující prvky

Průběh testu je vhodné určitým způsobem znázornit, aby měl uživatel přehled, v jaké části se v daném okamžiku nachází. Výhodou může být také doplnění o možnost vrácení se na určitou otázku, což by ovšem mělo být možno zakázat tvůrcem testu z důvodu, kdy tato možnost není pro daný test žádoucí.

Jednotlivé testové otázky mohou být obohaceny o doplňující obrázek či video, které mnohdy lépe nastíní daný problém než samotný text zadání.

2.3 Frameworky pro vývoj platformy

Frameworků umožňujících vývoj testovací platformy existuje několik. Obvykle každý takovýto framework nabízí možnost přidávat více různých typů odpovědí, nabízí správu testů, otázek či uživatelů či poskytuje výsledky z již vyplněných testů. Čím se však odlišují, je způsob jakým ukládají data a jak je k těmto datům v rámci vývoje možné přistupovat, což má zásadní roli při výběru vhodného frameworku.

2.3.1 Požadované vlastnosti frameworku

2.3.1.1 Přístup k datům

Pro práci v týmu je důležité, aby měli všichni účastníci přístup k aktuálním datům na platformě. To je možné zajistit tím, že jsou data uložena na internetu odkud jsou pak dále načítána jednotlivými klientskými zařízeními spolupracovníků.

Data by měla být uložena nejlépe v databázi, ke které je možné přistupovat z libovolného typu zařízení. Důležité je pak, aby data byla vhodně zařazena do odpovídajících tabulek a aby neobsahovala zbytečně redundantní informace, aby byla zajištěna rozumná velikost a také plynulost běhu samotné platformy.

Některé platformy svá data místo do databáze ukládají do souborů, avšak tato varianta není příliš ideální. Práce se soubory je obvykle pomalejší než s databází a je také složitější zachovat určitou strukturu a přehlednost dat. Nezanedbatelná je také kompatibilita při vývoji v různých operačních systémech či aplikacích.

2.3.2 Příklady frameworků

2.3.2.1 TaoTesting

TaoTesting je open source framework, který poskytuje základní prvky určené pro tvorbu testovací platformy. Umožňuje přidávat více typů odpovědí a např. přidělovat body jednotlivým otázkám. Výhodou tohoto software je jednoduchá a přehledná administrace. Nevýhodou je pak to, že pro jeho fungování je potřeba instalovat spustitelnou aplikaci, která využívá prostředky serveru a také že své data ukládá do souborů, takže není možné k nim externě přistupovat.

2.3.2.2 Moodle

Moodle [5] je open source systém umožňující nasazení a vývoj testovací platformy. Poskytuje velké množství detailních nastavení pro jednotlivé kurzy, testy či otázky a také správu studentů, studentských skupin a další. Výhodou Moodle je, že není nutné instalovat žádné aplikace a je možné jej rovnou nahrát přímo na webový server. Moodle všechny data ukládá do databáze, takže je možné k nim vzdáleně přistupovat odkudkoliv, což umožňuje jejich zapojení do vývojového procesu.

2.3.3 Výběr frameworku

Z výše zmíněných příkladů frameworků byl nakonec zvolen framework TaoTesting. I přestože tento framework není z hlediska ukládání do souboru ideální pro vývoj v týmu, tak oproti Moodle je u něj stále prostor pro možná vylepšení.

2.4 Uživatelé testovacích platforem

Každá testovací platforma obsahuje určité rozdělení na základě přihlášeného uživatele. Lze obvykle předpokládat, že po přihlášení pod určitým studentským účtem, se zobrazí informace relevantní pro daný účet. Naopak to, že by měl daný student přístup k zadání testů či jejich bodování je nejen neočekávané, ale přímo nežádoucí. Je třeba tedy identifikovat určité uživatelské role, podle kterých jsou jednotlivé stránky a funkcionality systému rozděleny. Jak je toto řešeno již přímo v rámci implementace vyvíjené platformy je možné vidět v sekci 4.1.

2.4.1 Uživatelské role testovacích platforem

Testovací platformy obvykle poskytují několik základních uživatelských rolí. Pro uživatele, který pracuje se zadáním kvízů, dotazníků či jakýchkoliv variant testů je určena role učitele či lektora. Tato role obvykle umožňuje spravovat zadání, bodování a další specifikace testů. Cílem této role je vytvořit test, který má vyzkoušet znalosti účastníka nějakého vzdělávacího kurzu.

Účastník kurzu neboli student by měl mít možnost vyřešit přidělené testy a následně zobrazit svůj výsledek a bodové ohodnocení. Cílem studenta je přehledné zobrazení dosavadních výsledků a občasné vypracování testu. Neměl by se tedy zaobírat problémy spojenými s tvorbou testů.

Správce je poté zodpovědný za udržování seznamu jednotlivých uživatelů. Správce by neměl mít přístup k funkcím učitele či studenta, jelikož se nemusí vždy jednat o osobu spojenou se vzděláváním (může jít např. o správce školní sítě). Tato role nemusí být vždy přítomna, např. v situaci, kdy se studenti registrují samostatně, nebo jejich účty spravuje sám učitel.

Kapitola 3

Požadavky, případy užití a architektura

3.1 Požadavky

Před samotnou implementací bylo potřeba analyzovat několik požadavků, které jsou kladeny na vyvíjené řešení. Tyto požadavky se odvíjí od potřeby autorizovaného přístupu na jednotlivé stránky a také rozdělení a správu rolí jednotlivých uživatelů.

ID	Popis	Typ	Druh požadavku	Případ užití	Doména
N1	Uživatelské účty z pohledu systému	N	–	–	SW
N1–P1	Systém umožňuje načítání uživatelských účtů	P	Funkční	–	SW
N1–P2	Systém zamezuje přístup k jednotlivým stránkám pro uživatele s nižší nebo vyšší než požadovanou rolí	P	Funkční	–	SW
N1–P2–I1	Student nemá přístup ke stránkám učitele a správce, učitel nemá přístup ke stránkám studenta a správce	I	–	–	SW
N2	Uživatelské účty z pohledu správce	N	–	–	SW
N2–P1	Systém umožňuje zobrazení všech uživatelských účtu správci	P	Funkční	13	SW
N2–P2	Systém umožňuje přidání studentského uživatelského účtu	P	Funkční	14	SW
N3	Uživatelské účty z pohledu studenta	N	–	–	SW
N3–P1	Systém umožňuje studentovi zobrazit pouze testy, které sám vyřešil	P	Funkční	1	SW
N3–P1–I1	Systém neumožňuje studentovi zobrazit testy vyřešené jinými studenty	I	–	–	SW

Tabulka 3.1: Požadavky pro uživatelské účty

Ve zmíněné tabulce 3.1 je každý požadavek popsán z hlediska různých kritérií. Každý požadavek má přidělené ID, což je identifikační řetězec, pomocí kterého jsme schopni požadavek jednoznačně identifikovat. Dále má požadavek přidělený popis, který slouží k vysvětlení samotné podstaty požadavku. Typ požadavku může být N (nadpis), P (požadavek) a nebo I (informace). Často se v tomto kontextu používají také zkratky anglických výrazů, a to sice H (heading), R (requirement), a I (information). Nadpis můžeme chápat jako kategorizaci požadavků, kde pod jeden nadpis spadá více požadavků. Dalším kritériem je druh požadavku, přičemž rozeznáváme dva druhy požadavků – funkční a nefunkční požadavky. Tabulka dále obsahuje mapování požadavků na případy užití obsažené v diagramu nacházejícím se na obrázku číslo 3.1. Posledním obsaženým kritériem je doména, jejíž hodnota udává, jestli se daný požadavek týká operací prováděných v rámci softwaru (SW) nebo hardwaru (HW).

3.2 Případy užití

Přístup do aplikace je rozdělen pomocí uživatelských rolí – student, učitel a správce. Každá role má přístup k určité sadě stránek a jejich funkcí. Díky tomu se také pro jednotlivé role velmi liší dostupné případy užití, jak je možné vidět na obrázku 3.1.

Student má přístup ke stránkám se seznamem vyřešených testů, prohlížení vyřešeného testu a prohlížení otázek a podotázek vyřešeného testu. Jednotlivé stránky obsahují informace relevantní přímo pro daného studenta, a tedy nemůže zobrazit informace jiného studenta. Student nemá přístup ke stránkám učitele či správce.

Učitel má přístup ke stránkám se seznamem zadání testů, správou zadání testu, správou otázek a podotázek zadání testu. Dále má přístup ke stránkám se seznamem všech vyřešených testů, správou vyřešeného testu a správou otázek a podotázek vyřešeného testu. Jednotlivé stránky obsahují informace ke všem studentům. Učitel nemá přístup ke stránkám studenta či správce.

Správce má přístup ke stránce se seznamem uživatelů, na které může přidávat či odebírat jednotlivé studenty, učitele a správce. Správce nemá přístup ke stránkám studenta či učitele, může si však vytvořit uživatelský účet s rolí učitele a přes něj poté přistupovat k systému. Vytvořit sice může i účet s rolí studenta, nicméně má na výběr pouze z dosud nepřiřazených studentů. Studentský účet však může kdykoliv odebrat a později znovu přidat, takže touto oklikou se může dostat i k jednotlivým stránkám studenta. Případy užití počítají přímo s danými uživatelskými rolemi a jejich pravomocemi, tedy tento způsob jakým může správce mít přístup k více částem systému není součástí jeho případů užití, jelikož by k systému přistupoval z několika různých účtů a také rolí. Sekce 4.1.3 obsahuje více informací o jednotlivých uživatelských rolích a jejich pravomocích.

Ze všech případů užití je vybrán jeden, u kterého je podrobněji rozepsán možný scénář, včetně různých možných situací, které mohou v jednotlivých fázích scénáře nastat.

Obrázek 3.1: Diagram případů užití

Scénář UC14: Přidání studentského uživatelského účtu

1. Správce se přihlásí do systému pomocí svého uživatelského účtu Google.
2. Systém ověří, že daný uživatelský účet v systému existuje.
3. Systém správce přesměruje na stránku správy uživatelů.
4. Systém vypíše seznam uživatelů.
5. Správce vyplní email uživatele.
6. Správce vybere z výběru roli student.
7. Správce vybere z výběru studenta, kterému chce přiřadit tento uživatelský účet.
8. Správce potvrdí zadané údaje stisknutím tlačítka „Přidat“.
9. Systém ověří, že zadané údaje jsou v pořádku.
10. Systém přidá uživatele.
11. Systém zobrazí potvrzovací hlášku: „Uživatel byl úspěšně přidán.“.

Rozšíření scénáře UC14: Přidání studentského uživatelského účtu

2.a) Systém zjistí, že systém momentálně neobsahuje žádné uživatelské účty.

2.a.1) Systém přidá uživatelský účet správce do systému.

2.a.2) Případ užití pokračuje krokem 3.

2.b) Systém zjistí, že daný uživatelský účet v systému neexistuje.

2.b.1) Případ užití pokračuje krokem 1.

6.a) Správce nevybere z výběru roli student, jelikož již neexistuje žádný student, který by ještě neměl uživatelský účet.

6.a.1) Případ užití končí.

7.a) Správce nevybere z výběru studenta, jelikož zbývá pouze jeden student, kterému je možné přiřadit vytvářený uživatelský účet.

7.a.1) Případ užití pokračuje krokem 8.

9.a) Systém zjistí, že zadaný email již v systému existuje.

9.a.1) Systém zobrazí chybovou hlášku: „Tento email již v systému existuje.“.

9.a.2) Případ užití pokračuje krokem 5.

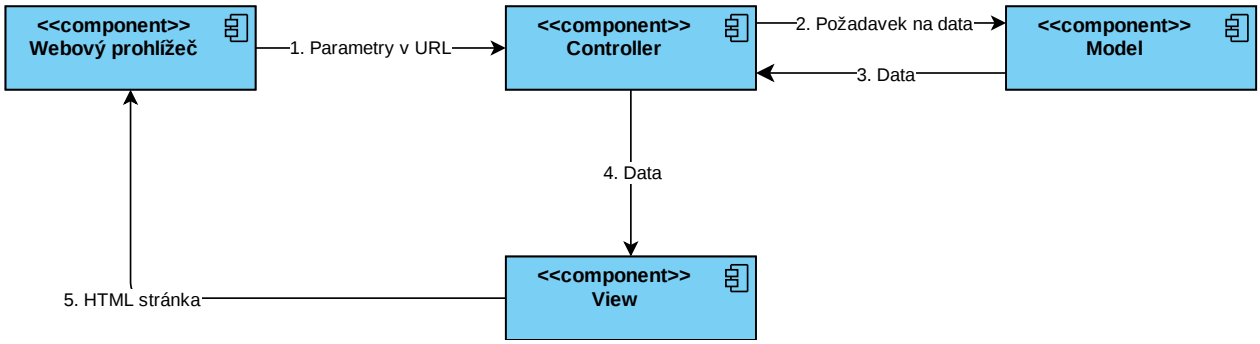
9.b) Systém zjistí, že zadaný email obsahuje nepovolené znaky.

9.b.1) Systém zobrazí chybovou hlášku: „Email obsahuje nepovolené znaky.“.

9.b.2) Případ užití pokračuje krokem 5.

3.3 Architektura

Systém je postaven na architektuře Model-View-Controller zkr. MVC, která interně rozděluje webovou stránku do tří nezávislých komponent Model, View (pohled), Controller (řadič). Toto rozdělení umožňuje rychlejší implementaci změn a strukturalizaci samotného kódu. Cílem je především oddělit logiku od výstupu a tím zjednodušit vývoj jednotlivých částí aplikace.



Obrázek 3.2: Diagram komponent – MVC architektura

Model přijímá požadavek na data. Model data načítá a poté je uchovává, zpracovává a poté vrací do řadiče. Jelikož model nemá přehled o existenci komponent pohledu či řadiče, tak by neměl obsahovat prvky, které jsou závislé na stavu některé z těchto komponent.

Pohled se skládá z HTML stránky obsahující uživatelské rozhraní webové aplikace a udává tak, jak má daná stránka vypadat. HTML stránka je tvořena HTML strukturou, CSS designem a skripty ve skriptovacím jazyce JavaScript. Tato komponenta má k logické vrstvě přístup skrze již zmíněný model, který by měl obsahovat všechny potřebné data bez nutnosti dodatečného přístupu do zbylých vrstev systému.

Řadič je důležitou komponentou MVC architektury, jelikož slouží jako prostředník mezi pohledem a modelem. Na vstupu přijímá uživatelská data, které dále předává příslušnému modelu. Přístup pohledu k modelu a jeho datům dává řadič tím, že jej posílá ve vstupním parametru pohledu. [6]

Kapitola 4

Implementace

4.1 Uživatelské účty

Uživatelské účty umožňují přistupovat k systému a pracovat s ním. Účet zajišťuje to, že uživatel pracuje právě s těmi daty, která mu náleží. Jedná se také o jistou formu zabezpečení, jelikož k systému nemohou přistupovat nepřihlášení uživatelé. Zabezpečení jednotlivých uživatelských účtů potom závisí mimo jiné na bezpečnosti použitého hesla, využití 2FA ověření a na zvolené přihlašovací službě. Některé aplikace používají vlastní přihlášení či používají tlačítka přihlášení pomocí Apple, Microsoft či Google. Právě poslední zmiňovaná varianta je pak implementována ve vyvíjené aplikaci TaoEnhancer.

4.1.1 Přihlašování pomocí Google

Přihlašování pomocí Google bylo zvoleno z důvodu, že se jedná o jednu z nejčastěji využívaných variant externího přihlášení. Google se také důkladně zaměřuje na zabezpečení uživatelských účtů, takže je dnes již každý nově vytvořený účet potřeba spárovat s telefonním číslem či využít nějakou aplikaci pro dvoufaktorové ověřování, např. Google Authenticator či Authy. Toto opatření zvyšuje úroveň zabezpečení účtu tím, že po zadání přístupových údajů je nutné zadat také jednorázový přístupový kód. Tento kód uživatel obdrží buď na telefonní číslo skrze SMS či si jej vygeneruje pomocí spárované aplikace pro dvoufaktorové ověření. Potenciální útočník se tedy k účtu nepřihlásí ani pokud zná přístupové údaje uživatele, jelikož je k přihlášení potřeba také tento přístupový kód.

4.1.1.1 API přihlášení Google

Google poskytuje vývojářům API knihovnu v jazyce JavaScript, kterou mohou využít pro integraci tlačítka přihlášení. Tato knihovna se vkládá do hlavičky HTML dokumentu a pomocí HTML, nebo JavaScriptu je poté možné nastavit dané přihlašovací tlačítko dle potřeby. [7]

Aby mohla API správně fungovat je potřeba jí dodat Googlem vygenerovaný klientský identifikátor a tajný klíč. Tyto dva klíče jsou implementovány ve třídě `Settings`, která slouží jako hlavní místo pro nastavení několika různých součástí aplikace. Tyto dva řetězce lze vygenerovat na webu Google Cloud Platform na stránce API a služby v sekci přihlašovací údaje. Tyto unikátní identifikátory Google používá pro identifikaci klienta, který se snaží používat jeho služby. V nastavení klienta je rovněž nutné nastavit autorizované JavaScriptové zdroje, což je URL vyvíjené aplikace a autorizované URI pro přesměrování, což odkazuje na stránku, kam Google uživatele po přihlášení přesměruje.

4.1.1.2 Přihlášení do TaoEnhancer

Po úspěšném přihlášení Google vrací základní údaje o uživateli, jako je jeho jméno, email, unikátní identifikátor či profilový obrázek. Pro samotnou implementaci TaoEnhancer je důležitý pouze email, který slouží k rozlišení jednotlivých uživatelů a jejich pravomocí.

Pokud je v systému daný email spárován s nějakým uživatelským účtem, který byl předem vytvořen správcem, tak dojde k úspěšnému přihlášení do TaoEnhancer a uživatel je dle své uživatelské role přesměrován na odpovídající stránku.

V případě, že v systému neexistuje uživatelský účet s daným emailem, tak je uživateli zobrazena chybová hláška „Uživatel s tímto emailem v systému neexistuje.“ a uživatel musí požádat správce o vytvoření uživatelského účtu s tímto emailem.

Výjimkou je situace, kdy v systému není žádný uživatel a kdy je naopak přihlášený uživatel přidán do systému automaticky v roli správce. Na to, že v systému dosud nejsou žádní uživatelé je uživatel dopředu upozorněn informační hláškou „Systém dosud neobsahuje žádného uživatele.“. Tato funkce umožňuje aplikaci TaoEnhancer začít používat již po instalaci a rovnou nastavit účet správce. Tento účet samozřejmě lze později odebrat. Odebrání není možné pouze v případě, že v systému není žádný jiný správce. Pokud by byl totiž odebrán poslední správce, tak by se již nikdo nemohl do systému v roli správce přihlásit, jelikož jediný, kdo má možnost přidat správce je právě správce. Samozřejmě je vždy možné přidat správce v situaci, kdy není v systému ani jeden uživatel, nicméně to je po instalaci a následném běžném používání aplikace nepravděpodobné.

Automatická registrace prvního přihlášeného uživatele po instalaci jako správce by mohla být potenciální bezpečnostní hrozbou, pokud by byl systém delší dobu veřejně přístupný, proto je velmi žádoucí, aby se co nejdříve po instalaci tento účet vytvořil.

4.1.2 Správa uživatelských účtů

Správu uživatelských účtů má na starosti správce systému. Správce je po přihlášení do systému automaticky přesměrován na stránku se seznamem uživatelů. Na této stránce má možnost přidat uživatele či některého z existujících uživatelů odebrat.

V případě, že chce správce uživatele přidat, musí nejdříve zadat Google email uživatele a poté vybrat jednu z dostupných uživatelských rolí. Dostupné uživatelské role se liší podle toho, zda jsou již vytvořeny všechny studentské účty. Pokud každý student má již v systému svůj účet, pak role student dostupná není. Role učitele a správce jsou dostupné naopak vždy. Pokud chce správce přidat učitele či správce pak již může formulář odeslat. Při přidávání studenta je však potřeba ještě vybrat odpovídajícího studenta, ke kterému má být daný Google email přiřazen. Správce vybírá z výběru studentů, kteří dosud nemají účet toho, kterého chce přidat. Pokud zbývá poslední student, který ještě nemá účet, pak není možné výběr rozkliknout a automaticky se počítá s tím, že chce správce přidat právě tohoto studenta, pokud tedy vybral v předchozím výběru uživatelskou roli student.

4.1.3 Uživatelské role

V systému figurují 3 uživatelské role – student, učitel a správce. Každá z těchto rolí má přístup na omezený počet stránek, které jsou přístupné přímo pouze pro danou roli. Rozdělení uživatelů do těchto rolí je důležité z toho důvodu, že v oblasti vzdělávacích kurzů obvykle figuruje učitel, který připravuje a spravuje testy a dále také student, který může zobrazit své výsledky, které mu poskytují důležitou zpětnou vazbu ke svému studijnímu výkonu. Všichni uživatelé se pak pro přístup k jednotlivým funkcím systému musí nejprve přihlásit skrze svůj přidělený Google účet. Přihlášení přes Google účet je nutné z toho důvodu, že se jedná o jedinou implementovanou metodu, která zajišťuje zabezpečený přístup do systému.

Student má po přihlášení přístup na stránky, které mu umožňují zobrazit celkový aktuální počet získaných bodů, seznam vyřešených testů a samotné jednotlivé vyřešené testy až na úroveň jednotlivých otázek a podotázek. U daných otázek a podotázek student vidí nejen bodové ohodnocení, ale také očekávanou správnou odpověď a také svou vlastní odpověď. Tyto informace jsou pro studenta důležité, jelikož se může na případné chyby zaměřit při svém dalším vzdělávání.

Učitel má v aplikaci vyhrazené stránky pro zobrazení seznamu zadání jednotlivých testů, u kterých má možnost nastavit body za správnou a špatnou odpověď na jednotlivé podotázky testu. Dále má také přístup na stránku, která obsahuje seznam všech vyřešených testů studenty, kde může upravit jednotlivé bodové ohodnocení konkrétního studenta. Tato možnost je důležitá, jelikož se v testu mohou vyskytovat také volné odpovědi, u kterých není možné zautomatizovat přidělování bodů, jelikož je potřeba manuálně zkontrolovat studentovu odpověď.

Správce má v aplikaci výhradní postavení správce uživatelských účtů. Jedná se o jedinou uživatelskou roli, která má možnost přidávat či odebírat uživatele. Více o správě uživatelů pojednává sekce 4.1.2.

Role správce nebývá vždy součástí testovacích platforem, nicméně ve vyvíjené aplikaci byla implementována z toho důvodu, že je potřeba manuálně přidávat či odebírat spárované Google účty. Tento proces by neměl být náplní práce učitele, takže je vhodné, aby tuto činnost prováděl někdo jiný, tedy např. správce školní sítě. Správci bychom nicméně neměli dávat přístup ke správě bodování, kterou má na starosti učitel, proto má vyhrazenu vlastní uživatelskou roli, která umožňuje pouze správu uživatelů. Role správce má své využití především v hromadném přidávání uživatelů na začátku kurzu či školního roku a následné hromadné odebírání na jeho konci. Potřeba jeho využití mimo dané časové úseky je způsobena především v situaci kdy některý z uživatelů ztratí přístup ke svému Google účtu a je nutné nastavit nový účet Google, který bude s tímto uživatelem spárován.

4.1.3.1 Číselná reprezentace uživatelské role

Jednotlivé uživatelské role mají v systému definovanou určitou číselnou reprezentaci, která umožňuje jednotlivé role navzájem rozlišovat a dále s nimi pracovat např. při autorizaci jednotlivých stránek o čemž pojednává následující sekce 4.1.4. Vzhledem k tomu, že je potřeba počítat také s tím, že je nutné zpřístupnit stránku přihlášení pro dosud nepřihlášeného uživatele, tak je v systému reprezentován také nepřihlášený uživatel. Rozdělení jednotlivých rolí a jejich číselnou reprezentaci znázorňuje tabulka 4.1 níže.

Uživatelská role	**Číselná reprezentace role**
Správce	2
Učitel	1
Student	0
Nepřihlášený uživatel	-1

Tabulka 4.1: Číselná reprezentace uživatelských rolí

4.1.4 Autorizace stránek

Autorizace stránek umožňuje omezit přístup na jednotlivé stránky dle předem zadaných kritérií. Toto ověření probíhá jako první při otevření dané stránky ještě před tím, než se provede cokoliv jiného. Cílem je zpřístupnit určité funkce a stránky pouze uživatelům, kteří by k nim skutečně měli mít přístup.

O autorizaci se stará funkce `CanUserAccessPage`, která kontroluje, zda se číselná reprezentace uživatelské role (viz sekce 4.1.3.1) pohybuje v požadovaném rozmezí. U studentů pak kontroluje také to, zda sedí jeho identifikátor s identifikátorem studenta dané stránky, aby měl student přístup pouze na své stránky a nikoliv na stránky jiného studenta. Výpis 4.1 obsahuje konkrétní implementaci funkce `CanUserAccessPage` ve třídě `HomeController` a výpis 4.2 pak její konkrétní využití pro autorizaci stránky pro správu uživatelů.

```
public bool CanUserAccessPage(string checkStudentIdentifier, int minRequiredRole,
    int maxRequiredRole)
{
    int userRole = GetUserRole();
    if (userRole >= minRequiredRole && userRole <= maxRequiredRole)
    {
        if (userRole == 0)
        {
            string userStudentIdentifier = "";
            try { userStudentIdentifier = studentController.LoadStudentByEmail(
                GetUserLoginEmail()).studentIdentifier; }
            catch { }

            if(checkStudentIdentifier != "" &&
                userStudentIdentifier != checkStudentIdentifier)
            {
                return false;
            }
        }
        return true;
    }
    return false;
}
```

Listing 4.1: Implementace funkce `CanUserAccessPage`

```
public IActionResult ManageUserList(string text = "")
{
    // Check if user can access page
    if (!CanUserAccessPage("", 2, 2)) { return RedirectToAction("Index", "Home",
        new { error = "access_denied" }); }

    ...
}
```

Listing 4.2: Autorizace stránky pro správu uživatelů s využitím funkce `CanUserAccessPage`

4.2 Architektura MVC

Architektura MVC (Model-View-Controller) umožňuje rozdělit implementaci webových stránek do tří nezávislých komponent. Toto rozdělení umožňuje lépe strukturalizovat jednotlivé části kódu a díky tomu také jeho snadnější údržbu a rychlejší implementaci potřebných změn. Cílem je také oddělení logiky od výstupu pro zpřehlednění a rozdělení kódu do jednotlivých nezávislých celků. Souhrn rolí jednotlivých komponent v rámci architektury MVC pojednává sekce 3.3.

Tato architektura byla zvolena z důvodu využití technologie ASP.NET pro tvorbu webových stránek, jelikož tato technologie v rámci nástroje Visual Studio obsahuje již základní architekturu MVC. Tato architektura je vhodná pro vývoj webových stránek v programovacím jazyce C#.

4.2.1 Model

Komponenta model uchovává data, které jsou použity v komponentě pohledu. Vzhledem k tomu, že všechny stránky obsahují nějaký nadpis, umožňují číst uživatelskou roli a název aplikace zůstává neměnný, tak je implementován model `PageModel` (viz výpis 4.3), který tyto informace obsahuje.

```
public class PageModel
{
    private string title;
    private (string message, string messageClass) headerMessageData;
    private int userRole;

    public string Solution { get { return "TaoEnhancer"; } }
    public string Title { get { return title; } set { title = value; } }
    public (string message, string messageClass) HeaderMessageData { get {
        return headerMessageData; } set { headerMessageData = value; } }
    public int UserRole { get { return userRole; } set { userRole = value; } }
}
```

Listing 4.3: Implementace modelu `PageModel`

Většina stránek však používá vlastní model, který dědí právě z modelu `PageModel`, který je pro všechny stránky společný. Výpis 4.4 obsahuje právě jeden takový model, konkrétně se jedná o model úvodní stránky, který obsahuje navíc text zobrazované chybové hlášky, její odpovídající HTML třídu a také URL, na kterou je uživatel po úspěšném přihlášení přesměrován. URL by jako jediná mohla být přímo v HTML dané stránky, jelikož je neměnná pro všechny případy užití úvodní stránky, nicméně díky tomu, že veškeré URL a souborové cesty jsou definovány ve třídě `Settings`, tak je možné tuto cestu spravovat také v této třídě a předávat její hodnotu právě pomocí modelu.

```
public class IndexModel : PageModel
{
  private string text;
  private string textClass;
  private string signInURL;

  public string Text { get { return text; } set { text = value; } }
  public string TextClass { get { return textClass; } set { textClass = value; } }
  public string SignInURL { get { return signInURL; } set { signInURL = value; } }
}
```

Listing 4.4: Implementace modelu `IndexModel`

4.2.2 View (pohled)

Komponenta pohledu se skládá z HTML stránky obsahující uživatelské rozhraní webové aplikace. Daná stránka je tvořena HTML strukturou, CSS designem a skripty v jazyce JavaScript. V rámci architektury MVC poté mimo jiné pracuje s daty z příslušného modelu.

4.2.2.1 HTML struktura

HTML struktura jednotlivých stránek je přibližně stejná a to z důvodu, že jednotná struktura je mnohem jednodušší na údržbu, nastylování pomocí CSS a napojení do JavaScriptu. Nejprve je zavedena hlavní struktura, která je úplně stejná na všech stránkách. Obsah, který se na jednotlivých stránkách liší je do ní poté vložen a je spravován v separátních souborech. Výpis 4.5 obsahuje implementaci hlavní struktury všech HTML stránek aplikace.

Hlavní struktura obsahuje nastavení jazyka (element `<html lang="cs">` nastavuje češtinu) a dále kódování znaků (element `<meta charset="utf-8" />` nastavuje kódování na UTF-8). Následující řádek zajišťuje, že se šířka webu bude řídit šířkou zařízení, což je technika, která se používá pro zajištění responzivního webu. Dále se nastavuje nadpis stránky, který pracuje s daty z modelu `PageModel`. Další řádek slouží k načtení externího fontu Material Icons, což je písmo obsahující sadu ikon navrženou společností Google, která poskytuje toto písmo k volnému použití pod licencí Apache ve verzi 2.0. Z této sady ikon jsou v řešení použity 4 – ikona šipky zpět, odhlášení, tmavého a světlého režimu. Dále je poté načítán soubor CSS obsahující veškeré stylování aplikace. Jako poslední se v hlavičce nachází načítání skriptu, který se mimo jiné stará o dynamiku formulářových prvků některých stránek. V těle hlavní struktury je poté vložení samotného obsahu dané stránky (pomocí ASP.NET metody `@RenderBody`).

```
<!DOCTYPE html>
<html lang="cs">
    <head>
        <meta charset="utf-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <meta name="theme-color" content="#111111" />
        <title>@Model.Title - @Model.Solution</title>
        <link rel="stylesheet" href="~/css/Material-Icons.css" />
        <link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />
        <script src="~/js/site.js" asp-append-version="true"></script>
    </head>
    <body>
        <div class="container">@RenderBody()</div>
    </body>
</html>
```

Listing 4.5: Implementace hlavní struktury všech HTML stránek

Jak již bylo zmíněno do hlavní struktury, která je pro všechny stránky společná, je vždy vložen také specifický obsah dané stránky. Obsah této struktury se sice velmi liší mezi jednotlivými stránkami, co však zůstává stejné je daná struktura. Každá stránka obsahuje hlavičku v horní části webu, kde je uveden nadpis dané stránky, tlačítko pro změnu barevného režimu a poté případně vlevo tlačítko zpět a vpravo tlačítko odhlášení. Zda se některé z těchto tlačítek bude na stránce nacházet či nikoliv záleží na tom, zda to na dané stránce dává smysl. Tlačítko odhlášení je tedy všude kromě úvodní stránky pro přihlášení a tlačítko zpět je všude mimo úvodní stránku a stránku, která se zobrazí po přihlášení.

Hlavní obsah stránky se nachází v elementu `<main class="main content-width">`, který pojímá jednotlivé sekce `<section class="section">` a jejich panely `<article class="article">`. Většina panelů má nějaký stručný nadpis `<h2 class="article-title title">` a také obsah samotného panelu `<div class="article-content">`. Vzhledem k tomu, že aplikace zobrazuje spoustu různých seznamů dat a strukturovaných informací, tak většinou obsahem panelu je také tabulka `<table class="article-table">`, která je pomocí elementů `<tr>`, `<th>` a `<td>` rozdělena na jednotlivé řádky a sloupce.

Výpis 4.6 obsahuje implementaci obsahu úvodní stránky. V HTML struktuře je možné vidět všechny výše uvedené elementy, které navíc doplňuje kód pro zobrazení hlášky a kód zobrazující tlačítko přihlášení Google. Pro zajištění funkčnosti API přihlášení Google je nutné vložit skript: `<script src="https://accounts.google.com/gsi/client" async defer></script>`, který načítá externí knihovnu Google.

```
@using Common
@model ViewLayer.Models.IndexModel

<script src="https://accounts.google.com/gsi/client" async defer></script>

<header class="header">
    ...
    <div class="header-content content-width">
        <div class="header-left">
            <button class="header-button button button-transparent material-icons"
                id="themeBtn" title="Přepnout do světlého režimu"
                onclick="changeTheme(this)">light_mode</button>
        </div>
        <img class="header-center header-logo" title="@Model.Solution"
            src="~/logo.ico">
    </div>
</header>
...
<main class="main max-content-width @mainHeaderMessagePadding">
    <section class="section justify-content-center">
        <article class="article flex">
            <h2 class="article-title title" title="Přihlášení">Přihlášení</h2>
            <div class="article-content">
                <form asp-controller="Account" asp-action="TestingSignIn"
                    method="POST">
                    <table class="article-table">
                        @{
                            if(@Model.Text != "") {
                                string[] textRowSplit = @Model.Text.Split("\n");
                                if(textRowSplit.Length > 0) {
                                    <tr>
                                        <th class="@Model.TextClass"
                                            align="center">@textRowSplit[0]</th>
                                    </tr>
                                    for(int i = 1; i < textRowSplit.Length; i++) {
                                        <tr>
                                            <td class="@Model.TextClass"
```

```
                                    align="center">
                                    @textRowSplit[1]
                                </td>
                            </tr>
                        }
                    }
                }
                ...
            }
            <tr>
                <td align="center">
                    <div id="g_id_onload"
                       data-client_id="@Settings.GoogleSignInClientId"
                       data-context="signin"
                       data-ux_mode="redirect"
                       data-login_uri="@Model.SignInURL"
                       data-auto_prompt="false">
                    </div>
                    <div id="g_id_signin" class="g_id_signin"
                       data-type="standard"
                       data-shape="rectangular"
                       data-theme="filled_blue"
                       data-text="signin_with"
                       data-size="large"
                       data-logo_alignment="left">
                    </div>
                </td>
            </tr>
        </table>
    </form>
</div>
</article>
</section>
</main>
```

Listing 4.6: Implementace obsahu úvodní stránky

4.2.2.2 Responzivní design

Responzivní design je dnes v podstatě standardem vývoje moderních webových stránek. Cílem je správně zobrazit stránku nejen na větších obrazovkách počítačů či tabletů, ale i mobilních zařízeních. Web by tak ideálně měl být schopen pomocí jednoho kódu zobrazit stejný obsah také na menších obrazovkách.

V rámci implementace je responzivita zajištěna několika technikami. První je přizpůsobení šířky webu: `<meta name="viewport" content="width=device-width,initial-scale=1.0" />`. Dále zajištění flexibility obsahu a jednotlivých prvků HTML. Této flexibility lze dosáhnout testováním webu na malé obrazovce a postupnou identifikací prvků, jejichž obsah je useknut obrazovkou zařízení. Prvky, které se na obrazovku „nevlezou“ celé, nejsou flexibilní a je potřeba toto různými HTML a CSS technikami vynutit.

Mezi základní HTML techniky je možné zařadit např. nahrazení textu tlačítek za ikony, které na menším prostoru zachovávají svůj význam, případně omezení délky textového obsahu, který by byl na menším zařízení zbytečně nepřehledný. V CSS je poté možné např. místo zobrazení celého textu nadpisu zobrazit jeho úvodní část, která se na obrazovku „vleze“ a ukončit jej „…“ s tím, že element ohraničující daný text obsahuje atribut `title=""`, který po najetí myší zobrazí celý text. Na mobilním zařízení samozřejmě myš použít nelze, ale jedná se o vhodný způsob, jak zajistit, že tento text je stále dostupný alespoň na menších obrazovkách některých počítačů. Výpis 4.7 obsahuje způsob, jak pomocí CSS zajistit useknutí přesahující části textu a její nahrazení třemi tečkami.

```
white-space: nowrap;
overflow: hidden;
text-overflow: ellipsis;
```

Listing 4.7: CSS způsob nahrazení přesahující části textu třemi tečkami

4.2.2.3 Podpora tmavého režimu

Aplikace podporuje tmavý režim v případě, že je nastaveno preferované barevné schéma v nastavení operačního systému či prohlížeče. Preferované schéma je detekováno pomocí vlastnosti média: `@media (prefers-color-scheme: dark)`. Pro zajištění snadné úpravy jednotlivých částí, které se v CSS opakují nebo jsou měněny právě třeba při změně barevného schéma je vhodné nahradit proměnnými. Jak je možné proměnné definovat je vidět ve výpisech 4.9 a 4.8. Výpis 4.10 pak znázorňuje, jak je možné dané proměnné v CSS využít.

```
:root {
    --color: #FFF;
    --background: #000;
    --header-background: #111E;
    ...
}
```

Listing 4.8: Redefinice proměnných v CSS (tmavý režim)

```
:root.light-theme {
    --color: #000;
    --background: #FFF;
    --header-background: #EEEE;
    ...
}
```

Listing 4.9: Definice proměnných v CSS (světlý režim)

```
html, body {
    color: var(--color);
    background: var(--background);
    ...
}
```

Listing 4.10: Využití proměnných v CSS

4.2.3 Controller (řadič)

Komponenta řadiče slouží jako prostředník mezi komponentou pohledu a modelu. Na vstupu (v parametrech odpovídající metody) přijímá uživatelská data a na výstupu vrací komponentu pohledu obsahující odpovídající model. Výpis 4.11 obsahuje implementaci řadiče pro stránku správy uživatelů. Řadič na vstupu přijímá odpovídající text hlášky (neprázdný pouze při provedení nějaké akce) a provádí úvodní autorizaci stránky, aby se ujistil, že uživatel, který na ní přistupuje je v roli správce. Dále načítá uživatele dle emailu a informací z TAO a seskupuje je dle toho, zda jsou či nejsou spárovaní s účtem Google a v jaké roli případně vystupují. Tato část kódu není součástí tohoto výpisu vzhledem ke snaze zachovat čitelnost daného výpisu. Její implementaci je možné najít v příloze obsahující zdrojové kódy vyvíjeného řešení v souboru *HomeController.cs*. Všechny informace jsou následně vloženy do připraveného modelu `ManageUserListModel`, který je společně s komponentou pohledu vrácen na výstupu dané metody.

```
public IActionResult ManageUserList(string text = "") {
   // Check if user can access page
   if (!CanUserAccessPage("", 2, 2)) { return RedirectToAction("Index", "Home",
      new { error = "access_denied" }); }
   int userRole = GetUserRole();

   ...

   ManageUserListModel model = new ManageUserListModel {
      Title = "Správa uživatelů",
      UserRole = userRole,
      Students = students,
      StudentsByRoles = studentsByRoles,
      StudentsOfTao = studentsOfTao,
      StudentsOfTaoNotPaired = studentsOfTao.Except(studentsOfTaoPaired).ToList(),
      RoleTexts = new string[] { "Student", "Učitel", "Správce" },
      LoginEmail = "",
      Role = "",
      StudentNumberIdentifier = ""
   };
   switch (text) {
      case "user_successfully_added":
         model.Text = "Uživatel byl úspěšně přidán.";
         model.TextClass = "correct";
         break;
      case "user_successfully_deleted":
         model.Text = "Uživatel byl úspěšně odebrán.";
         model.TextClass = "correct";
         break;
      case "last_admin_cannot_be_deleted":
         model.Text = "Poslední správce nemůže být odebrán.";
         model.TextClass = "incorrect";
         break;
   }
   return View(model);
}
```

Listing 4.11: Implementace řadiče pomocí metody `ManageUserList`

4.3 Práce se soubory

4.3.1 Načítání exportovaných dat

Aplikace je postavena na práci s daty ze souborů exportovaných z testovací platformy TAO Core. Tyto data jsou většinou exportována v souborech formátu XML. Jedinou výjimkou jsou data o studentech, které TAO nazývá test-takers, jejichž soubory mají formát RDF. V úvodní fázi implementace byly soubory formátu RDF načítány pomocí .NET knihovny dotNetRDF, která je pro načítání validní volbou. Pro snížení závislosti na externích knihovnách však bylo od této knihovny ustoupeno ke klasickému načítání XML právě díky tomu, že je vnitřní struktura těchto souborů téměř identická s klasickým XML. K načítání všech XML a RDF souborů je využita třída `XmlReader`, která je součástí oboru názvů `System.Xml`. Výpis 4.12 zachycuje implementaci funkce `LoadStudentByNumberIdentifier`, která slouží k načtení informací o studentovi ze souboru z již zmíněného formátu RDF.

```
public (
    string studentNumberIdentifier, string studentIdentifier, string login,
    string firstName, string lastName, string email
) LoadStudentByNumberIdentifier(string studentNumberIdentifier) {
    (
        string studentNumberIdentifier, string studentIdentifier, string login,
        string firstName, string lastName, string email
    ) student = (studentNumberIdentifier, "", "", "", "", "");

    if (Directory.Exists(Settings.GetStudentsPath())) {
        if (System.IO.File.Exists(Settings.GetStudentFilePath(
            studentNumberIdentifier))) {
            XmlReader xmlReader = XmlReader.Create(Settings.GetStudentFilePath(
                studentNumberIdentifier));
            while (xmlReader.Read()) {
                if (xmlReader.Name == "rdf:Description" &&
                    xmlReader.NodeType != XmlNodeType.EndElement) {
                    student.studentIdentifier = xmlReader.GetAttribute(
                        "rdf:about").Split("#")[1];
                }
                if (xmlReader.Name == "ns0:login" &&
                    xmlReader.NodeType != XmlNodeType.EndElement) {
                    student.login = xmlReader.ReadInnerXml();
                }
```

```
                if (xmlReader.Name == "ns0:userFirstName" &&
                    xmlReader.NodeType != XmlNodeType.EndElement) {
                    student.firstName = xmlReader.ReadInnerXml();
                }
                if (xmlReader.Name == "ns0:userLastName" &&
                    xmlReader.NodeType != XmlNodeType.EndElement) {
                    student.lastName = xmlReader.ReadInnerXml();
                }
                if (xmlReader.Name == "ns0:userMail" &&
                    xmlReader.NodeType != XmlNodeType.EndElement) {
                    student.email = xmlReader.ReadInnerXml();
                }
            }
        }
        else { throw Exceptions.StudentFilePathNotFoundException; }
    }
    else { throw Exceptions.StudentsPathNotFoundException; }

    return student;
}
```

Listing 4.12: Implementace metody `LoadStudentByNumberIdentifier`

Metoda `LoadStudentByNumberIdentifier` přijímá na vstupu číselný identifikátor studenta, což je rovněž název RDF souboru, který TAO exportovalo. Dle tohoto identifikátoru se metoda snaží najít daný soubor za předpokladu, že tedy existuje složka, ve které by se tento soubor měl nacházet.

Všechny souborové cesty jsou pro případy načítání a ukládání nastaveny relativně vůči kořenové složce pro exportovaná data a veškeré nastavení cest je řešeno v rámci třídy `Settings`. Toto rozhodnutí umožňuje jednoduše přepsat kořenovou složku exportovaných dat na jednom místě v případě, že by bylo potřeba danou složku přesunout. Jednotné nastavení je také velmi výhodné při vývoji aplikace na více operačních systémech, jelikož Windows a Linux pracují s jinou adresářovou strukturou. Vzhledem k tomu, že se může stát, že některá složka či soubor nemusí existovat či může být některý soubor porušen, tak je nutné s těmito situacemi v programu počítat. Z tohoto důvodu je zavedena třída `Exceptions`, která poskytuje nastavení chybových hlášek pro případ některé z chyb při načítání.

V případě, že je soubor nalezen, tak je postupně načítán studentský identifikátor, login, jméno, příjmení a email. Všechny tyto informace jsou poté vráceny na výstupu metody společně s číselným identifikátorem studenta v datovém typu C# nazvaném Tuple, který umožňuje několik datových typů udržovat v rámci struktury jedné proměnné.

4.3.2 Načítání a ukládání dat TaoEnhancer

Aplikace TaoEnhancer, jak je nazváno vyvíjené řešení, pracuje nejen s vyexportovanými soubory XML či RDF, ale také se svými vlastními daty ve formátu TXT, což je formát, který slouží k ukládání dat ve formě prostého textu. Aplikace s TXT soubory pracuje tak, že jednotlivé větší celky (např. bodování jednotlivých podotázek v testu) odděluje do několika řádků textu a parametry těchto celků dále podrobněji rozděluje pomocí středníku. Nejedná se o jediný možný způsob práce s daty, nicméně tato varianta byla zvolena mým kolegou Bc. Josefem Micakem především z toho důvodu, že je již nutné pracovat s vyexportovanými daty, které mají určitou adresářovou strukturu. Tuto strukturu je tedy možné využít a ukládat data přímo do příslušných adresářů. Řešení poskytuje prostor pro vylepšení implementací relační SQL databáze pro práci nejen s vyexportovanými, ale i aplikačními daty TaoEnhancer.

```
john_1642833661;0
```

Listing 4.13: Data uživatelského účtu studenta (soubor: john@gmail.com.txt)

Výpis 4.13 obsahuje data uživatelského účtu studenta jménem John Smith, který byl vytvořen v rámci testování vyvíjené aplikace. Jak je možné vidět, soubor má v názvu email, který byl k uživatelskému účtu spárován správcem a obsahuje středníkem oddělený číselný identifikátor studenta a číselnou reprezentaci role student. Při přihlašování tedy systém vyhledá dle emailu daný textový soubor a poté načte jeho obsah. V případě, že se jedná o studenta, tak načte mimo jiné také identifikátor, který slouží pro rozlišení jednotlivých studentů mezi sebou.

4.4 Podpora pro testování

Při vývoji a údržbě aplikace se mohou objevit některé nedostatky či chyby, které v dané aplikaci nejsou žádoucí. Může se jednat o chyby spojené s bezpečností či nefunkčností některých částí systému, což znemožňuje její plnohodnotné použití. Tyto chyby je obvykle možné odhalit a eliminovat již během vývoje díky unit testům či automatizovanému testování. Aby tyto testy bylo možné provést, musí být systém na toto testování připraven.

Pro potřeby testování byla ve třídě `Settings` zavedena proměnná `Testing`. Tato booleovská proměnná jednotlivým částem systému říká, zda aktuálně probíhá testování či nikoliv. V nasazeném řešení by měla vždy zůstat ve stavu `false`, jelikož by se v opačném případě mohlo jednat o bezpečnostní riziko. Testování totiž vyžaduje vstup do systému přes zadní vrátka, což je v rámci testování užitečné, jelikož právě potřebujeme přístup ke všem funkcionalitám aplikace. V nasazeném řešení by se však jednalo o přístup nejen pro testery, ale také pro potenciální útočníky, kteří by byli schopni této zranitelnosti využít.

4.4.1 Testovací uživatel

Pro potřeby testování systému z pohledu určitého uživatelského účtu byla ve třídě `Settings` zavedena proměnná `TestingUser`. Tato proměnná uchovává data o testovacím uživateli, který může mít libovolný email a uživatelskou roli, takže je s jeho pomocí možné otestovat úplně všechny stránky a funkce aplikace. Aby byl testovací uživatel aktivní, je potřeba nastavit již zmíněnou proměnnou `Testing` na hodnotu `true`.

V rámci testování vyvíjené aplikace byl testovací uživatel využit v rámci automatizovaného testování, které prochází několik různě autorizovaných stránek aplikace. Vzhledem k tomu, že kromě úvodní stránky každá stránka systému vyžaduje přístup pod nějakou testovací rolí případně identifikátorem studenta, tak je nutné k otestování těchto stránek využít právě testovacího uživatele. Pokud by systém pracoval s vlastním přihlášením, tak by bylo možné upustit od použití tohoto uživatele, nicméně s využitím externího přihlášení přes Google neexistuje způsob, jak aplikaci otestovat, pokud má v systému zůstat autorizace jednotlivých stránek. Autorizace je důležitá ve finální aplikaci, a proto je nutné otestovat také ji samotnou, a proto byl testovací uživatel v systému implementován.

```
public static (string loginEmail, int role) TestingUser {
   get {
      string loginEmail = "";
      int role = -2;
      if(File.Exists(GetTestingDataPath())) {
         if(Testing) {
            using var fileStream = new FileStream(GetTestingDataPath(),
               FileMode.Open, FileAccess.Read);
            try {
               using var streamReader=new StreamReader(fileStream, Encoding.UTF8);
               string[] userData = (streamReader.ReadToEnd()).Split(';');
               loginEmail = userData[0];
               role = int.Parse(userData[1]);
            } catch {
               loginEmail = "";
               role = -2;
            } finally { fileStream.Close(); }
         }
         else { FileIO.Delete(GetTestingDataPath()); }
      }
      return (loginEmail, role);
   }
```

```
    set {
        if(Testing && value.role > -2) {
            if (!File.Exists(GetTestingDataPath())) {
                FileStream f = File.Create(GetTestingDataPath());
                f.Close();
            }
            using var fileStream = new FileStream(GetTestingDataPath(),
                FileMode.Truncate, FileAccess.Write);
            try {
                using var streamWriter = new StreamWriter(fileStream, Encoding.UTF8);
                streamWriter.Write(value.loginEmail + ";" + value.role);
            } catch { } finally { fileStream.Close(); }
        }
        else if (File.Exists(GetTestingDataPath())) {
            FileIO.Delete(GetTestingDataPath());
        }
    }
}
```

Listing 4.14: Implementace testovacího uživatele

Výpis 4.14 obsahuje implementaci testovacího uživatele. Při pokusu nastavit testovacího uživatele je nejprve provedena kontrola, zda je proměnná `Testing` nastavena na `true` a zda je poskytnutá role vyšší než -2, což je hodnota o 1 menší, než je role nepřihlášeného uživatele. V případě, že tato kontrola selže, tak je smazán textový soubor, obsahující informace o testovacím uživateli. V opačném případě jsou naopak do souboru zapsány poskytnuté informace o testovacím uživateli. Výpis 4.15 ukazuje možný obsah souboru držícího data o testovacím uživateli.

Při pokusu získat testovacího uživatele je nejprve provedena kontrola, zda existuje soubor obsahující informace o testovacím uživateli a následně, zda je proměnná `Testing` nastavena na `true`. V případě, že kontrola projde, tak jsou poté následně načteny informace z příslušného souboru a vráceny v Tuple dané proměnné.

```
student@email.com;0
```

Listing 4.15: Data testovacího uživatelského účtu studenta (soubor: Testing.txt)

Kapitola 5

Testování

Aplikace byla v průběhu jejího vývoje několikrát testována a to unitovými a automatizovanými testy. Obecným cílem testování je odhalit potenciální chyby a zkontrolovat, zda má program očekávané chování. Úkolem je projít několik funkcí aplikace a snažit se záměrně pracovat s nesmyslnými či poškozenými daty, které mohou pocházet jak od uživatele, tak od samotného systému, a na které by systém měl být schopen adekvátně reagovat. V případě, že jsou testy správně nastaveny a některý z nich selže, tak je možné předpokládat, že bylo nalezeno nestandardní chování a je potřeba testovanou funkci opravit. Obecně nám testování snižuje potenciální chybovost aplikace, nicméně velmi záleží na kvalitě jednotlivých testů a také na testovaných komponentách.

Testy byly vytvořeny s pomocí .NET frameworku NUnit [8] a v rámci řešení se nachází v projektu `NUnitTests`. Před spuštěním libovolného testu by měla být nejprve změněna proměnná `Testing` ve třídě `Settings` na hodnotu `true`, čímž aplikace dostane signál o tom, že s ní budeme chtít pracovat v testovacím režimu, který umožňuje např. využít testovacího uživatele. Toto nastavení je důležité hlavně pro automatizované testy, které bez tohoto nastavení nefungují korektně.

5.1 Unitové testy

Unitové testy byly vytvořeny za účelem testování tří vybraných funkcí aplikace. Všechny vybrané funkce testují, zda jsou správně načítány jednotlivé data týkající se testů, bodů a otázek. První test zjišťuje, zda při načítání testu nenastala chyba související s neexistující složkou či souborem. Dále pak testuje, zda je u nalezeného XML souboru schopen najít očekávaná data, nebo zda je tento soubor či jeho struktura nějakým způsobem poškozena. Druhý test testuje data jednotlivých otázek testu, tedy nejprve to, zda program správně reaguje na chybějící složky či soubory. Následně kontroluje, zda sedí informace o tom, zda jsou body za otázku přítomny či nikoliv, což je závislé na obsahu načítaného textového souboru. Třetí unitový test se zaměřuje na načítání parametrů testové otázky. Nejprve kontroluje, zda program správně reaguje na neexistují složky či soubory a poté testuje strukturu XML souboru.

Vytvořené testy jsou parametrizované a je u nich zajištěno pokrytí kódu. U třetího testu je pak navíc použito mockování objektů. V rámci řešení se testy nachází v projektu `NUnitTests` ve třídě `UnitTests`. Pro spuštění výše zmíněných unitových testů není potřeba ve třídě `Settings` měnit hodnotu proměnné `Testing`, jelikož unitové testy nepracují s konkrétními uživateli.

5.2 Automatizované testy

Automatizované testy byly vytvořeny za účelem testování tří vybraných funkcí uživatelského rozhraní aplikace. První test probíhá na stránce správy uživatelů a kontroluje, zda se systém chová korektně při vytváření a odebrání jednotlivých uživatelů. Cílem je otestovat reakci na přidávaný email, který je buď duplikátní či obsahuje nepovolené znaky a dále otestovat, zda je možné úspěšně přidat či odebrat uživatele. U odebrání uživatele je testováno to, zda se nejedná o posledního správce, kterého by nemělo být možné odebrat. O tom, proč je toto opatření nutné, pojednává sekce 4.1.1.2. Druhý test kontroluje funkčnost autorizace jednotlivých stránek aplikace. Cílem tohoto testu je zjistit, zda nějaká uživatelská role nemá díky chybnému nastavení přístup na stránku, na kterou by přístup mít neměla. Testováno je celkem pět stránek postupně od stránek dostupných pro roli nepřihlášeného uživatele až po roli správce s tím, že pro studenta je testována stejná stránka, ale s jiným identifikátorem. Důvodem je to, že nechceme, aby nějaký student měl přístup na stránky jiného studenta. Poslední test se zaměřuje na kontrolu formuláře pro změnu studentových bodů za vyřešený test. V případě, že testová otázka nemá určený počet obdržených bodů, pak není možné upravit počet bodů studenta, což vede k chybové hlášce. V případě, že použijeme testovou otázku, která má určený počet bodů, pak u ní můžeme obdržené body měnit. U změny bodů testujeme především neplatný vstup, tedy např. zadání písmen i když jsou očekávány číselné hodnoty a také to, zda správně funguje číselná hranice zadaných bodů. U otázky, která má mít max. 6 bodů je tato hranice -6 až 6 bodů tzn. testujeme desetinné hodnoty okolo těchto dvou hranic, kde je potenciálně nejvyšší šance najít nestandardní reakci aplikace na zadaný vstup.

Vytvořené testy se v rámci řešení nachází v projektu `NUnitTests` ve třídě `AutomatedTests`. Před spuštěním výše uvedených testů je již na rozdíl od unitových testů nutné změnit hodnotu proměnné `Testing` ve třídě `Settings` na `true`, jelikož v těchto testech pracujeme s různými testovacími uživateli, které není bez tohoto nastavení možné používat. Pro spuštění je také potřeba mít stažený WebDriver některého z prohlížečů, ideálně EdgeDriver, což je ovladač prohlížeče Microsoft Edge, který byl při testování použit. Stažený ovladač by měl být vložen do kořenového adresáře řešení, kde s ním automatizované testy počítají. Ve třídě `AutomatedTests` je případně možné manuálně změnit adresář ze kterého se má ovladač použít. Tento ovladač je u automatizovaných testů důležitý proto, jelikož jej používají ke spouštění testů.

Kapitola 6

Použité technologie

6.1 Microsoft .NET

Microsoft .NET [9] je souhrnný název pro skupinu softwarových technologií tvořících platformu, kterou můžeme použít pro vytváření různých druhů aplikací pro různé typy zařízení (webové aplikace, mobilní aplikace, desktopové aplikace, hry, aplikace pro IoT). Platforma je vyvíjena společností Microsoft, využití této platformy není zpoplatněno a samotný zdrojový kód platformy je volně dostupný, tudíž platformu řadíme do kategorie otevřeného softwaru, což je pojem spíše známý pod svým anglickým názvem open source.

Tato platforma umožňuje k vývoji použít různé programovací jazyky, mezi které patří jazyky C#, F# a Visual Basic. V tomto semestrálním projektu byl k vývoji použit programovací jazyk C#, což je jeden z nejznámějších a nejpoužívanějších programovacích jazyků. K vývoji aplikací prostřednictvím této platformy je určen nástroj Visual Studio, pro vývoj tohoto semestrálního projektu byla použita verze Microsoft Visual Studio Community 2022.

I samotná platforma .NET má různé verze. V tomto semestrálním projektu byla použita verze .NET 6, která obsahuji množství nových funkcí zaměřených zejména na zlepšení výkonu aplikací a zjednodušení vývoje daných aplikací. Verze .NET 6 byla vybrána také kvůli dlouhodobé podpoře – vydána byla v listopadu roku 2021, a podpora je zajištěna do listopadu roku 2024.

6.2 ASP.NET Core

ASP.NET Core [10] je webový framework, který slouží k vývoji webových aplikací. Mohli bychom jej také definovat jako soubor knihoven, přičemž tyto knihovny obsahují řešení problémů, se kterými se programátoři při vývoji často potýkají. Jedná se také o následníka technologie ASP.NET, která už v dnešní době není příliš často využívaná.

Stránky vytvořené prostřednictvím této technologie můžeme identifikovat podle koncovky *.cshtml*, mezi hlavní výhody a vylepšení ASP.NET Core v porovnání s ASP.NET patří zlepšení výkonu, umožnění vývoje na platformách Linux a macOS (ASP.NET umožňuje vývoj pouze prostřednictvím operačního systému Windows) a lepší podporu modulární architektury.

6.3 XAMPP

XAMPP je balík služeb vyvíjený společností Apache Friends, který slouží k snadnému vytvoření lokálního webového serveru. Mohli bychom jej charakterizovat podle jeho zkratky, každé písmeno obsažené ve zkratce totiž představuje jednu funkci nebo vlastnost (X – meziplatformní, A – Apache, M – MariaDB, P – PHP, P – Perl).

V této semestrální práci byl XAMPP využit k vytváření a řešení testů v rámci samotné platformy TAO.

6.4 GitHub

GitHub je webová služba umožňující efektivní spolupráci mezi vývojáři softwarového díla. Kdokoli může na tuto službu bezplatně uložit zdrojový kód, který pak mohou ostatní členové vývojářského týmu (popřípadě po povolení kdokoli jiný) číst a upravovat. Ke své činnosti používá verzovací nástroj Git, který slouží k ukládání historie vývoje celého projektu.

V této semestrální práci jsme GitHub pro zmíněné účely aktivně používali. Díky využití této služby měli všichni členové týmu přístup k nejaktuálnější verzi projektu, a také jsme mohli prohlížet starší verze projektu při vzniku nových chyb.

Kapitola 7

Závěr

Cílem této práce bylo zanalyzovat existující řešení testovacích platforem a vybrat jeden nástroj, který by byl následně rozšířen o funkce, které v něm nebyly dostatečně zpracované nebo úplně chyběly. Pro rozšíření byl vybrán nástroj TAO Core, který neměl dostatečně komplexní možnost zadání bodů včetně možnosti nastavení záporných bodů. Práce byla vytvářená jako týmový projekt, a proto v tomto textu práce není podrobný popis bodování, jelikož tuto část implementace řešil pouze kolega Bc. Josef Micak. Jednotlivé cíle vyvíjeného řešení byly v týmu rozděleny již na začátku a jednotlivé změny byly pravidelně konzultovány nejen s vedoucím práce, ale také v rámci týmu. Při vývoji byl využit verzovací nástroj Git, který umožňuje zpětně zobrazit historii provedených změn a také umožňuje přispívat více členům týmu včetně přidání doplňujícího komentáře, který pomáhá změny lépe dohledat.

Tato práce se zaměřuje především na požadavek zajištění autorizovaného přístupu k jednotlivým funkcím aplikace, což je v práci řešeno pomocí přihlášení přes Google. Dále bylo žádoucí v systému aplikovat architekturu, která by rozdělila systém na několik nezávislých komponent, což by aplikaci zpřehlednilo a umožnilo její jednodušší úpravy do budoucna. Z toho důvodu byla vybrána architektura MVC, která toto rozdělení zajišťuje. Vzhledem k tomu, že se jedná o webovou aplikaci, tak bylo cíleno také na podporu více typů zařízení. Toto bylo zajištěno implementací responzivního designu, který umožňuje použití i na mobilních zařízeních. Nechybí ani podpora tmavého režimu. Pro podporu testování byl implementován testovací uživatel, díky kterému je možné při testování nastavit libovolného uživatele a otestovat jeho konkrétní části systému.

Aplikace má i přes implementované funkce stále prostor pro možné rozšíření. Aktuální řešení pracuje s daty v souborech, což by bylo možné zlepšit použitím relačního databázového systému. Pro ještě lepší strukturalizaci kódu by poté bylo vhodné použít třívrstvou architekturu. Dále by bylo vhodné snížit závislost na Google skrze jeho přihlašování a to buď přidáním tlačítka např. Microsoft či Apple, nebo vytvořením vlastního přihlašování. Co se týká použití dat z původního nástroje TAO Core, bylo by vhodné data získávat přímo bez potřeby jejich manuálního exportování, nebo případně převést jeho důležité části do vytvořeného řešení.

Literatura

1. *Formuláře Google* [online] [cit. 2021-10-03]. Dostupné z: `https://forms.google.com/`.
2. *Survey Maker: Online Survey Software for Free* [online] [cit. 2021-10-03]. Dostupné z: `https://www.proprofs.com/survey/`.
3. *Quiz Maker | Make Amazing Online Quizzes in Minutes* [online] [cit. 2021-10-03]. Dostupné z: `https://www.quiz-maker.com/`.
4. *Online Testing Free Quiz Maker Create the Best quizzes* [online] [cit. 2021-10-03]. Dostupné z: `https://www.classmarker.com/`.
5. *About Moodle - MoodleDocs* [online] [cit. 2022-03-20]. Dostupné z: `https://docs.moodle.org/311/en/About_Moodle`.
6. *MVC architektura* [online] [cit. 2022-04-06]. Dostupné z: `https://www.itnetwork.cz/navrh/mvc-architektura-navrhovy-vzor`.
7. *Overview | Sign In With Google | Google Developers* [online] [cit. 2022-04-06]. Dostupné z: `https://developers.google.com/identity/gsi/web/guides/overview`.
8. *NUnit.org* [online] [cit. 2022-03-20]. Dostupné z: `https://nunit.org/`.
9. *What is .NET? An open-source platform.* [Online] [cit. 2022-03-06]. Dostupné z: `https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet`.
10. *Overview of ASP.NET Core | Microsoft Docs* [online] [cit. 2022-03-06]. Dostupné z: `https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-6.0`.