Tvorba studijních materiálů demonstrující způsoby práce s databází ve frameworku .NET

Petr Mařák

Bakalářská práce
2024

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

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

Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Petr Mařák**
Osobní číslo:	**A20411**
Studijní program:	**B0613A140020 Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Tvorba studijních materiálů demonstrující způsoby práce s databází ve frameworku .NET**
Téma práce anglicky:	**Creating Study Material Demonstrating How to Work with Databases in the .NET framework**

Zásady pro vypracování

1. Popište současný stav technologií pro práci s databází ve frameworku .NET.
2. Zaměřte se především na technologii ADO.NET, knihovnu Dapper a Entity Framework Core.
3. Navrhněte zadání úkolů pro studenty zaměřené na práci s databází ve frameworku .NET.
4. Vytvořte navržené úkoly, popište jejich klíčové časti, jejich cíle, obsah, materiály a teorie k vypracování a také vytvořte vzorová řešení a možné kontrolní otázky.
5. Zhodnoťte dosažené výsledky a formulujte možnosti dalšího rozvoje úkolů.

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

Seznam doporučené literatury:

1. .NET documentation: Learn about .NET, an open-source developer platform for building many different types of applications [online]. Redmond, WA, USA: Microsoft Learn, 2022 [cit. 2022-11-27]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/fundamentals/.
2. Entity Framework Core [online]. Redmond, WA, USA: Microsoft Learn, 2021 [cit. 2022-11-27]. Dostupné z: https://learn.microsoft.com/en-us/ef/core/.
3. ADO.NET [online]. Redmond, WA, USA: Microsoft Learn, 2021 [cit. 2022-11-27]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/framework/data/adonet/.
4. Dapper Tutorial: Getting Started with Dapper [online]. ZZZ Projects, 2019 [cit. 2022-11-27]. Dostupné z: https://dapper-tutorial.net/dapper.
5. PRICE, Mark J. C# 10 and .NET 6 – Modern Cross-Platform Development: Build apps, websites, and services with ASP.NET Core 6, Blazor, and EF Core 6 using Visual Studio 2022 and Visual Studio Code. 6th Edition. Birmingham, UK: Packt Publishing, 2021, 826 s. ISBN 978-1801077361.
6. SMITH, Jon P. Entity Framework Core in Action. 2nd edition. Shelter Island, NY, USA: Manning Publications Co., 2021, 624 s. ISBN 978-1617298363.

Vedoucí bakalářské práce: **Ing. Dušan Fojtů, Ph.D.**
Ústav počítačových a komunikačních systémů

Datum zadání bakalářské práce: **5. listopadu 2023**
Termín odevzdání bakalářské práce: **13. května 2024**

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

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

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

Ve Zlíně dne 5. ledna 2024

Prohlašuji, že

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

Prohlašuji,

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

Ve Zlíně, dne 13. 5. 2024

Petr Mařák, v. r.
…….
podpis studenta

ABSTRAKT

Byly vytvořeny podpůrné materiály pro výuku práce s databázemi v .NET aplikacích. Byly popsány relační a nerelační typy databázových systémů, vývojová platforma Microsoft .NET a její součásti, a také možnosti dostupných technologií pro práci s databázemi v .NET. Byla navržena zadání úkolů pro studenty a jejich vzorová řešení. Dále byla pro tyto úkoly vytvořena výuková aplikace, sloužící také jako praktická ukázka implementace vybraných technologií. Aplikace umožňuje správu uživatelů, výukového obsahu, testových otázek a jejich vyhodnocení.

Klíčová slova: Databázové systémy, SQL, .NET, Studijní materiály, Úkoly, Výuková aplikace, Entity Framework Core, ADO.NET, Dapper

ABSTRACT

Supporting materials were created for teaching how to work with databases in .NET applications. Relational and non-relational types of database systems, the Microsoft .NET development platform and its components, as well as the possibilities of available technologies for working with databases in .NET were described. Assignments for students and sample solutions were proposed. Furthermore, a tutorial application was created for these tasks, which also served as a practical demonstration of the implementation of the selected technologies. The application allows the management of users, educational content, test questions and their evaluation.

Keywords: Database Systems, SQL, .NET, Study Materials, Assignments, Educational Application, Entity Framework Core, ADO.NET, Dapper

Rád bych poděkoval panu Ing. Dušanovi Fojtů, Ph.D. za jeho ochotu, rady a možnost pravidelných konzultací při vedení bakalářské práce.

Prohlašuji, že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

Málokterá aplikace se v dnešní době obejde bez možnosti trvalého ukládání dat. Konkrétní implementace řešení tohoto požadavku se může lišit dle rozsahu aplikace nebo platformy, pro kterou byla vytvořena. Jedno mají však společné – neobejdou se bez použití některého databázového systému.

V dnešní době je velký důraz kladen na splnění legislativních požadavků na zabezpečení, ochranu a archivaci dat. V každém kroku vývoje aplikace je důležité zajistit, aby data uživatelů byla chráněna a spravována s ohledem na platné právní předpisy. To zahrnuje dodržování směrnic GDPR, které upravují způsob, jakým jsou osobní údaje zpracovávány a uchovávány. Kromě toho je nutné dbát na bezpečnostní standardy a pravidla pro zálohování dat, aby byla zajištěna jejich dostupnost a integrita v případě napadení či selhání systému. Spolehlivé a efektivní řízení dat je tak klíčovou součástí každého moderního informačního systému.

Problematika využití nejmodernějších technologií je velmi významná. Srovnání mezi tradičními databázovými systémy, jako je například Access, a novými cloudovými nástroji ukazuje mnohé výhody moderních databázových systémů. Tyto novodobé nástroje nabízejí mnohem lepší funkcionalitu, včetně pokročilých možností zabezpečení a archivace dat. Rychlost zápisu a dostupnost dat jsou díky nim také výrazně posíleny, což přináší nejen efektivitu, ale zároveň i vyšší úroveň spolehlivosti pro společnosti a zákazníky.

Tvorba studijních materiálů v online formě je v dnešní době důležitá, připomeňme si třeba zkušenosti s obdobím Covid, kdy drtivá většina pedagogických institucí po celé České republice byla nucena ve velmi krátkém období přejít z prezenční výuky na distanční. Pro instituce, které měly již před touto dobou studijní materiály v online formě, neřkuli celé předměty a studijní programy, byl tento přechod mnohem snazší. A to nejen pro pedagogy, ale i studenty a jejich rodiče. Tento obrat podtrhuje potřebu moderních technologií ve vzdělávání a zdůrazňuje význam kvalitních a dostupných studijních materiálů. Řešení, která umožňují flexibilní přístup ke vzdělávání, jsou tak podstatná pro budoucnost vzdělávacího systému.

I. TEORETICKÁ ČÁST

1 DATABÁZOVÉ SYSTÉMY

Databáze je organizovaná kolekce strukturovaných dat, obvykle uložených elektronicky v počítačovém systému. Databáze je řízena systémem řízení báze dat (DBMS), pomocí kterého mohou k databázi správci a aplikace přistupovat. Společně se označují jako databázový systém. [1]

První počítačem řízené databáze byly vytvořeny na počátku 60. let 20. století Charlesem Bachmanem pod názvem Integrated Data Store. [2]

V současnosti jsou využívány zejména relační databáze a nerelační NoSQL databáze. [1]

	NoSQL (nerelační)	SQL (relační)
NEJVHODNĚJŠÍ PRO:	• Zpracování velkých objemů vzájemně nesouvisejících, neurčitých nebo rychle se měnících dat. • Data nezávislá na schématu nebo schéma určené aplikací. • Aplikace, u kterých je důležitější výkon a dostupnost než silná konzistence. • Neustále přístupné aplikace, které slouží uživatelům po celém světě.	• Zpracování dat, která jsou relační a mají logické a diskrétní požadavky, které je možné určit předem. • Schéma, které je potřeba udržovat a synchronizovat mezi aplikací a databází. • Starší systémy vytvořené pro relační struktury. • Aplikace vyžadující komplexní dotazování nebo transakce s více řádky.
SCÉNÁŘE:	• Mobilní aplikace. • Analýzy v reálném čase. • Správa obsahu. • Přizpůsobení. • Aplikace IoT. • Migrace databází.	• Účetní, finanční a bankovní systémy. • Systémy řízení zásob. • Systémy pro správu transakcí.
ŠKÁLOVÁNÍ:	• Horizontální škálování dat prostřednictvím horizontálního dělení napříč servery.	• Vertikální škálování dat prostřednictvím zvyšování zatížení serveru.
DATOVÝ MODEL:	• Typy databází: databáze párů klíč-hodnota, dokumentů, sloupců a grafové databáze. • Ukládání dat v závislosti na typu databáze.	• Typ databáze: databáze tabulek řádků seskupených do vztahů. • Používání jazyka SQL (Structured Query Language). • Ukládání dat ve formě řádků v tabulkách; odděleně uložená související data, která se v případě složitých dotazů spojují.

Obrázek 1. Porovnání relačních a NoSQL databází [3]

1.1 Relační databáze

Relační databázový model navrhl E. F. Codd v 70. letech 20. století v IBM a dosud patří k těm nejrozšířenějším.

Data v relační databázi jsou uspořádána jako soubor tabulek se sloupci a řádky. Někdy se také používají výrazy relace, atributy a záznamy. Sloupce určují datový typ, a každý řádek představuje hodnoty těchto datových typů – tedy konkrétní záznam v tabulce databáze. Každý sloupec uvnitř tabulky, tak jako i jednotlivé tabulky, musí mít jedinečné pojmenování.

Každá tabulka musí mít alespoň 1 sloupec nastavený jako primární klíč, který představuje jedinečný identifikátor řádku, a každý řádek lze použít k vytvoření vztahu mezi různými tabulkami pomocí cizího klíče. Cizí klíč představuje odkaz na primární klíč jiné tabulky a umožňuje tak související tabulky vzájemně propojit a získat všechna související data.

Díky uspořádání dat do předem definovaných vztahů se můžeme na data dotazovat deklarativně. Deklarativní dotaz je způsob, jak definovat, co chceme ze systému získat, aniž bychom vyjadřovali, jak má systém výsledek vypočítat. To je podstatou relačního systému na rozdíl od jiných systémů. Další výhodou je možnost kdykoliv vkládat nová data a měnit či mazat data stávající, aniž by se měnila samotná struktura databáze.

Pro řízení přístupu, dotazování, manipulaci s databází a definování dat se využívá strukturovaný dotazovací jazyk SQL. [1][4]

Na obrázku (Obr. 2) je ukázka obsahu tabulky s názvem „prihozy“ v relační databázi.

Id	Castka	CreatedOn	UserId	AukceId
1	100.00	2024-01-14 20:54:50.622874	3	1
2	150.00	2024-01-14 20:54:50.622874	1	1
3	100.00	2024-01-14 20:54:50.622874	1	2
4	300.00	2024-01-14 20:54:50.622874	3	2
5	700.00	2024-01-14 20:54:50.622874	1	2
6	500.00	2024-01-14 20:54:50.622874	3	2
7	10.00	2024-01-14 22:14:45.252257	4	4
8	15.00	2024-01-14 23:00:10.440308	4	5
9	25.00	2024-01-14 23:09:18.362509	4	5
10	15.00	2024-01-14 23:15:37.683976	4	6
NULL	NULL	NULL	NULL	NULL

Obrázek 2. Ukázka výpisu relační tabulky

V této ukázce je sloupec Id je nastaven jako primární klíč. Sloupce UserId a AukceId jsou cizí klíče, pomocí kterých je údaj o konkrétním příhozu jednoznačně propojen s konkrétním uživatelem a aukcí, uloženými v jiných tabulkách.

Existuje několik implementací relačního databázového systému od různých organizací.

1.1.1 Dotazovací jazyk SQL

Structure Query Language je široce používaný jazyk pro správu relačních dat a databázových systémů. Na základě teoretické práce E. F. Codda vytvořili v roce 1974 výzkumníci ve společnosti IBM první praktický návrh syntaxe tohoto jazyka a za dalších 5 let byl poprvé použit v produktu pro komerční využití.

Každý databázový systém, jakými jsou například MySQL, Postgre SQL, MS SQL Server nebo SQLite, implementují vlastní variantu jazyka SQL, jejíž syntaxe se může vyjma základních vlastností lišit, jelikož ANSI standard nespecifikuje všechny aspekty SQL nutné pro implementaci databázového systému.

Zde jsou příklady některých SQL příkazů a jejich syntaxe. U SQL příkazů nezáleží na velikosti písmen.

- Vytvoření nové databáze s názvem „company“:
  CREATE DATABASE company;
- Vytvoření nové tabulky „employees“:
  CREATE TABLE employees (id int, first_name varchar(64), last_name varchar(128), hire_date date, salary numeric);
- Vložení nových dat do tabulky „employees“:
  INSERT INTO employees (first_name, last_name, hire_date, salary) VALUES ('Samuel', 'Smith', '2006-03-23', 28660),
  ('Lee', 'Reynolds', '2011-05-22', 32720);
- Výběr dat z tabulky „employees“, kde „salary“ je více než 30000 a data jsou seřazena sestupně podle „hire_date“:
  SELECT * FROM employees WHERE salary > 30000 ORDER BY hire_date DESC;

1.2 Nerelační (NoSQL) databáze

První nerelační databáze vznikaly již v 60. letech 20. století. V současnosti používané typy, označované jako NoSQL databáze, se však používají až od počátku 21. století, kdy se začaly razantně snižovat náklady na úložiště dat.

V porovnání s relačními databázemi NoSQL databáze pro ukládání dat nepoužívají tabulky ani nevyžadují pevně dané schéma. Místo toho ukládají všechna data v rámci jediné datové struktury. Díky těmto vlastnostem jsou více flexibilní a nabízí výbornou škálovatelnost pro zpracování velkých objemů nestrukturovaných dat.

Existuje několik datových modelů, které mohou NoSQL databázové systémy využívat pro ukládání dat, lišící se svými silnými a slabými stránkami. [5][6]

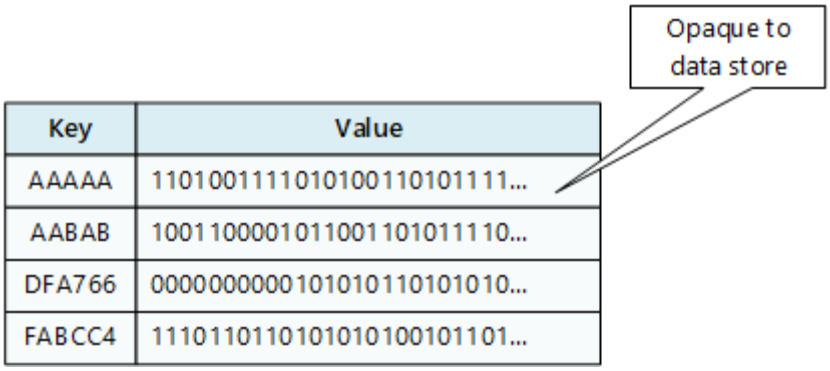
1.2.1 Úložiště typu klíč-hodnota

Nejjednodušší forma NoSQL databáze. Tento datový model je implementován jako velká hash tabulka a představuje slovník dvojic, kde každá položka má svůj jedinečný klíč a k němu přiřazenou hodnotu. Hodnota představuje uložená data, obvykle pole dat. Pokud bychom chtěli princip porovnat s relačními databázemi, svým významem by se klíč podobal primárnímu klíči a hodnota datům ze všech ostatních sloupců v rámci jednoho řádku. [5][7]

Obvykle jsou podporovány pouze základní operace dotazování, vkládání a mazání dat. Pokud chceme změnit některou uloženou hodnotu, musí se přepsat všechna existující data, které daná hodnota obsahuje.

Slovník nemá žádné vlastní schéma a každý záznam tedy může mít odlišnou strukturu uložených dat. Informace o schématu dat v konkrétním slovníku tedy musí umět správně interpretovat aplikace, která data z tohoto úložiště zpracovává.

Tento typ úložiště není vhodný, pokud je třeba vyhledávat mezi daty i podle jiných parametrů než jen podle jejich klíče. [7]



Obrázek 3. Znázornění datového modelu klíč-hodnota [7]

1.2.2 Dokumentové úložiště

Pro ukládání dat se používají entity zvané dokument. Každý dokument obsahuje sadu polí, které jsou pojmenovány pomocí textových řetězců, a k nim náležící hodnoty s objektovými daty.

Data uvnitř dokumentů mohou být kódována různými způsoby, obvykle jsou však ukládána ve formátu JSON. Hodnota pole může být buď skalární prvek, např. číslo, nebo složený prvek, jako je list. Pole v dokumentech jsou vystavena systému správy úložiště, což aplikacím umožňuje se na uložená data dotazovat a filtrovat je pomocí hodnot v těchto polích. [7]

Key	Document
1001	{ "CustomerID": 99, "OrderItems": [{ "ProductID": 2010, "Quantity": 2, "Cost": 520 }, { "ProductID": 4365, "Quantity": 1, "Cost": 18 }], "OrderDate": "04/01/2017" }
1002	{ "CustomerID": 220, "OrderItems": [{ "ProductID": 1285, "Quantity": 1, "Cost": 120 }], "OrderDate": "05/08/2017" }

Obrázek 4. Znázornění modelu dokumentového úložiště [7]

1.2.3 Sloupcové úložiště

Data jsou organizována do sloupců a řádků, svou koncepcí tedy tento druh úložiště může připomínat relační databázi. Sloupce jsou však rozděleny do skupin nazývaných jako rodiny sloupců. Každá rodina sloupců obsahuje sadu sloupců, které spolu logicky souvisí a obvykle jsou načítány nebo se s nimi manipuluje jako s jedním celkem.

Do každé rodiny lze dynamicky přidávat další nové sloupce, všechny řádky nemusejí mít uloženou hodnotu pro každý sloupec a data patřící konkrétní entitě mají v každé rodině sloupců pro svůj řádek stejný klíč. V ukázce modelu na obrázku (Obr. 5) jsou tyto vlastnosti znázorněny na dvou rodinách sloupců s daty o zákaznících.

Díky tomu, že se řádky pro libovolný objekt v rodině sloupců mohou dynamicky lišit, je tato forma úložiště ideální pro ukládání dat s různými schématy. [7]

CustomerID	Column Family: Identity
001	**First name:** Mu Bae **Last name:** Min
002	**First name:** Francisco **Last name:** Vila Nova **Suffix:** Jr.
003	**First name:** Lena **Last name:** Adamcyz **Title:** Dr.

CustomerID	Column Family: Contact Info
001	**Phone number:** 555-0100 **Email:** someone@example.com
002	**Email:** vilanova@contoso.com
003	**Phone number:** 555-0120

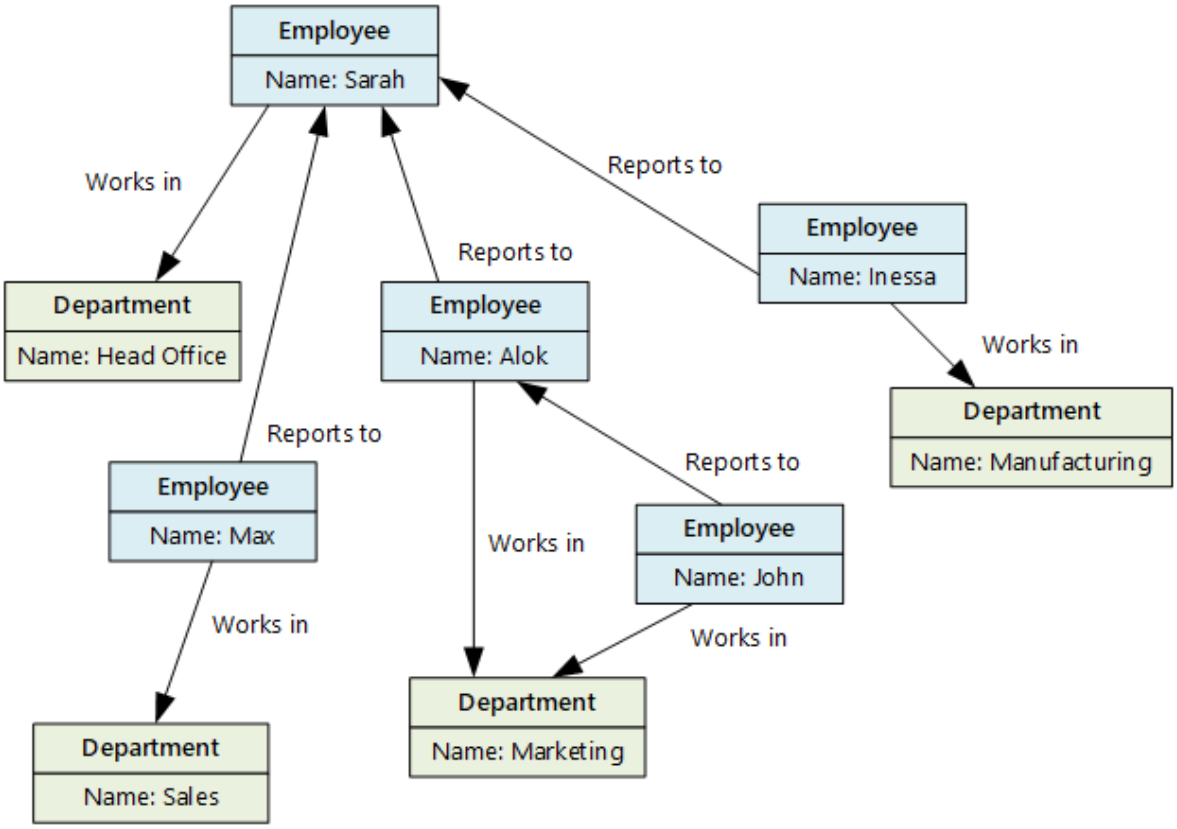
Obrázek 5. Znázornění modelu sloupcového úložiště [7]

1.2.4 Grafové úložiště

Grafové databáze kladou důraz na vztahy mezi datovými prvky. [6] Sestávají ze dvou typů prvků, uzlů a hran, pro vyjádření vzájemně propojených dat. Uzly představují entity a vztahy mezi nimi jsou reprezentovány jako hrany. Ty mohou mít také směr udávající povahu vztahu. Oba typy prvků mají vlastnosti, které poskytují informace o daném uzlu nebo hraně, podobně jako sloupce v tabulce.

Účelem tohoto modelu datového úložiště je umožnit aplikacím efektivně procházet sítí uzlů a hran a analyzovat vztahy mezi entitami. Lze tedy přímočaře a rychle provádět dotazy, jako například "Najdi všechny zaměstnance, kteří jsou přímo nebo nepřímo podřízení Sarah." nebo "Kdo pracuje ve stejném oddělení jako John?".

I u rozsáhlých grafů je tak možné v krátkém čase provádět komplexní analýzy. [7]



Obrázek 6. Znázornění modelu grafového úložiště [7]

1.3 Cloudové databáze

Společnosti také mohou využít tzv. cloudových databází, kde provádí administrativní úkoly, údržbu a technicky zajišťuje provoz poskytovatel služeb. Mohou podporovat nasazení relačních i NoSQL databází.

Mezi hlavní motivace pro jejich využití patří zejména snaha o úsporu nákladů, vyšší spolehlivost a snadná škálovatelnost. Mohou poskytovat lepší zabezpečení dat, nad kterým však společnost využívající těchto služby nemá přímou kontrolu. Společnost by si před přechodem na tento typ služeb měla ověřit, zda přechodem skutečně docílí snížení provozních nákladů.

Existují dvě hlavní varianty nasazení cloudových databází, lišící se zejména mírou zodpovědnosti ze strany společnosti, která tuto službu využívá. [8]

1.3.1 Tradiční databáze

Společnost si ponechává plnou zodpovědnost za správu a dohled nad svou databází, ale odpadá jí starost o provoz vlastní hardwarové infrastruktury, kterou v tomto případě zajišťuje

poskytovatel cloudových služeb. Společnost si zakoupí kapacitu virtuálního stroje, na kterém je databáze následně nasazena a provozována. Tato varianta tak umožňuje flexibilně měnit výkonností parametry DB infrastruktury dle současných požadavků společnosti, bez dalších nákladů na pořízení a instalaci vlastního hardware. [8]

1.3.2 Database as a service (DBaaS)

Je to služba poskytovaná společnostem v rámci předplatného. Plnou zodpovědnost za provoz této varianty nasazení databáze má poskytovatel cloudových služeb. Tato služba pro společnosti nabízí největší přidanou hodnotu, jelikož umožňuje bezstarostně využívat databázový systém, aniž by bylo nutné nakupovat a instalovat vlastní hardware, databázový software, nebo zaměstnávat vlastní databázové specialisty. Poskytovatel cloudu zajišťuje provoz po všech stránkách – od pravidelných upgradů, oprav a záloh, automatizovaného škálování výpočetního výkonu a velikosti úložiště, až po zajištění, že databázový systém zůstane vždy dostupný a zabezpečený. [8][9]

2 PLATFORMA MICROSOFT .NET

Microsoft .NET je zdarma dostupná, multiplatformní a open-source softwarová platforma pro vývoj aplikací pomocí tří různých programovacích jazyků, kterými jsou Visual Basic, F#, a z těchto tří nejrozšířenější objektově orientovaný C#.

Platformu lze použít pro vývoj různých typů aplikací:

- Webové a cloudové aplikace s ASP.NET Core a .NET Aspire.
- Mobilní aplikace s .NET MAUI.
- Konzolové aplikace s textovým rozhraním.
- Desktopové aplikace s grafickým uživatelským rozhraním s Windows Forms, WPF, MAUI a dalšími frameworky.
- Mikroslužby pro Docker kontejnery.
- Počítačové hry.

Vedle UI frameworků od Microsoftu existují i frameworky třetích stran, jakým je například Avalonia UI pro vývoj multiplatformních aplikací.

.NET nabízí automatickou správu paměti, je typově bezpečný a usnadňuje asynchronní programování díky klíčovým slovům async a await.

Pro .NET existuje téměř 400 000 knihoven přinášejících nejrůznější přídavnou funkcionalitu. Jsou distribuovány pomocí správce balíčků NuGet, a nabízeny jak přímo od Microsoftu, tak i od dalších vývojářů. [10][11]

3 .NET TECHNOLOGIE PRO PRÁCI S DATABÁZEMI

Kapitola teoreticky pojednává o třech klíčových technologiích pro práci s databázemi v rámci platformy .NET.

3.1 ADO.NET

ADO.NET, celým názvem ActiveX Data Objects .NET, je technologie pro přístup k datům, která se standardně používala pro vývoj aplikací před vznikem Entity Frameworku. Je implementována jako sada tříd definovaných v .NET Frameworku. Existují dva základní přístupy, jakými lze tyto třídy použít pro přístup k datům a jejich změnám. [12][13]

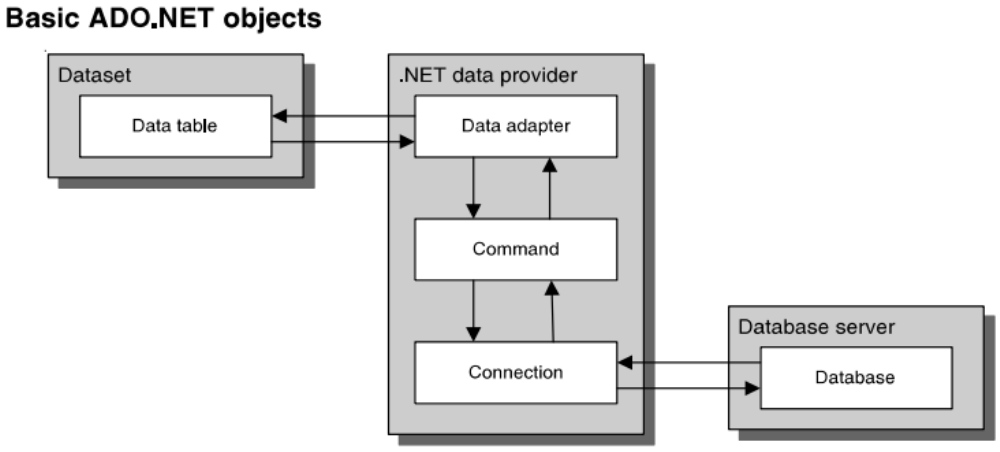
3.1.1 ADO.NET s datasety

Jednou z cest pro vývoj aplikací s přístupem k databázi je použití tzv. datasetů. Aplikace načte data z DB a uloží je do datasetu v mezipaměti, následně může v rámci tohoto datasetu přidávat, měnit nebo mazat řádky a později tyto změny uložit do databáze.

Pro načtení dat do tabulky definované uvnitř datasetu se používá tzv. data adapter. Jeho hlavním úkolem je zajišťovat tok dat mezi datasetem a databází. Data adapter k tomu používá commands, kterými se definují SQL příkazy, které mají být aplikovány.

Například command pro získání dat typicky definuje výraz Select, následně se command připojí k DB pomocí connection a předá tento Select výraz databázovému systému. Jakmile je vykonán, sada dat, označena jako set, která je výsledkem, je odeslána zpět do data adapteru, který výsledky uloží do tabulky.

Data nacházející se v datasetu jsou nezávislá na DB, odkud byla původně získána. Připojení k DB je obvykle uzavřeno ihned poté, co jsou vyžádaná data načtena a je znovu otevřeno až v moment, kdy je nutné s DB znovu komunikovat. Z tohoto důvodu musí aplikace pracovat s kopií dat, která je aktuálně načtená v datasetu. Tento přístup může vypadat jako komplikovaný, má však své výhody. Může například vést k lepšímu systémovému výkonu, jelikož není nutné dlouhodobě udržovat otevřená připojení. [13]



Obrázek 7. Základní ADO.NET objekty [13]

```
string connectionString =
    "Data Source=localhost\\SqlExpress;Initial Catalog=Payables;" +
    "Integrated Security=True";
SqlConnection connection = new SqlConnection(connectionString);

string selectStatement = "SELECT * FROM Terms";
SqlCommand selectCommand = new SqlCommand(selectStatement, connection);

SqlDataAdapter termsDataAdapter = new SqlDataAdapter(selectCommand);

DataSet termsDataSet = new DataSet;
```

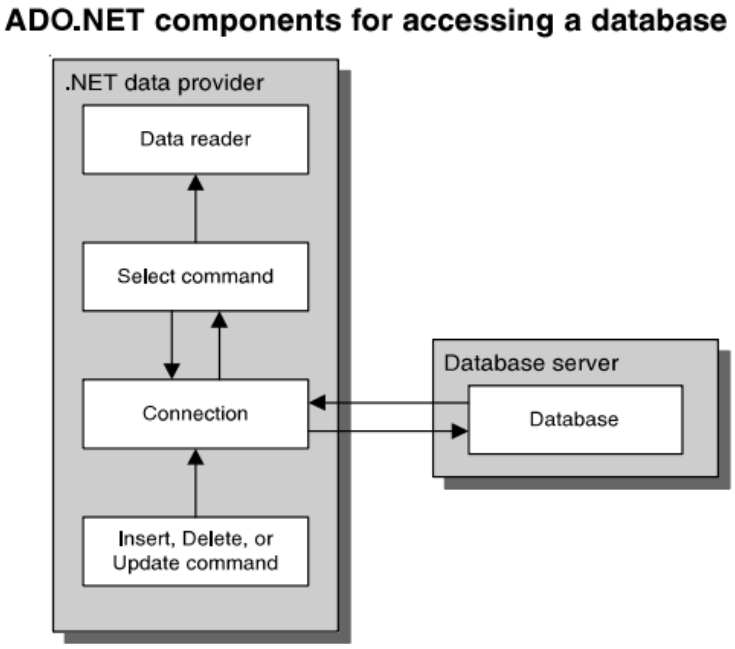
Obrázek 8. Ukázka C# kódu pro vytvoření základních ADO.NET objektů [13]

3.1.2 ADO.NET bez datasetů

Druhou variantou, jak vyvíjet aplikace s ADO.NET, je pracovat s databází napřímo.

U této varianty se nepoužívají datasety a příkazy nevykonává data adapter, ale spouštějí se napřímo. Pokud se vydáme touto cestou, je nutné napsat vlastní kód pro zpracování výsledku každého příkazu. Například při vykonání příkazu pro vložení, změnu nebo smazání záznamu, výsledkem bude celé číslo indikující počet řádků, které byly touto operací ovlivněny. Na základě výsledku je možné vyhodnotit, zda byla operace úspěšná.

Výsledkem spuštění Select příkazu je result set s vyžádanými řádky, pro jejichž přečtení se používá data reader objekt. Tento objekt nelze použít pro změnu řádků a řádky lze přečíst pouze po jednom a směrem vpřed. [13]



Obrázek 9. ADO.NET objekty pro přímý přístup k databázi [13]

```
string insertStatement =
    "INSERT Vendors (Description, DueDays) " +
    "VALUES (@Description, @DueDays)";
SqlCommand insertCommand = new SqlCommand(insertStatement, connection);
insertCommand.Parameters.AddWithValue("@Description", terms.Description);
insertCommand.Parameters.AddWithValue("@DueDays", terms.DueDays);
connection.Open();
int count = insertCommand.ExecuteNonQuery();
connection.Close();
```

Obrázek 10. Ukázka C# kódu s příkazem pro vložení řádku do databáze [13]

3.1.3 Data providers

Data providers v .NET poskytují ADO.NET třídy pro umožnění připojení k databázi a přímé práce s ní. Jsou rozdílné dle použitého databázového systému. Například data provider pro přístup k SQL Serveru lze do C# projektu přidat pomocí `using System.Data.SqlClient`.

Základní objekty, které data providers poskytují, jsou:

- Connection – Zakládá připojení k databázi.
- Command – Reprezentuje jednotlivý SQL výraz nebo uloženou proceduru, která může být spuštěna na databázovém serveru.

- Data reader – Poskytuje přístup k datům v databázi. K datům lze však přistupovat pouze směrem „vpřed“ a data jsou pouze pro čtení.
- Data adapter – Poskytuje propojení mezi objekty command a connection s datasetem. [13]

3.1.4 Třída SqlConnection

Předtím, než bude možné přistupovat k databázovým datům, je třeba vytvořit connection objekt, který toto připojení definuje. Třída specifická pro SQL Server s implementací toho základního objektu nese název SqlConnection. Nejdůležitější vlastností této třídy je connection string. Tento textový řetězec obsahuje informace, díky kterým je možné vytvořit připojení k databázi. Mezi obsažené informace patří název připojované databáze a serveru, na kterém se nachází, a přístupové údaje k tomuto serveru, typicky v podobě uživatelského jména a hesla. SqlConnection umožňuje manuálně otevírat a zavírat připojení, připojení by však mělo zůstávat otevřené pouze po dobu získávání a úprav dat. Proto je výhodnější namísto toho používat data adapter, který připojení k databázi otevírá a uzavírá automaticky dle potřeby. [13]

3.1.5 Třída SqlCommand

Pro účely spuštění SQL příkazu na databázovém serveru se vytváří SqlCommand objekt obsahující daný příkaz. Pro spouštění command objektů lze použít data adapter, třída SqlCommand však obsahuje další tři metody, které lze pro provedení příkazu použít.

Pro výběrové dotazy přichází v úvahu metoda ExecuteReader, jež vrací výsledky v podobě DataReader objektu. Další metodou je ExecuteScalar, která vrátí pouze tu hodnotu, která se v rámci výsledku dat nachází v prvním sloupci a řádku, proto je vhodná pouze v případech, kdy je jediná hodnota očekávána, například výsledek součtu všech hodnot ve sloupci.

Pokud příkaz obsahuje SQL výraz pro vložení, změnu nebo smazání dat, pro jeho vykonání je určena metoda ExecureNonQuery. Tato metoda vrací celočíselnou hodnotu udávající počet řádků, které byly tímto příkazem ovlivněny. [13]

3.1.6 Třída SqlDataAdapter

Úkolem data adaperu je propojit databázi s datasetem. SqlDataAdapter obsahuje 4 property pro identifikaci SQL příkazů, které data adapter používá pro přenos dat z databáze do datasetu a opačně.

SelectCommand property identifikuje command objekt, který se používá pro získávání dat z databáze. Pro spuštění takového příkazu a uložení obdržených dat do datasetu se používá Fill metoda. Následně může aplikace pracovat s daty v datasetu bez ovlivňování dat uvnitř databáze.

Pokud aplikace provede změny těchto dat, může použít Update metodu pro spuštění příkazů identifikovaných pomocí InsertCommand, UpdateCommand nebo DeleteCommand properties. Tak aplikace odešle provedené změny do databáze. [13]

3.1.7 Třída SqlDataReader

Data reader nabízí efektivní způsob čtení řádků v sadě dat, která byla navrácena jako výsledek dotazu. Data reader nedovoluje modifikaci řádků, jelikož je pouze pro čtení. Navíc umožňuje číst řádky pouze dopředným směrem a jakmile je přečten další řádek, ten předchozí již není k dispozici. Pro přečtení následujícího řádku se používá metoda Read. [13]

3.2 Dapper

Dapper je open-source Micro ORM knihovna pro .NET aplikace. Dapper je postaven nad technologií ADO.NET a vývojářům umožňuje rychlý a snadný přístup k datům z databází bez nutnosti psát zdlouhavý kód. Dále umožňuje spouštět dotazy SQL, mapovat výsledky na objekty a spouštět uložené procedury. Je distribuován jako balíček ve službě NuGet. [14]

	Micro ORM	ORM
Map queries to objects	✓	✓
Caching results	✗	✓
Change tracking	✗ [1]	✓
SQL generation	✗ [2]	✓
Identity management	✗	✓
Association management	✗	✓
Lazy loading	✗	✓
Unit of work support	✗	✓
Database migrations	✗	✓

Obrázek 11. Porovnání mezi Micro ORM a plnohodnotným ORM [14]

Jedná se o jednoduchý, ale výkonný nástroj pro mapování objektů pro jakýkoli jazyk .NET, jako je C#, který umožňuje vývojářům rychle a snadno mapovat výsledky dotazů z ADO.NET data reader objektů na instance .NET objektů.

Má výbornou podporu pro asynchronní i synchronní databázové dotazy a dávkování více dotazů do jednoho volání. [14]

3.2.1 Dotazování na data

Stačí zadat dotaz pro výběr dat s parametry a poté Dapper automaticky namapuje výsledné sloupce na jejich odpovídající vlastnosti v modelu. Příklad dotazu je vidět na obrázku (Obr. 12). [15]

```
// Connect to the database
using (var connection = new SqlConnection(connectionString))
{
    // Create a query that retrieves all books with an author name of "John Smith"
    var sql = "SELECT * FROM Books WHERE Author = @authorName";

    // Use the Query method to execute the query and return a list of objects
    var books = connection.Query<Book>(sql, new { authorName = "John Smith" }).ToList();
}
```

Obrázek 12. Dotazování na data pomocí knihovny Dapper [15]

3.2.2 Provádění příkazů, které nejsou určeny k vrácení výsledků

Dapper umožňuje spouštět SQL příkazy, které nejsou určeny pro vrácení výsledků s daty. Patří mezi ně příkazy Insert, Update a Delete.

K provedení těchto příkazů se používá metoda Execute z rozhraní IdbConnection, kterou Dapper implementuje. Tato metoda vrací celočíselnou hodnotu, představující počet ovlivněných řádků po dokončení příkazu. [16]

```
string sql = "INSERT INTO Customers (Name, Email) VALUES (@Name, @Email);"
object[] parameters = { new { Name = "John Doe", Email = "jdoe@example.com" } };

using (var connection = new SqlConnection(connectionString))
{
    connection.Execute(sql, parameters);
}
```

Obrázek 13. Příklad použití Execute metody [16]

3.3 Entity Framework Core

Entity Framework Core (EF Core) je rozšiřitelná, open source multiplatformní ORM technologie pro přístup k datům. Umožňuje pracovat s databází s pomocí .NET objektů a podporuje spoustu různých relačních i nerelačních databázových systémů.

S EF Core je přístup k datům prováděn pomocí modelu, který se skládá z tříd pro entity a kontextu, představující relaci s databází. Tento kontextový objekt umožňuje dotazování na data a manipulaci dat. [17]

3.3.1 Co představuje termín ORM

ORM, neboli Object Relational Mapper, je software určený k překladu mezi reprezentací dat používanou databázemi a tou používanou v rámci objektově orientovaného programování. Jelikož se tyto dva způsoby práce s daty liší, snahou ORM je tyto rozdíly eliminovat. ORM slouží jako abstraktní vrstva mezi aplikací a databází, aby se vývojář nemusel zaobírat interními rozdíly v návrhu designu těchto dvou systémů.

Z pohledu vývojáře ORM umožňuje pracovat s daty z databáze pomocí stejných struktur a mechanismů jako s kterýmkoliv jiným typem interních dat. [18]

3.3.2 Database providers

EF Core je distribuován v podobě NuGet balíčků. Je třeba vybrat a nainstalovat balíček v závislosti na tom, jaký databázový systém bude aplikace používat. Tyto balíčky jsou nazývány jako database providers, pouze některé z nich jsou součástí oficiálního projektu vývoje EF Core a podporovány tak přímo Microsoftem. Mezi hlavní z nich patří:

- Microsoft.EntityFrameworkCore.SqlServer
- Microsoft.EntityFrameworkCore.Sqlite
- Npgsql.EntityFrameworkCore.PostgreSQL
- Pomelo.EntityFrameworkCore.MySql
- MongoDB.EntityFrameworkCore

Společné vlastnosti a funkce pro různé databázové systémy jsou implementovány jako součást základních EF Core komponent. Poskytovatelé databází však mohou rozšířit EF Core o funkce jedinečné pro konkrétní databázové typy, které nejsou univerzálně platné. [19]

3.3.3 Databázový kontext

Hlavní třídou, která koordinuje funkčnost EF Core pro daný datový model, je kontextová třída databáze DbContext. Vlastní třída, která je odvozená (dědí) z DbContext, určuje, které entity mají být zahrnuty v datovém modelu, a některá chování EF Core lze uvnitř kontextu přizpůsobit.

Instance DbContext představuje relaci s databází a lze ji použít k dotazování a ukládání instancí vlastních entit. Životnost konkrétní instance je obvykle velmi krátká, jelikož je designována tak, aby byla použita pro jedinou „jednotku práce“. Typická jednotka práce zahrnuje tyto kroky:

- Vytvoření instance třídy DbContext.
- Sledování instancí entit, které byly buď navráceny jako výsledek dotazu na data, anebo byly nově přidány do databázového kontextu.
- Provedení změn ve sledovaných entitách dle potřeby.
- Volání metody SaveChanges, při kterém EF Core vyhodnotí provedené změny a zapíše je do databáze.
- Po tomto je daná instance třídy DbContext zrušena.

Instanci DbContextu lze vytvořit standardním způsobem, tedy použitím klíčového slova new. V ASP.NET Core aplikacích se pro to ale standardně používá dependency injection, do kterého lze DbContext zaregistrovat. To zajistí, že proběhne vytvoření instance pro každý HTTP požadavek, a provede se automatické zrušení instance, jakmile se požadavek dokončí.

Výchozím bodem pro veškerou konfiguraci DbContextu je metoda OnConfiguring, kterou lze přepsat klíčovým slovem override v těle třídy odvozené z DbContextu a pomocí jejího parametru optionsBuilder provádět vlastní konfiguraci.

Database provider se pro kontext konfiguruje specifickou „Use“ metodou, poskytovanou nainstalovaným providerem. Pro MS SQL server databázi vypadá volání této metody takto: UseSqlServer(connectionString), a pro porovnání s SQLite takto: UseSqlite(connectionString).

Connection string je specifikace způsobu připojení ke zdroji dat. [20]

```
public class ApplicationDbContext : DbContext
{
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlServer(@"Server=(localdb)\mssqllocaldb;Database=Test;ConnectRetryCount=0");
    }
}
```

Obrázek 14. Konfigurace databázového providera v OnConfiguring [20]

Případné další konfigurace lze řetězit na optionsBuilder, na jejich pořadí nezáleží.

3.3.4 Model

EF Core používá model k popisu toho, jak jsou typy entit aplikace mapovány na databázi. Tento model je postaven pomocí sady konvencí, které hledají obvyklé vzory. Model lze poté upravit pomocí atributů známých jako anotace dat, nebo pomocí volání ModelBuilder metod v OnModelCreating, známých také jako fluent API. Obě tyto funkce přepíší konfiguraci provedenou podle konvencí.

Pro konfiguraci modelu pomocí fluent API je třeba přepsat metodu OnModelCreating ve vlastním odvozeném kontextu, a to obdobným způsobem jako u metody OnConfiguring. Tento způsob konfigurace má nejvyšší prioritu, přepíše tedy jak konvence, tak případné anotace dat. Konfigurace je provedena v pořadí zápisu volání metod. [21]

```
internal class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    #region Required
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>()
            .Property(b => b.Url)
            .IsRequired();
    }
    #endregion
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
}
```

Obrázek 15. Konfigurace modelu pomocí fluent API v OnModelCreating [21]

Další možností pro konfiguraci modelu je aplikace atributů pro anotaci dat do tříd entit a jejich properties.

```
internal class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
}

[Table("Blogs")]
public class Blog
{
    public int BlogId { get; set; }

    [Required]
    public string Url { get; set; }
}
```

Obrázek 16. Konfigurace modelu pomocí atributů v entitě [21]

3.3.5 Typy entit a konvence

Zahrnutí property DbSet určitého typu do vlastního kontextu znamená, že je tento typ zahrnut do modelu EF Core. Takový typ se obvykle označuje jako entita. EF Core může číst instance entit z databáze a zapisovat je do ní. A pokud používána relační databáze, EF Core umí vytvářet tabulky pro entity prostřednictvím migrací.

Dále jsou zahrnuty entity, které jsou specifikovány v metodě OnModelCreating, stejně jako všechny typy, které jsou nalezeny rekurzivním průzkumem navigačních properties jiných objevených typů entit.

Konkrétní typ nebo jen některou jeho property lze z modelu vyjmout pomocí atributu NotMapped nebo fluent API metodou Ignore. [22]

```
[NotMapped]
public class BlogMetadata
{
    public DateTime LoadedFromDatabase { get; set; }
}
```

Obrázek 17. Vyjmutí entity pomocí atributu [22]

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Ignore<BlogMetadata>();
}
```

Obrázek 18. Vyjmutí entity pomocí fluent API [22]

Podle konvence bude každý typ entity nastaven tak, aby se mapoval na databázovou tabulku se stejným názvem, jaký má property DbSet, jež entitu zpřístupňuje.

Manuálně lze tento název změnit atributem `Table("blogs")` nebo metodou fluent API:

```
modelBuilder.Entity<Blog>()
        .ToTable("blogs");
```

K dalším příkladům chování EF Core podle konvencí lze jmenovat automatické vyhodnocení, která property představuje primární klíč na základě jejího názvu. V případě třídy entity „Truck“ by EF Core property „Id“ nebo „TruckId“ považoval za primární klíč i bez manuální specifikace např. atributem Key. Pokud navíc bude tato property číselného nebo GUID typu, nastaví se automatické generování hodnoty pro tento sloupec v databázovém systému. [23][24]

3.3.6 Reprezentace vztahů

Vztah definuje, jak spolu dvě entity souvisí. Vztahy se v EF Core zapisují pomocí navigačních properties. Pro příklad lze použít dvě entity, Blog a Post. Blog může obsahovat více příspěvků a zároveň každý příspěvek patří do nějakého blogu. Tento 1:M vztah by bylo možné zapsat přidáním navigační property `public Blog Blog { get; set; }` v entitě Post a `public ICollection<Post> Posts { get; }` v entitě Blog.

Každý vztah může být povinný nebo volitelný, což je určeno symbolem otazníku u property. Nastavení vztahů lze provést na základě konvence pomocí správně pojmenovaných navigačních properties nebo pomocí deklarace s fluent API. [25]

3.3.7 Databázová schémata

Datové modely se běžně mění s tím, jak jsou implementovány nové funkce do aplikace. Entity a properties mohou být přidávány a odebírány a jejich konfigurace se může změnit. Z toho důvodu je třeba odpovídajícím způsobem měnit i databázové schéma. Databázové migrace poskytuje možnost postupně aktualizovat schéma databáze, aby odpovídalo datovému modelu aplikace, při zachování stávajících dat v databázi. [26]

Existuje také druhý přístup ke správě databázových schémat, a tím je tzv. reverzní inženýrství. Tento přístup umožňuje vytvořit databázový kontext a třídy pro entity na základě schéma již existující databáze. Dá se tedy označit jako opačný postup oproti migracím a lze provést příkazem Scaffold-DbContext s parametrem connection stringu v konzoli Správce balíčků. [27]

II. PRAKTICKÁ ČÁST

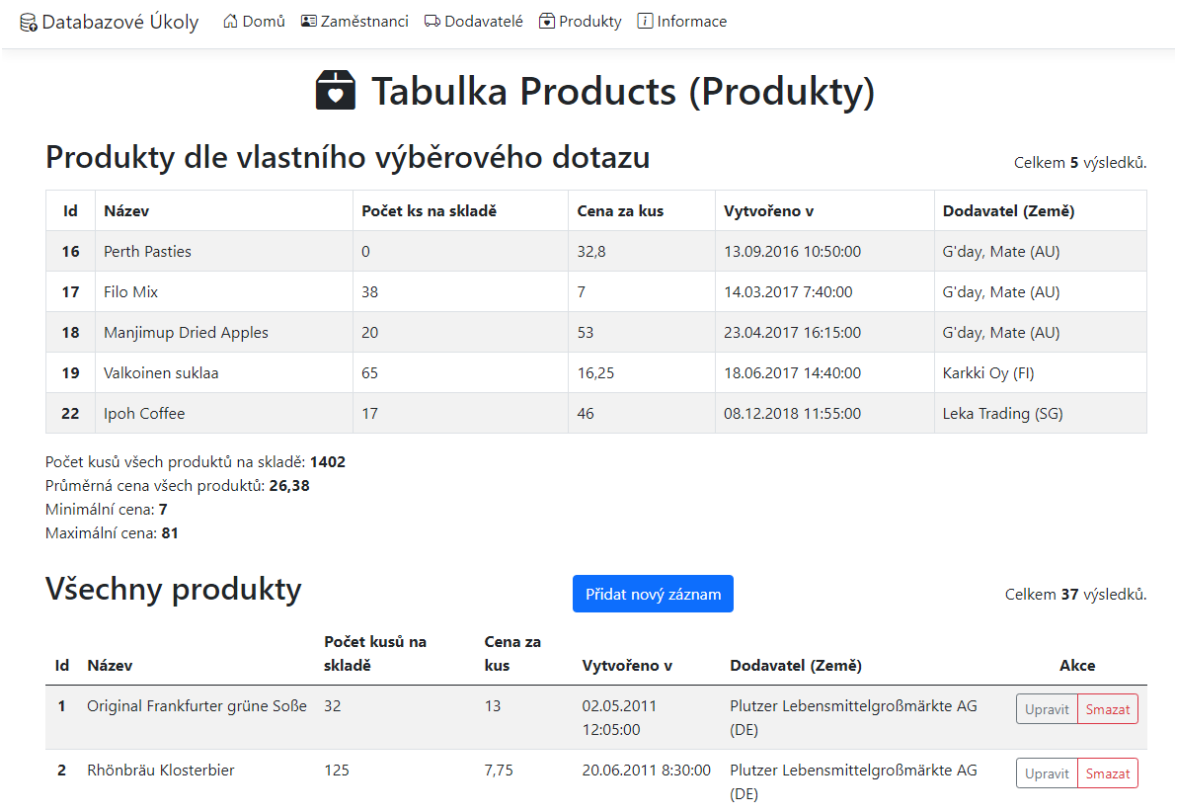
4 NÁVRH ÚKOLŮ PRO STUDENTY

Jednotlivé úkoly jsou navrženy formou postupné návaznosti, vedoucí k plynulému doplňování funkcionality databáze do aplikace. Pro tvorbu úkolů je zvolena technologie Entity Framework Core.

Úkoly obsahují konkrétní zadání, postup řešení se zdrojovými kódy a možné kontrolní otázky.

Pro možnost vypracování úkolů je přichystán jednoduchý ASP.NET Core (MVC) projekt pro snadnou prezentaci a modifikaci dat v přirozeném prostředí webové aplikace. Do této aplikace je v rámci úkolů postupně implementována funkcionalita databáze a její součástí jsou předpřipravená vzorová data.

Vzorová data byla inspirována a z větší části převzata z ukázkové databáze Microsoft Northwind. [28]



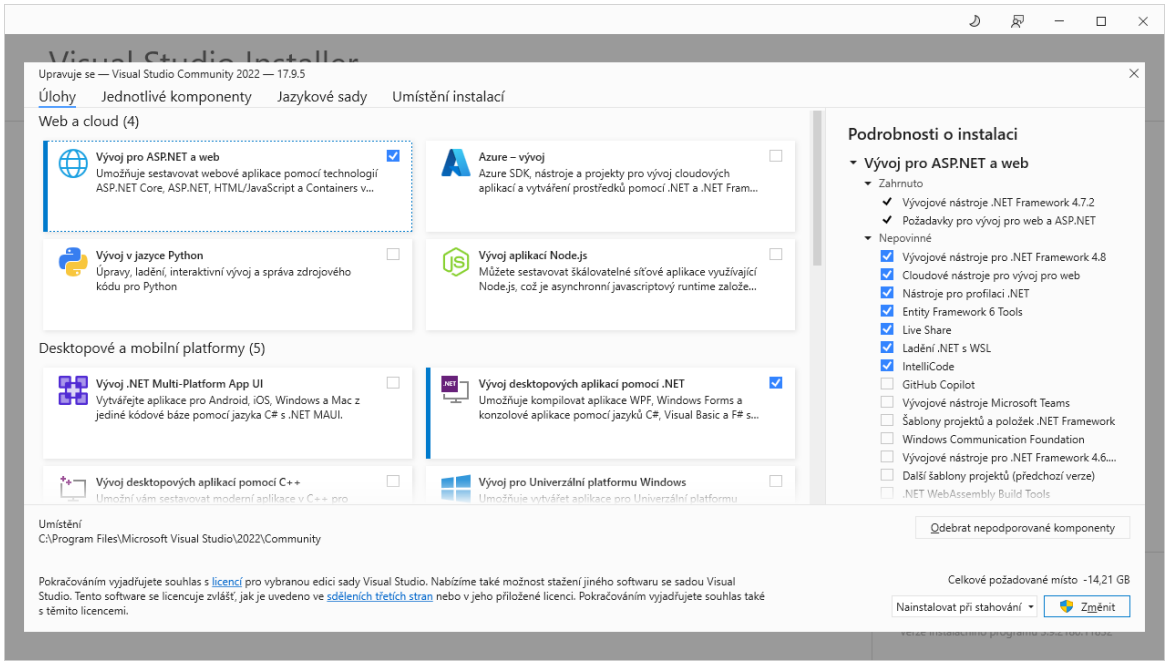
Obrázek 19. Ukázka rozhraní připravené aplikace pro databázové úkoly

Řešení úkolů a přiložené projekty jsou koncipovány primárně pro softwarové technologie Microsoft .NET 8, OS Windows, IDE Microsoft Visual Studio 2022 a databázi typu SQLite

3. Pro vypracování úkolů je však možné využít i jiný operační systém a jiné IDE s podporou pro Visual Studio projekty, jakým je například multiplatformní JetBrains Rider.

4.1 Postup instalace požadovaného softwaru

1. Stáhněte si **Microsoft Visual Studio Community** dostupné zdarma z oficiálních stránek: https://visualstudio.microsoft.com/cs/free-developer-offers
2. Při instalaci označte součásti „Vývoj pro ASP.NET a web“ a „Vývoj desktopových aplikací pomocí .NET“ (viz Obr. 11).



Obrázek 20. Instalace součástí Visual Studia

Vhod přijde také aplikace pro prohlížení struktury a obsahu SQLite databáze, aby bylo možné snadno zkontrolovat provedené změny. Vhodnou volbou je open-source aplikace **DB Browser for SQLite**, která je dostupná pro všechny běžně používané operační systémy a ke stažení ji lze nalézt na tomto odkazu: https://sqlitebrowser.org/dl

4.2 Instalace NuGet balíčků a připojení k SQLite databázi

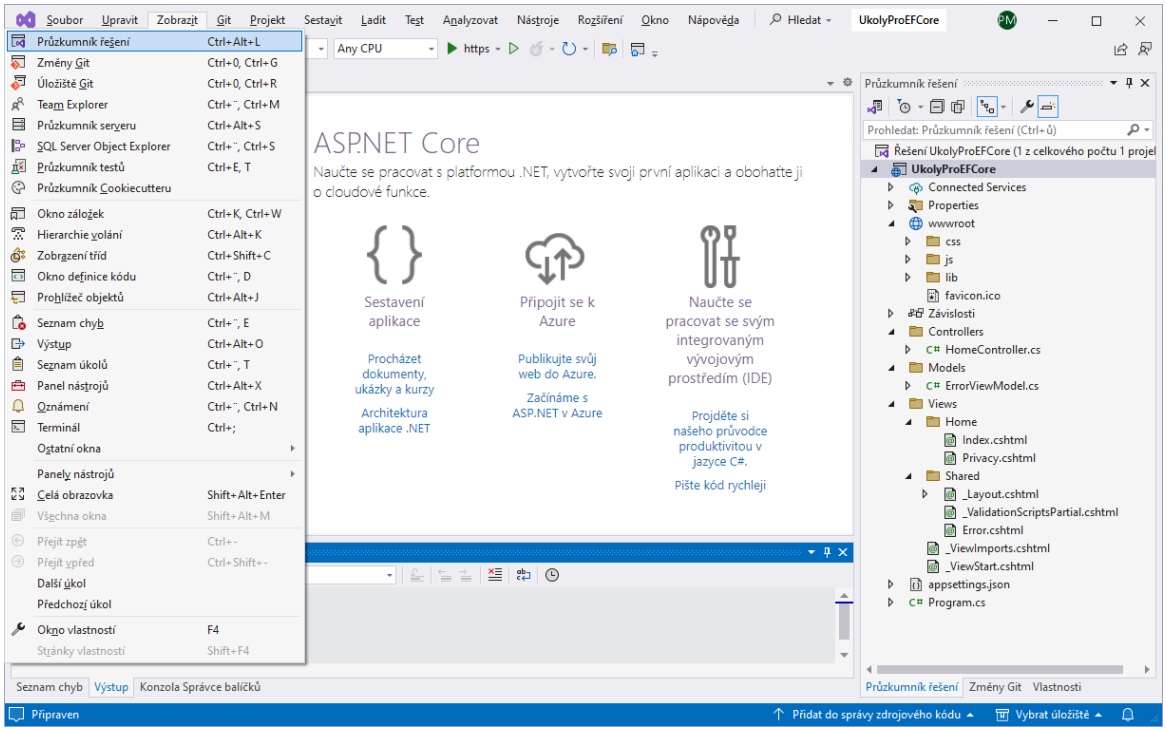
4.2.1 Zadání úkolu

Stáhněte si předpřipravený projekt v podobě jednoduché ASP.NET Core MVC aplikace. Otevřete projekt aplikace uvnitř Visual Studia. Poté nainstalujte potřebné NuGet balíčky pro implementaci Entity Framework Core do projektu a podporu připojení k SQLite databázi. Také si nainstalujte NuGet balíček Microsoft.EntityFrameworkCore.Tools, který budete potřebovat pro následující úkoly.

Aplikace zatím nepůjde spustit kvůli chybějícím součástem, které si do projektu postupně doplníte v rámci následujících úkolů.

4.2.2 Postup řešení

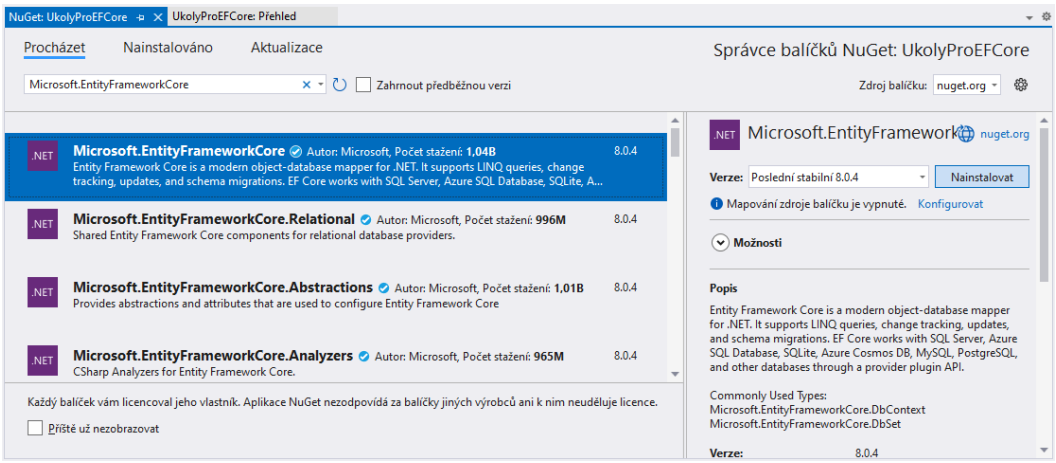
Stáhněte si a otevřete předpřipravený Visual Studio projekt s aplikací. V okně Průzkumník řešení se zobrazuje struktura tohoto projektu, všechny složky lze rozbalit kliknutím na malou šipku. Pokud byste toto podokno omylem zavřeli, lze jej snadno vrátit zpět pomocí tlačítka Zobrazit v záhlaví hlavního okna Visual Studia a volby Průzkumník řešení z právě zobrazené nabídky.



Obrázek 21. Znovuotevření okna Průzkumník řešení projektu ve Visual Studiu

Přejděte k instalaci potřebných NuGet balíčků pro podporu EF Core a SQLite databáze.

V Průzkumníku řešení klikněte pravým tlačítkem na Závislosti a zvolte Spravovat balíčky NuGet... Přepněte se na kartu Procházet a do pole hledat zadejte výraz **Microsoft.EntityFrameworkCore**. Po kliknutí na tento balíček jej z pravého sloupce nainstalujte v aktuální verzi a potvrďte akci. Během toho se také nainstalují všechny závislosti na dalších balíčcích.



Obrázek 22. Instalace NuGet balíčků do projektu

Stejným postupem nainstalujte ještě tyto dva další balíčky, které budete potřebovat později:

1. **Microsoft.EntityFrameworkCore.Tools** – Sada nástrojů pro provádění úloh návrhu databáze během vývoje. S nástroji lze pracovat v Konzoli Správce balíčků (angl. Package Manager Console) uvnitř Visual Studia. [29]
2. **Microsoft.EntityFrameworkCore.Sqlite** – EF Core přistupuje k různým typům databází pomocí tzv. database providers. Dle toho, s jakým typem databáze chcete svou aplikaci propojit, je třeba v podobě NuGet balíčku nainstalovat odpovídacího database providera. [19]

V tento okamžik projekt obsahuje všechny podstatné součásti pro podporu práce s databází typu SQLite a s využitím Entity Framework Core.

4.2.3 Kontrolní otázky

- Jak se nazývá oficiální systém distribuce a správy přídavných knihoven pro .NET aplikace?
- Je podporována komunikace s databázovými systémy již po instalaci samotného balíčku Microsoft.EntityFrameworkCore?

- Je database provider pro SQLite podporován přímo Microsoftem?

4.3 Vytvoření entit a jejich vlastností

4.3.1 Zadání úkolu

Vytvořte 4 entity definující, jaká data se budou ukládat do jednotlivých databázových tabulek. Entity se budou jmenovat: **Supplier**, **Employee**, **Country** a **Product**. Třídy s entitami přidejte do připravené složky Models/**Entities**.

Následně do každé entity přidejte vhodné properties (vlastnosti). Properties budou definovat sloupce v tabulce databáze.

Kromě primárního klíče s názvem Id typu int, který bude definován ve všech entitách totožným způsobem, přidejte entitám jejich properties s názvy a datovými typy dle zadání v tabulce (Tab. 1). Sloupce, které mají povolenou prázdnou hodnotu NULL, mají u datového typu symbol otazníku. Cizí klíče jsou označeny zkratkou FK.

Nezapomeňte vhodně doplnit také navigační properties, které souvisí s použitím cizích klíčů a slouží pro snadné spojování tabulek, čehož bude aplikace využívat například při získávání dat z databáze.

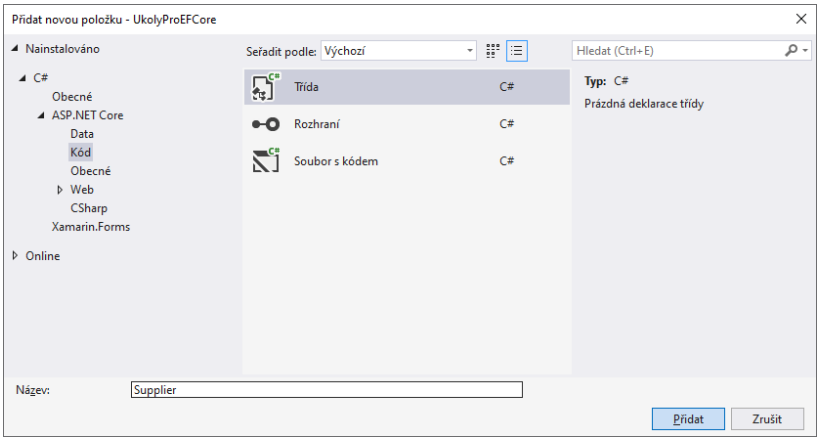
Tabulka 1. Entity a jejich properties pro úkol

Supplier	Name: string	Street: string?	City: string?	PostalCode: string?	CountryId: int? (FK)
Employee	FirstName: string	LastName: string	HireDate: DateOnly	JobTitle: string	Email: string?
Country	Name: string	Code: string			
Product	Name: string	StockUnits: int	Price: decimal?	CreatedOn: DateTime	SupplierId: int? (FK)

4.3.2 Postup řešení

Pro zachování dobré přehlednosti struktury projektu byla pro vlastní entity vytvořena samostatná složka **Entities**. Pro vytvoření nové entity zvolte z kontextové nabídky této složky možnost Přidat → Třída...

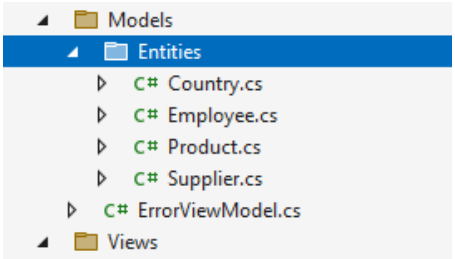
Do pole Název napište jméno vytvářené entity. Pro vytvoření první entity v pořadí dle tabulky uveďte název **Supplier** a potvrďte Enterem nebo tlačítkem Přidat. Příponu .cs je možné z názvu vynechat, jelikož bude přidána automaticky.



Obrázek 23. Vytvoření nové třídy pro entitu

Vytvořený soubor se ihned otevře v nové kartě Visual Studia.

Opakováním tohoto postupu vytvořte třídy také pro ostatní entity dle zadání.



Obrázek 24. Vytvořené entity v Průzkumníku řešení

Nyní do entit doplňte jejich properties (vlastnosti). Můžete začít zase od entity Supplier, na kterou se tedy přepněte přes Průzkumník řešení. V základním nastavení bude pořadí properties v entitě odpovídat také pořadí sloupců v tabulce vytvořené databáze.

Zde do těla třídy uveďte každou property na samostatný řádek. Jejich zápis má tento formát:
`public datovy_typ NazevProperty { get; set; }`

Properties, jež nemusejí obsahovat žádnou hodnotu (mohou být null), se označují symbolem otazníku za jejich datovým typem. EF Core namapuje .NET datové typy na nejbližší datové typy, které jsou podporované použitým databázovým systémem.

Primární klíč není nutné specificky označovat, stačí, když je srozumitelně pojmenován jako „Id“, případně „SupplierId“. EF Core je schopen tuto property jako primární klíč automaticky rozpoznat na základě tzv. konvence.

Properties, které představují cizí klíč, se zapisují stejným způsobem jako ostatní properties. Aby je však EF Core jako cizí klíče správně rozpoznal, jejich název by měl být ve formátu názvu entity, na kterou odkazují, s přídomkem Id.

V rámci EF Core se k nim navíc přidávají také další tzv. navigační properties, a to na obě strany jednoho vztahu. Přes navigační properties lze snadno přistupovat k přidruženým datům z dalších entit, např. při úpravě nebo získávání dat z databáze. Jejich datový typ koresponduje s entitou, na kterou cizí klíč odkazuje, a jejich název by se měl také shodovat. Pokud je totiž dodržen tento zápis, EF Core dokáže tyto vazby sám správně vyhodnotit na základě konvencí, bez nutnosti další manuální konfigurace.

Jedna instance entity Supplier může mít přiřazenu 0-1 zemi a 0-M produktů. Navigační property pro zemi bude typu Country a navigační property pro produkty bude typu kolekce produktů, jejíž instance bude rovnou inicializovaná. V řešení úkolu byl konkrétně použit typ IList<Product>.

4.3.3 Zdrojový kód řešení

Takto by měla vypadat úplná entita Supplier:

```
public class Supplier
{
    public int Id { get; set; }

    public string Name { get; set; }
    public string? Street { get; set; }
    public string? City { get; set; }
    public string? PostalCode { get; set; }

    public int? CountryId { get; set; }
    public Country? Country { get; set; }

    public IList<Product> Products { get; set; } = new List<Product>();
}
```

Obdobným způsobem doplňte všechny properties i ke zbývajícím entitám podle tabulky ze zadání úkolu.

4.3.4 Kontrolní otázky

- Jakým způsobem označujeme properties, které mohou obsahovat prázdnou hodnotu null?
- Proč je důležité definovat navigační properties v entitách?
- Jakým výrazem se nazývá vlastnost, kdy EF Core dokáže správně vyhodnotit některé skutečnosti a vazby na základě názvu a datového typu property, bez nutnosti další konfigurace?

4.4 Vytvoření vlastního databázového kontextu

4.4.1 Zadání úkolu

Vytvořte vlastní databázový kontext ve složce **Data**, která je pro vás připravena v nejvyšší úrovni struktury projektu. Pojmenujte jej jako **CompanyDbContext**. Zahrňte do datového modelu všechny vlastní entity a nastavte kontext tak, aby inicializoval výchozí data ze třídy **InitialData** pro všechny 4 databázové tabulky.

Následně nastavte připojení k SQLite databázi, konkrétně connection string a inicializaci vašeho databázového kontextu pro dependency injection v rámci ASP.NET Core aplikace. Databáze se bude nacházet v cestě Data/Db/CompanyDatabase.db.

4.4.2 Postup řešení

Do složky Data přidejte novou třídu a pojmenujte ji jako CompanyDbContext.

Tato třída bude dědit ze třídy `DbContext`, kterou poskytuje EF Core. Tímto nastavíte, aby se EF Core používal pro váš datový model. Visual Studio by mělo nabídnout doplnění názvu třídy, které poté stačí potvrdit, čímž se automaticky vloží výraz `using Microsoft.EntityFrameworkCore` pro import potřebného balíčku.

Třída bude obsahovat prázdný konstruktor s parametrem `DbContextOptions<CompanyDbContext> options`, a bude volat základní konstruktor s tímto parametrem.

V CompanyDbContext nyní specifikujte, že si přejete do datového modelu zahrnout všechny entity z předchozího úkolu, pomocí properties typu `DbSet<Entita>`. DbSet představuje konkrétní tabulku databáze, jejich pojmenování bude odpovídat názvu dané tabulky.

Následně pro entity zahrňte vzorová data ze třídy InitialData. K tomuto účelu budete muset překrýt za pomocí klíčových slov protected override metodu `OnModelCreating` s parametrem `ModelBuilder modelBuilder`, která je součástí zděděné EF Core `DbContext` třídy. Tato metoda umožňuje vlastní přizpůsobení konfigurace datového modelu EF Core.

Pomocí parametru nastavte ke všem čtyřem entitám jejich počáteční vzorová data zřetězeným voláním metod v tomto formátu:

```
modelBuilder.Entity<Supplier>()
    .HasData(InitialData.GetSuppliers());
```

Jako poslední krok nezapomeňte zavolat tuto metodu z rodičovské třídy.

Nyní je třeba tento CompanyDbContext do aplikace zaregistrovat a připojit samotnou databázi pomocí connection stringu.

V ASP.NET Core projektech se connection string obvykle ukládá do souboru appsettings.json. Přidejte zde nové pole s názvem "ConnectionStrings" a dovnitř dvojici klíče s hodnotou `"SQLite": "Data Source=Data/DB/CompanyDatabase.db"`.

Connection string se liší dle typu použitého databázového systému. U databáze typu SQLite stačí za rovnítko uvést cestu k souboru s databází. Respektive umístění, kde se nová databáze vytvoří, až přidáte a aplikujete svou první databázovou migraci.

Jako poslední krok přidejte do Program.cs následující dva řádky kódu pro zaregistrování vlastního DbContextu v ASP.NET Core pro dependency injection. První řádek načte connection string ze souboru appsettings.json. Druhý řádek zaregistruje váš CompanyDbContext jako službu pro dependency injection a připojí jej k SQLite databázi.

Kód vložte nad volání metody Build pro sestavení webové aplikace.

```
string connectionString = builder.Configuration.GetConnectionString("SQLite");
builder.Services.AddDbContext<CompanyDbContext>(optionsBuilder =>
optionsBuilder.UseSqlite(connectionString));
```

Nyní by již měl projekt správně komunikovat s databázovým systémem. Tuto skutečnost si ověříte hned v následujícím úkolu.

4.4.3 Zdrojový kód řešení

Takto by měl vypadat celý obsah třídy CompanyDbContext:

```
public class CompanyDbContext : DbContext
{
    public CompanyDbContext(DbContextOptions<CompanyDbContext> options)
        : base(options)
    {
    }

    public DbSet<Supplier> Suppliers { get; set; }
    public DbSet<Employee> Employees { get; set; }
    public DbSet<Country> Countries { get; set; }
    public DbSet<Product> Products { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Supplier>()
            .HasData(InitialData.GetSuppliers());
        modelBuilder.Entity<Employee>()
            .HasData(InitialData.GetEmployees());
        modelBuilder.Entity<Country>()
            .HasData(InitialData.GetCountries());
        modelBuilder.Entity<Product>()
            .HasData(InitialData.GetProducts());

        base.OnModelCreating(modelBuilder);
    }
}
```

Obsah souboru appsettings.json s vloženým connection stringem:

```
{
  "ConnectionStrings": {
    "SQLite": "Data Source=Data/Db/CompanyDatabase.db"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```

A nakonec registrace CompanyDbContext pro dependency injection v souboru Program.cs:

```
// Add services to the container.
builder.Services.AddControllersWithViews();

// Registrace databázového kontextu a jeho připojení k SQLite databázi pomocí
connection stringu.
string connectionString = builder.Configuration.GetConnectionString("SQLite");
builder.Services.AddDbContext<CompanyDbContext>(optionsBuilder =>
optionsBuilder.UseSqlite(connectionString));
```

```
var app = builder.Build();

// … další obsah souboru zkrácen
```

4.4.4 Kontrolní otázky

- K čemu slouží databázový kontext?
- Z jaké EF Core třídy musí dědit náš vlastní databázový kontext?
- Jaký datový typ bude mít property, jejíž úkolem je zahrnout entitu Product do datového modelu EF Core?
- Kterou metodu musíte ve svém databázovém kontextu překrýt, abyste mohli kontext nakonfigurovat dle vlastních požadavků?
- K čemu slouží connection string?
- Bude se connection string lišit v závislosti na databázovém systému, se kterým má být aplikace propojena?

4.5 Správa databáze pomocí konzoly Správce balíčků

4.5.1 Zadání úkolu

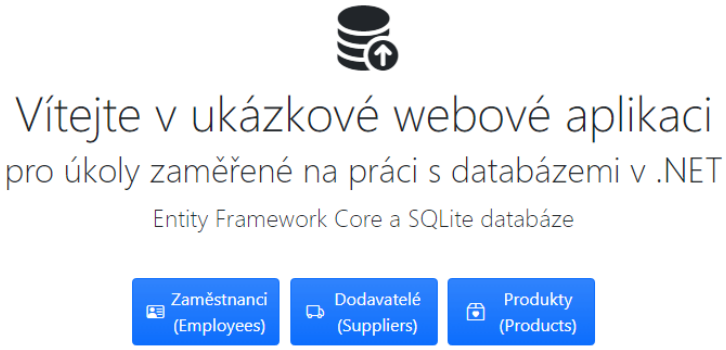
Vyzkoušejte si příkazy pro práci s databází v konzoli Správce balíčků (anglicky Package Manager Console). Začněte tím, že si zobrazíte nápovědu se všemi dostupnými příkazy.

Poté vytvořte první databázovou migraci, vhodně ji pojmenujte a aplikujte na databázi.

Vyzkoušejte si také smazání vzniklé databáze a odstranění poslední migrace.

Nakonec migraci znovu vytvořte a aplikujte.

Po dokončení tohoto úkolu by již aplikaci mělo být možné spustit. Ověřte, že lze projekt spustit v debug režimu, buď kliknutím na tlačítko „https“ se zelenou šipkou nebo klávesou F5. V novém okně prohlížeče by se měla zobrazit uvítací stránka jako je vidět na obrázku (Obr. 14).



Obrázek 25. Uvítací obrazovka aplikace pro úkoly

Zavřením okna prohlížeče aplikaci ukončíte.

4.5.2 Postup řešení

Otevřete si okno konzoly Správce balíčků z nabídky záhlaví okna Zobrazit → Ostatní okna → Konzola Správce balíčků. V anglické mutaci Visual Studia jej najdete pod názvem Package Manager Console.

Jako první spusťte příkaz `help entityframeworkcore`. Tento příkaz slouží pro vypsání nápovědy se všemi dostupnými příkazy tohoto nástroje. Také si tím ověříte, že jsou nástroje korektně nainstalovány. U příkazů nezáleží na velikosti písmen.

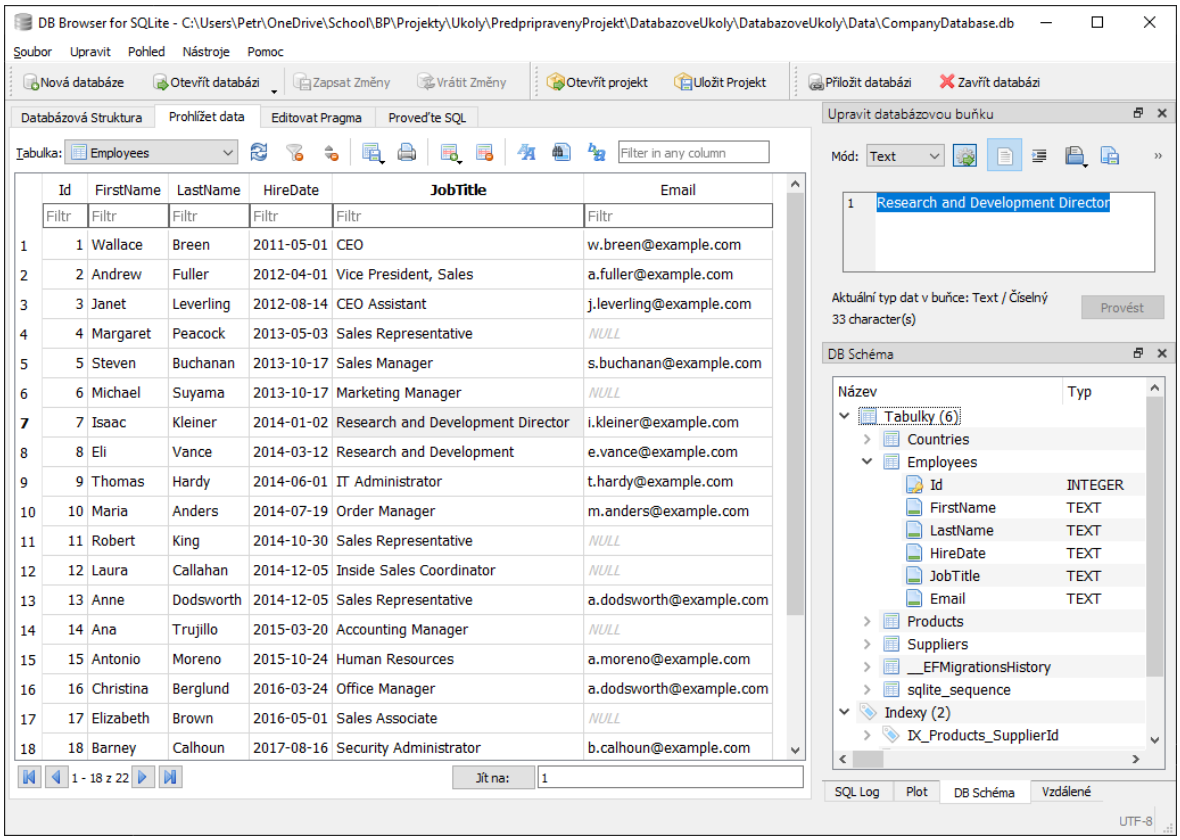
Pro vytvoření migrace aplikujte příkaz `Add-Migration` a migraci pojmenujte např. jako SQLite_1.0.0_Initial. Pojmenování lze napsat za příkaz v rámci jediného řádku, v opačném případě vás konzole k zadání názvu vyzve.

Po krátké prodlevě se zobrazí vygenerovaný soubor s databázovou migrací, ve kterém je možné zkontrolovat provedené změny. Tento vygenerovaný soubor by neměl být upravován napřímo, vlastní konfiguraci databáze byste namísto toho měli provádět s použitím fluent API uvnitř třídy pro databázový kontext, případně pomocí atributů uvnitř entit. A to zejména z důvodu, aby konfigurace na všech místech zůstala konzistentní. Každá taková změna poté vyžaduje vytvoření další migrace, aby mohla být na databázi aplikována.

Samotné vytvoření migrace ještě nemělo žádný přímý vliv na vaši databázi. Proto je možné v tento okamžik v případě chybných změn migraci snadno zrušit. To byste provedli příkazem `Remove-Migration` bez dalších parametrů.

Aplikaci změn na připojenou databázi provedete příkazem `Update-Database`. Nyní si zkontrolujte, že se databáze skutečně vytvořila v cestě uvedené v connection stringu, konkrétně Data/Db/CompanyDatabase.db.

Také si můžete zkontrolovat obsah databáze, např. pomocí programu DB Browser for SQLite, kde lze soubor s databází snadno otevřít. Měly by být vidět tabulky a přes možnost Prohlížet tabulku v kontextové nabídce u konkrétní tabulky také obsažená vzorová data. Tato data jsou předdefinována uvnitř třídy InitialData v rámci webové aplikace a v předcházejícím úkolu byla připojena k databázovému kontextu. Proto se automaticky nahrála do databáze jako součást prvotní migrace.



Obrázek 26. Obsah tabulky Employees ve vytvořené databázi

Po aplikaci poslední migrace na databázi již nelze tuto migraci odstranit pouhým příkazem `Remove-Migration`. Nejprve je nutné určit, ke které bývalé migraci se má struktura databáze navrátit. K tomu slouží již zmiňovaný příkaz `Update-Database`, tentokrát však s parametrem nesoucím název konkrétní migrace. Až poté půjde ta aktuální odstranit.

Samotnou databázi, včetně všech v ní uložených dat, lze kompletně smazat příkazem `Drop-Database`, vytvořené migrace však zůstanou nedotčeny. Tyto zůstávají součástí projektu aplikace, takže není nutné je znovu vytvářet. Po použití příkazu `Update-Database` budou všechny na databázi aplikovány.

4.5.3 Zdrojový kód řešení

Vytvoření první migrace s názvem SQLite_1.0.0_Init a její aplikace pro založení databáze:

```
Add-Migration SQLite_1.0.0_Init
Update-Database
```

4.5.4 Kontrolní otázky

- K čemu slouží databázové migrace?
- Záleží u příkazů konzoly Správce balíčků na velikosti písmen?
- Jaké kroky je potřeba provést, pokud potřebujete navrátit poslední migraci, která však již byla na databázi aplikována?
- Je doporučeno provádět změny v souboru s vygenerovanou migrací napřímo?
- Uveďte postup pro vytvoření nové migrace s názvem MySQL_1.2.5_Add-Orders a aplikaci této migrace na databázi.
- Uveďte příkaz pro smazání celé databáze.

4.6 Výběrové dotazy s LINQ

Úkoly zaměřené na výběr, filtrování a řazení dat jsou rozděleny podle databázových tabulek, ze kterých se data získávají. Každá tabulka obsahuje množství ukázkových dat. Tato data jsou předdefinována přímo uvnitř aplikace. Je vytvořeno celkem 19 výběrových dotazů.

Ve webové aplikaci se v každém view pro prezentaci dat nachází dvě tabulky. V první tabulce se zobrazují data načtená vlastním výběrovým dotazem. Ve druhé tabulce je vždy vidět výpis všech uložených dat v odpovídající databázové tabulce bez jakýchkoliv dalších úprav, jsou tedy seřazena od nejstaršího záznamu po nejnověji přidaný.

LINQ je název pro sadu technologií založených na integraci datového dotazování přímo do jazyka C#. [30]

Svůj LINQ kód, který budete psát dle požadavku každého úkolu, zapisujte vždy do metody Index uvnitř patřičného controlleru, který je pojmenován stejně jako související entita. Určené místo pro zápis je uvnitř metody označeno komentářem.

Parametry pro LINQ metody se zadávají ve formátu lambda výrazů. Data se získávají skrz DbSet properties z databázového kontextu.

Související data z entit připojených pomocí cizího klíče lze zahrnout pomocí metody Include, jejíž parametrem je odpovídající navigační property, přes kterou se k entitám přistupuje. Volání metody se vkládá na začátek LINQ dotazu jako jeho součást.

Voláním ToListAsync na konci dotazů se asynchronně spustí dotaz na data na základě specifikovaných pravidel. Výsledkem je seznam získaných dat převedený do podoby standardního .NET listu. Alternativou jsou metody First, Last, Single a jejich OrDefault varianty, které vrací vždy jeden jediný záznam.

Na data se lze dotazovat i pomocí standardních SQL dotazů s pomocí metody FromSql. Takto by například vypadal výběr všech zaměstnanců pomocí SQL dotazu uvnitř aplikace:

```
IList<Employee> AllEmployees = await dbContext.Employees.FromSql($"SELECT *
FROM Employees").ToListAsync();
```

4.6.1 Dotazy na tabulku Employees

Zadání 1: Vypište zaměstnance, jejichž křestní jméno končí na písmeno 'T'.

Řešení 1:

```
IList<Employee>? CustomEmployees = await dbContext.Employees
    .Where(e => e.FirstName.EndsWith("T"))
    .ToListAsync();
```

Zadání 2: Vypište zaměstnance, jejichž příjmení začíná na písmeno 'B', 'C', 'F' nebo 'H'. Seřaďte je sestupně dle datumu přijetí do pracovního poměru.

Řešení 2:

```
IList<Employee>? CustomEmployees = await dbContext.Employees
    .Where(e => e.LastName.StartsWith("B") || e.LastName.StartsWith("C") ||
e.LastName.StartsWith("F") || e.LastName.StartsWith("H"))
    .OrderByDescending(e => e.HireDate)
    .ToListAsync();
```

Zadání 3: Vypište zaměstnance, jejichž pracovní pozice obsahuje výraz 'Sales', ale zároveň neobsahuje výraz 'Representative'.

Řešení 3:

```
IList<Employee>? CustomEmployees = await dbContext.Employees
    .Where(e => e.JobTitle.Contains("Sales") &&
!e.JobTitle.Contains("Representative"))
    .ToListAsync();
```

Zadání 4: Vypište zaměstnance, u kterých není vyplněna e-mailová adresa.

Řešení 4:

```
IList<Employee>? CustomEmployees = await dbContext.Employees
    .Where(e => string.IsNullOrEmpty(e.Email))
    .ToListAsync();
```

Zadání 5: Vypište všechny zaměstnance seřazené abecedně dle jejich pracovní pozice.

Řešení 5:

```
IList<Employee>? CustomEmployees = await dbContext.Employees
    .OrderBy(e => e.JobTitle)
    .ToListAsync();
```

Zadání 6: Vypište všechny zaměstnance přijaté mezi lety 2014-2017, jejichž příjmení je delší než 6 písmen.

Řešení 6:

```
IList<Employee>? CustomEmployees = await dbContext.Employees
    .Where(e => e.HireDate >= DateOnly.Parse("1.1.2014") && e.HireDate <=
DateOnly.Parse("31.12.2017"))
    .Where(e => e.LastName.Length > 6)
    .ToListAsync();
```

4.6.2 Dotazy na tabulku Suppliers

Zadání 7: Vyberte všechny dodavatele, seřaďte je dle názvu země a města podle abecedy.

Řešení 7:

```
IList<Supplier>? CustomSuppliers = await dbContext.Suppliers
    .Include(s => s.Country)
    .OrderBy(s => s.Country.Name)
    .ThenBy(s => s.City)
    .ToListAsync();
```

Zadání 8: Vyberte dodavatele, u kterých je nastaven PostalCode a sestává přesně ze 4 znaků. Výpis seřaďte podle kódu země sestupně.

Řešení 8:

```
IList<Supplier>? CustomSuppliers = await dbContext.Suppliers
    .Include(s => s.Country)
    .Where(s => s.PostalCode != null && s.PostalCode.Length == 4)
    .OrderByDescending(s => s.Country.Code)
```

```
    .ToListAsync();
```

Zadání 9: Vyberte dodavatele s neúplnými informacemi o jejich adrese.

Řešení 9:

V tomto případě je nutné použít pouze jeden Where výraz. Pokud byste totiž kontrolovali každou property v samostatném Where výrazu, výsledek by fungoval jako „a zároveň".

Druhou možností, jak zjistit nenastavenou zemi, by byla kontrola na hodnotu 0 v property pro cizí klíč s.CountryId.

```
IList<Supplier>? CustomSuppliers = await dbContext.Suppliers
    .Include(s => s.Country)
    .Where(s => string.IsNullOrWhiteSpace(s.Street) ||
string.IsNullOrWhiteSpace(s.City) || string.IsNullOrWhiteSpace(s.PostalCode)
|| s.Country == null)
    .ToListAsync();
```

Zadání 10: Vyberte dodavatele, kteří nejsou spojeni s žádným produktem v tabulce Products.

Řešení 10:

```
IList<Supplier>? CustomSuppliers = await dbContext.Suppliers
    .Include(s => s.Country)
    .Include(s => s.Products)
    .Where(s => !s.Products.Any())
    .ToListAsync();
```

Zadání 11: Vyberte dodavatele, kteří jsou nastaveni jako dodavatelé pro alespoň 2 produkty, a seřaďte je podle počtu produktů sestupně.

Řešení 11:

```
IList<Supplier>? CustomSuppliers = await dbContext.Suppliers
    .Include(s => s.Country)
    .Include(s => s.Products)
    .Where(s => s.Products.Count() >= 2)
    .OrderByDescending(s => s.Products.Count)
    .ToListAsync();
```

Zadání 12: Vyberte pouze ty dodavatele, jejichž název je stejně dlouhý jako adresa ulice.

Řešení 12:

```
IList<Supplier>? CustomSuppliers = await dbContext.Suppliers
    .Include(s => s.Country)
    .Where(s => s.Name.Length == s.Street.Length)
    .ToListAsync();
```

4.6.3 Dotazy na tabulku Products

Zadání 13: Najděte všechny produkty, jejichž cena za kus není prázdná hodnota a je vyšší než 10. Výpis seřaďte sestupně od nejvyšší ceny.

Řešení 13:

```
IList<Product>? CustomProducts = await dbContext.Products
    .Include(p => p.Supplier)
    .Where(p => p.Price != null && p.Price > 10)
    .OrderByDescending(p => p.Price)
    .ToListAsync();
```

Zadání 14: Najděte 8 nejnovějších produktů.

Řešení 14:

```
IList<Product>? CustomProducts = await dbContext.Products
    .Include(p => p.Supplier)
    .OrderByDescending(p => p.CreatedOn)
    .Take(8)
    .ToListAsync();
```

Zadání 15: Najděte produkty, jejichž cena za kus je mezi 12 až 33.5 a na skladě se jich nachází více, než 40 kusů. Seřaďte je od nejstaršího podle datumu a času přidání do databáze.

Řešení 15:

```
IList<Product>? CustomProducts = await dbContext.Products
    .Include(p => p.Supplier)
    .Where(p => p.Price >= 12 &&  p.Price <= 33.5)
    .Where(p => p.StockUnits > 40)
    .OrderBy(p => p.CreatedOn)
    .ToListAsync();
```

Zadání 16: Najděte produkty, jejichž množství kusů na skladě je menší než 10 a byly přidány od července roku 2019. Seřaďte je dle počtu kusů sestupně.

Řešení 16:

```
IList<Product>? CustomProducts = await dbContext.Products
    .Include(p => p.Supplier)
    .Where(p => p.StockUnits < 10)
    .Where(p => p.CreatedOn >= DateTime.Parse("1.7.2019"))
    .OrderByDescending(p => p.StockUnits)
    .ToListAsync();
```

Zadání 17: Najděte produkty, jejichž dodavatel je z Německa (DE), Francie (FR) nebo Itálie (IT). Pro filtrování použijte dvoumístný standardní kód dané země. Srovnejte produkty podle názvu.

Řešení 17:

Bude nutné specifikovat další související data z již zahrnuté entity Supplier. Pro tento účel slouží výraz ThenInclude. Pomocí ThenInclude lze přistoupit k navigační property z entity, která byla právě načtena předchozím Include výrazem.

```
IList<Product>? CustomProducts = await dbContext.Products
    .Include(p => p.Supplier)
    .ThenInclude(s => s.Country)
    .Where(p => p.Supplier.Country.Code == "DE" || p.Supplier.Country.Code ==
"FR" || p.Supplier.Country.Code == "IT")
    .OrderBy(p => p.Name)
    .ToListAsync();
```

Zadání 18: Najděte produkty, jejichž název je složen z více slov oddělených mezerou a byly přidány mezi daty 1. září 2016 až 15. prosince 2018.

Řešení 18:

```
IList<Product>? CustomProducts = await dbContext.Products
    .Include(p => p.Supplier)
    .Where(p => p.Name.Contains(" "))
    .Where(p => p.CreatedOn >= DateTime.Parse("1.9.2016") && p.CreatedOn <=
DateTime.Parse("15.12.2018"))
    .ToListAsync();
```

Zadání 19: Zjistěte celkové množství kusů všech produktů na skladě a jejich průměrnou, minimální a maximální cenu.

Řešení 19:

Cílem tohoto úkolu je získat čtyři číselné hodnoty, data tedy budou v jiném formátu než doposud.

Nejprve napište své LINQ dotazy pro získání požadovaných dat dle zadání úkolu. Nenahrazujte načítání dat do proměnné CustomProducts, nové dotazy napište samostatně pod tuto část a získaná data přiřaďte do nových proměnných:

- `int totalUnitsInStock`
- `double? avgPriceInStock`
- `double? minPriceInStock`
- `double? maxPriceInStock`

Tyto proměnné přidejte do modelu typu Tuple, který data předává do view, následujícím způsobem. Nahraďte prázdné hodnoty null proměnnými s daty ve správném pořadí:

```
// Předání načtených dat z databáze do View jako model typu Tuple.
(IList<Product> AllData, IList<Product>? CustomQueryData, int? totalUnits,
double? avgPrice, double? minPrice, double? maxPrice) TupleDataForView =
    (AllProducts, CustomProducts, totalUnitsInStock, avgPriceInStock,
minPriceInStock, maxPriceInStock);
```

Data by se nyní měla správně zobrazovat ve view pod tabulkou produktů.

LINQ dotazy pro získání požadovaných dat:

```
int totalUnitsInStock = await dbContext.Products
    .SumAsync(p => p.StockUnits);

double? avgPriceInStock = await dbContext.Products
    .AverageAsync(p => p.Price);

double? minPriceInStock = await dbContext.Products
    .MinAsync(p => p.Price);

double? maxPriceInStock = await dbContext.Products
    .MaxAsync(p => p.Price);
```

4.6.4 Kontrolní otázky

- K čemu slouží LINQ?
- Je možné snadno řetězit více podmínek pro výběr dat v jednom LINQ dotazu?
- Do jaké LINQ metody se obvykle zadává vlastní podmínka pro výběr dat?
- Jaký název má LINQ metoda pro seřazení dat sestupně?
- Kterou asynchronní LINQ metodu byste použili pro výpočet průměrné hodnoty ze všech údajů jednoho sloupce?

4.7 Vytváření nových dat

4.7.1 Zadání úkolu

Přidejte funkcionalitu pro vytváření a ukládání nových dat do databázových tabulek. Kód se bude nacházet uvnitř metody HttpPost Create v jednotlivých controllerech. Otestujte funkčnost přidáním nových záznamů do tabulek přes tlačítko a formulář přímo v aplikaci. Konkrétní hodnoty, jako je např. jméno zaměstnance, si můžete vymyslet vlastní. Zkontrolujte, že vám systém nedovolí přidat nový záznam, pokud nevyplníte všechny povinné údaje, a také, že se nový záznam zobrazuje ve výpisu dat z tabulek.

4.7.2 Postup řešení

Metoda Create s atributem HttpPost přijímá a zpracovává data po odeslání formuláře pro vytvoření nového záznamu.

Podmínka ModelState.IsValid v ASP.NET Core aplikacích na straně serveru kontroluje, zda předávané hodnoty v modelu jsou pro každou property validní. Příkladem by mohla být kontrola, zda je vyplněno jméno a příjmení u zaměstnance nebo zda je zadaná cena za kus větší než 0. Pokud by podmínky nebyly splněny, data nebudou uložena. Ve webové aplikaci, kterou pro úkoly používáte, zůstane formulář otevřený a zobrazí se validační chyby. Tyto podmínky lze nastavit pomocí validačních atributů pro jednotlivé properties uvnitř entit.

Pokud s ukládanými daty nepotřebujete nikterak dále manipulovat, postačí, když je v rámci kódu přidáte do DbSet kolekce pro správnou databázovou tabulku a uložíte provedené změny do databáze. Pokud by součástí formuláře byla možnost např. nahrát soubor s obrázkem na server, bylo by třeba tento úkon zajistit jako součást Create metody ještě před samotným uložením databázových dat.

4.7.3 Zdrojový kód řešení

Jako ukázka implementace řešení je vyobrazen kód pro EmployeeController. Implementace pro dalších controllery bude srovnatelná.

```
[HttpPost]
public async Task<IActionResult> Create(Employee employee)
{
    if (ModelState.IsValid)
    {
        dbContext.Employees.Add(employee);

        await dbContext.SaveChangesAsync();

        return RedirectToAction(nameof(Index));
    }
    else
    {
        return View(employee);
    }
}
```

4.7.4 Kontrolní otázky

- Pokud máte v proměnné instanci entity, kterou chcete uložit jako nový záznam do databáze, co pro to musíte udělat?
- Jak se nazývá asynchronní metoda, kterou uložíte všechny provedené změny do databáze?

4.8 Úpravy existujících dat

4.8.1 Zadání úkolu

Přidejte funkcionalitu pro změnu existujících dat v databázi. Inspirujte se formulářem v aplikaci pro úpravu databázových záznamů pro jednotlivé tabulky, napoví vám, jaká data má být možné při úpravě záznamu dané entity přepisovat. Data se po odeslání formuláře předávají do metody HttpPost Edit v každém controlleru.

4.8.2 Postup řešení

Podobně jako při vytváření nový dat se nejprve kontroluje validita odeslaných dat z formuláře. Pokud jsou validní, je třeba na základě primárního klíče vyhledat a načíst aktuálně upravovaný záznam z databáze. K tomu poslouží metoda Find, příp. její asynchronní varianta FindAsync s parametrem primárního klíče.

Pokud byl záznam nalezen, je vaším úkolem přenastavit jeho hodnoty, resp. property na nové hodnoty získané z formuláře. Ne všechny property by však měly být manuálně upravitelné. Např. datum a čas vytvoření produktu může být nastavován pouze automaticky.

4.8.3 Zdrojový kód řešení

Příklad implementace funkcionality v HttpPost Edit metodě pro ProductController:

```
[HttpPost]
public async Task<IActionResult> Edit(Product product)
{
    if (ModelState.IsValid)
    {
        Product? editedProduct = await
dbContext.Products.FindAsync(product.Id);
        if (editedProduct != null)
        {
            editedProduct.Name = product.Name;
            editedProduct.StockUnits = product.StockUnits;
            editedProduct.Price = product.Price;
            editedProduct.SupplierId = product.SupplierId;

            await dbContext.SaveChangesAsync();

            return RedirectToAction(nameof(Index));
        }
        else
        {
            return NotFound();
        }
    }
    else
    {
        // Příprava dat pro select list s výběrem dodavatele
```

```
        ViewBag.Suppliers = await dbContext.Suppliers.ToListAsync();

        return View(product);
    }
}
```

4.8.4 Kontrolní otázky

- Která asynchronní metoda poskytuje funkci snadného nalezení a načtení konkrétního záznamu na základě známého primárního klíče?
- Je nutné při provádění úprav přepsat všechna dosavadní data v upravované entitě?

4.9 Odstraňování dat

4.9.1 Zadání úkolu

Poslední z CRUD (Create, Read, Update, Delete) operací, jejíž implementace zatím chybí, je možnost data mazat. Pro tuto funkci jsou připraveny metody s názvem Delete, taktéž ve všech třech controllerech. Zatím v nich však chybí kód, aby byly schopny tento úkon provádět. Vaším úkolem je nyní tento kód doplnit.

4.9.2 Postup řešení

Oproti vytváření a úpravám dat se nemusí kontrolovat validita, jelikož se nepředávají žádná nová data. Je však potřeba získat záznam určený ke smazání, k čemuž se opět použije metoda Find nebo FindAsync s parametrem primárního klíče. Pokud záznam pro daný klíč existuje, metodou Remove jej ze správného DbSetu smažte a uložte změny do databáze.

4.9.3 Zdrojový kód řešení

Příklad kódu ze SupplierController Delete metody:

```
public async Task<IActionResult> Delete(int id)
{
    Supplier? deletedSupplier = await dbContext.Suppliers.FindAsync(id);
    if (deletedSupplier != null)
    {
        dbContext.Suppliers.Remove(deletedSupplier);
        await dbContext.SaveChangesAsync();

        return RedirectToAction(nameof(Index));
    }
    else
    {
        return NotFound();
    }
}
```

4.9.4 Kontrolní otázky

- Je po odstranění záznamu metodou Remove tento záznam také okamžitě smazán z databáze?

4.10 Změna struktury a přidání validací pomocí atributů

4.10.1 Zadání úkolu

V entitě Product pomocí atributů pro anotaci dat proveďte tyto změny:

- Název produktu může mít maximální délku textového řetězce 255 znaků.
- Přidejte novou property s libovolným datovým typem s možnou hodnotou null (označen symbolem ‚?') a libovolným názvem. Nastavte ji však tak, aby se pro ni v databázi nevytvořil sloupec a neukládala se tam její hodnota.
  Null hodnoty je nutné povolit, aby nebyl problém s odesíláním formuláře pro přidání nebo úpravu produktu v aplikaci, jelikož tato hodnota nebude vyplněna.
- Zajistěte, že hodnota ceny produktu nesmí být méně než 0.
- Přejmenujte název databázového sloupce pro property StockUnits v entitě Product z výchozího názvu převzatého na základě jména property. Název samotné property však neměňte.

Nezapomeňte vytvořit novou migraci a aplikovat tak provedené změny na databázi. V programu DB Browser for SQLite můžete zkontrolovat, zda struktura databáze skutečně odpovídá požadavkům dle zadání tohoto úkolu.

4.10.2 Postup řešení

Atributy se zapisují nad property do hranatých závorek. Případné parametry poté do kulatých závorek za název obdobně jako při volání metody.

První část úkolu je typický příklad pro využití atributu pro anotaci dat `StringLength`. Kromě validace při odeslání dat také způsobí změnu datového typu odpovídajícího sloupce v databázi. Použitý SQLite nerozlišuje maximální délku textového řetězce v datovém typu, nicméně kdybyste použili např. MySQL server, ve struktuře databáze byste mohli vidět změnu typu na varchar s omezením délky podle zadaného parametru pro `StringLength` atribut. [31]

V druhé části si pomocí atributu `NotMapped` vyzkoušejte, že lze konkrétní property označit tak, aby ji EF Core ignoroval ve vztahu k databázovému systému přesto, že je součástí .NET třídy pro entitu.

Povolený rozsah pro cenu produktu jednoznačně zajistíte atributem `Range`, který přebírá dva parametry: minimum a maximum. Toto nastavení se následně použije pro kontrolu nových dat z odesílaných formulářů v aplikaci v rámci validace, aby nedošlo k uložení nepovolené hodnoty do databáze.

Poslední požadavek úkolu bude mít narozdíl od předchozího přímý vliv na strukturu databáze. Atribut `Column` umožňuje zadání vlastního názvu pro odpovídající sloupec v tabulce databáze, který se tak může lišit oproti názvu property, se kterou je pomocí EF Core propojen.

4.10.3 Zdrojový kód řešení

Takto by měl vypadat obsah entity Product po provedení všech změn:

```
public class Product
{
    public int Id { get; set; }

    [StringLength(255)]
    public string Name { get; set; }

    [NotMapped]
    public string? MyPropertyNotInDatabase { get; set; }

    [Column("UnitsInStock")]
    public int StockUnits { get; set; }

    [Range(0, double.MaxValue)]
    public double? Price { get; set; }
    public DateTime CreatedOn { get; set; }

    public int? SupplierId { get; set; }
    public Supplier? Supplier { get; set; }
}
```

4.10.4 Kontrolní otázky

- K čemu slouží atributy pro anotaci dat? Jmenujte 3 příklady využití.
- Jakým atributem nastavíte datový typ varchar(255) pro sloupec v podporovaných databázových systémech?

- Sloupec v databázi je pojmenován na základě názvu jemu odpovídající DbSet property. Je možné toto pravidlo změnit pomocí atributu? Pokud ano, uveďte, jakým způsobem.
- Pro každou property, která je součástí určité entity, je vždy vytvořen odpovídající databázový sloupec a toto chování nelze změnit bez toho, aniž bychom danou property smazali. Je toto tvrzení pravdivé?

4.11 Úpravy pomocí fluent API

4.11.1 Zadání úkolu

Nastavte sloupec CreatedOn pro produkt tak, aby se do ní při ukládání nových produktů do databáze automaticky uložil aktuální datum a čas na straně databázového systému.

Dále nastavte název dodavatele jako unikátní tak, aby databázový systém nepovolil vícenásobné uložení dodavatele se stejným názvem.

Obě tato nastavení proveďte uvnitř metody OnModelCreating ve vašem databázovém kontextu s využitím fluent API.

Po vytvoření a aplikování migrace v aplikaci otestujte vytvoření nového produktu a zkontrolujte, zda se u něj skutečně nastavil aktuální datum a čas.

Také se pokuste přidat nového dodavatele s přesně stejným názvem, jaký už existuje. Tento pokus by měl selhat s výjimkou ve znění: „*SqliteException: SQLite Error 19: 'UNIQUE constraint failed: Suppliers.Name'*.“.

4.11.2 Postup řešení

Spoustu nastavení ovlivňující databázový systém a chování EF Core je možné provádět dvěma způsoby, buď prostřednictvím atributů pro anotaci dat nebo pomocí fluent API. Existují však případy, kdy pro určité nastavení atribut neexistuje, případně styl zápisu s fluent API jednoduše preferujete. Také z těchto důvodů si práci s tímto rozhraním nyní vyzkoušíte na dvou přímočarých úkolech.

Nastavení pomocí fluent API se zapisuje do databázového kontextu, konkrétně dovnitř jeho metody OnModelCreating, kterou jste si překryli v rámci úkolu pro vytvoření vlastního kontextu.

Pro nastavení automatického ukládání aktuálního data a času do sloupce CreatedOn v produktu při vytvoření nového záznamu použijte funkci datetime() z databázového systému SQLite. V zápisu pomocí fluent API musíte zřetězeným voláním metod přes parametr modelBuilder vybrat entitu Product a poté její property CreatedOn, pro kterou zavoláte metodu HasDefaultValueSql s parametrem zmíněné funkce zapsané jako text.

Sloupec Name v tabulce Supplier nastavíte jako unikátní metodou HasIndex s parametrem odpovídající property. Pro tento index pak zavoláte metodu IsUnique.

4.11.3 Zdrojový kód řešení

Implementace obou vlastností pomocí fluent API by nakonec měla vypadat takto:

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Supplier>()
        .HasData(InitialData.GetSuppliers());
    // … zkraceny vypis inicializace pocatecnich dat pro entity

    modelBuilder.Entity<Product>()
        .Property(p => p.CreatedOn)
        .HasDefaultValueSql("datetime()");

    modelBuilder.Entity<Supplier>()
        .HasIndex(s => s.Name)
        .IsUnique();

    base.OnModelCreating(modelBuilder);
}
```

4.11.4 Kontrolní otázky

- Do které metody databázového kontextu se zapisují nastavení provedená pomocí fluent API?
- Jsou SQL funkce, které lze zadat jako parametr pro metodu HasDefaultValueSql, nezávislé na typu připojeného databázového systému?
- Je konfigurace pomocí fluent API vždy jediným způsobem, jakým lze danou změnu nastavit?
- Jak se změní chování sloupce databáze po aplikaci metody IsUnique na odpovídající property?

4.12 Možnosti dalšího rozvoje úkolů

V této části jsou uvedeny návrhy pro vylepšení, doplnění a vytvoření dalších úkolů.

- Složitější výběrové dotazy, např. s využitím GroupBy.
- Rozšíření entity Product o vztah many-to-many s entitou Supplier pro možnost propojení více dodavatelů s jedním produktem.
- Propojení také s jinými databázovými systémy kromě použitého SQLite a otestování případných odlišností.
- Aktualizace úkolů v závislosti na případných nových funkcích EF Core nebo .NET v budoucích aktualizacích.
- Rozšíření úkolů pro další databázové .NET technologie a frameworky.

5 VÝUKOVÁ APLIKACE

Pro podporu navržených úkolů a přehlednou práci s nimi byla vytvořena výuková aplikace s názvem Výukový Portál. Tato aplikace také slouží jako praktická ukázka využití Entity Frameworku Core pro propojení .NET aplikace s databázovým serverem MySQL.

Tato aplikace je také hostována online ve službě Microsoft Azure a je dostupná na tomto odkazu: https://vyukovyportalweb2024.azurewebsites.net

5.1 Návrh aplikace

Funkční a nefunkční požadavky se odvíjejí zejména od účelu aplikace a formátu vytvořených úkolů.

5.1.1 Funkční požadavky

Pojem správa použitý v některých požadavcích hromadně zahrnuje funkce zobrazení, vytvoření, úpravy a odstranění dané položky.

- RQ001: Systém musí umožnit správu témat.
- RQ002: Systém musí umožnit správu kapitol pro dané téma.
- RQ003: Systém musí umožnit správu výukového obsahu pro danou kapitolu.
- RQ004: Systém musí umožnit správu testových otázek pro danou kapitolu.
- RQ005: Systém musí umožnit přihlášení uživatele.
- RQ006: Systém musí umožnit vytvoření nového uživatelského účtu.
- RQ007: Systém musí umožnit změnu role uživatelského účtu.
- RQ008: Systém musí umožnit kontrolu odpovědi na testovou otázku.
- RQ009: Systém musí umožnit kontrolu a uložení výsledku celého testu.
- RQ010: Systém musí umožnit zobrazení odeslaných testů.

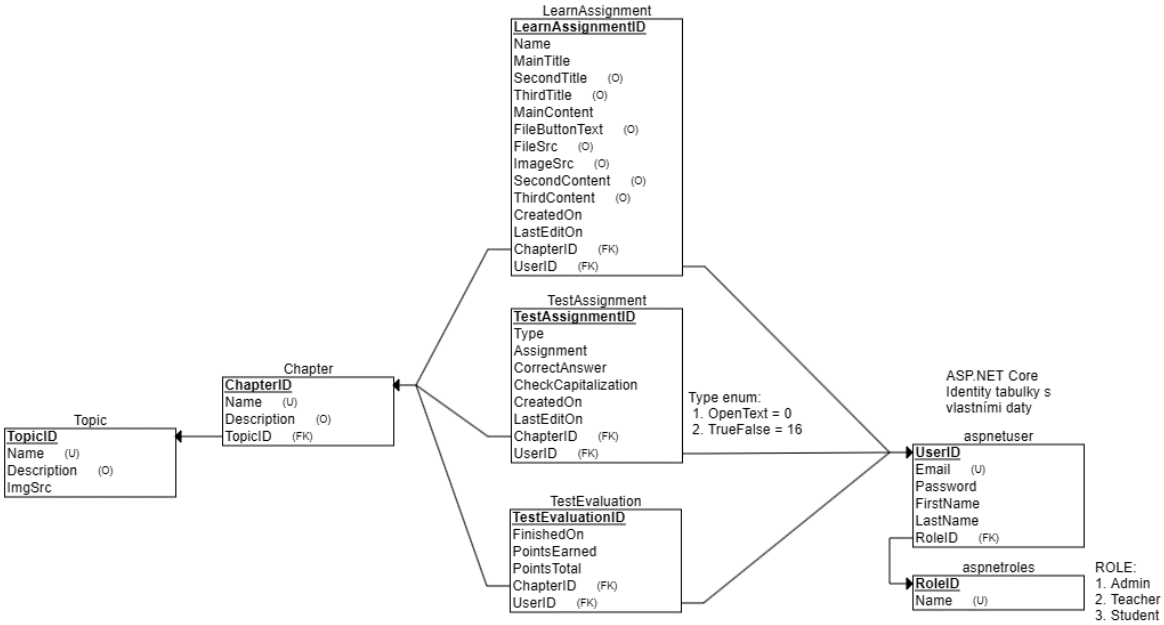
5.1.2 Nefunkční požadavky

- RQ011: Uživatelské rozhraní musí mít responzivní design pro podporu mobilních zařízení.
- RQ012: Systém pro stejný uživatelský účet uloží výsledek odeslaného testu pouze jednou za 5 minut.
- RQ013: Změny uložených dat musí provádět pouze autorizovaný uživatel na základě přiřazené role.

- RQ014: Hesla uživatelských účtů musí splňovat minimální požadavky: délka alespoň 6 znaků, minimálně 4 různé znaky, musí obsahovat malé písmeno a číslici.

5.1.3 Relační schéma databáze

Na základě vyhodnocených požadavků bylo vytvořeno relační schéma databáze, se kterou bude aplikace propojena pro dlouhodobé ukládání dat.



Obrázek 27. Relační schéma databáze výukové aplikace

5.2 Využité technologie

Veškerý vývoj probíhal v prostředí IDE Microsoft Visual Studio 2022 ve verzi Community.

5.2.1 Na straně serveru

Aplikace byla vytvořena v technologii ASP.NET Core na platformě .NET 8 pomocí architektonického vzoru MVC (Model-View-Controller).

Pro účely dlouhodobého ukládání dat je připojena k MySQL databázi s využitím Entity Framework Core. Pro podporu uživatelských účtů a možnost nastavení odlišných úrovní oprávnění pomocí rolí byl použit ASP.NET Core Identity.

5.2.2 Na straně klienta

Pro návrh a tvorbu uživatelského rozhraní byl zvolen Bootstrap framework ve verzi 5.3.3. Bootstrap je vhodnou volbou zejména z důvodů jeho snadného použití a možností vlastních úprav, kvalitní a srozumitelné dokumentace a dostatečného výběru komponent pro potřeby této aplikace.

Funkcionalita pro stránku testových úkolů včetně předávání dat mezi klientem a serverem, pro předávání dat do modal dialogů a možnosti jejich vyvolání, a pro nastavení odkazu v navigačním panelu jako aktivního na základě právě zobrazené stránky, je implementována pomocí Javascriptu.

5.3 Struktura kódu aplikace

Pro přehlednější uspořádání kódu a zamezení jeho nadbytečnému opakování je aplikace členěna do 4 projektů v rámci jednoho Visual Studio řešení „VyukovyPortal“.

5.3.1 Web

Toto je základní ASP.NET Core projekt se všemi views a controllers, které komunikují s databází skrze služby předané v konstruktoru pomocí dependency injection.

Projekt dále obsahuje složku wwwroot, ve které jsou uloženy všechny CSS a Javascript soubory, jsou zde nainstalovány front-end frameworky Bootstrap a jQuery, a také se zde ukládají soubory nahrané uživateli portálu na server do samostatných složek pro ně připravených.

Má závislost na všech třech ostatních projektech.

5.3.2 Domain

Zde se nachází třídy pro definici entit, jež jsou předpisem pro tabulky v databázi.

Nemá závislost na žádném dalším projektu.

5.3.3 Infrastructure

V projektu Infrastructure je nainstalován EF Core. Projekt zajišťuje konfiguraci databáze pomocí PortalDbContext, obsahuje vytvořené migrace a vzorová data pro prvotní seeding databáze.

Dále je zde nainstalována knihovna Identity pro podporu uživatelských účtů a řízení oprávnění přístupu v aplikaci.

Má závislost na projektu Domain.

5.3.4 Application

Obsahuje rozhraní i implementaci všech služeb (services) pro dependency injection. Nejčastěji obsahují CRUD operace s databází. Tyto služby se často volají z controller metod pro získávání a úpravu dat v databázi na základě přijatého požadavku od klienta.

To vede k logickému oddělení závislostí, kdy žádný controller nepoužívá PortalDbContext napřímo. Dále zabraňuje neustálému opakování stejného či velmi podobného kódu v jednotlivých controller metodách a tím přispívá k menšímu množství kódu a celkově větší přehlednosti napříč řešením.

Tento projekt má závislosti na projektech Domain a Infrastructure.

5.4 Funkce a uživatelské rozhraní aplikace

5.4.1 Uživatelské role pro řízení oprávnění přístupu

Některé části portálu jsou přístupné i bez uživatelského účtu, u jiných existuje omezení přístupu jen pro přihlášené účty na základě jejich členství v rolích. V aplikaci jsou používány role Student, Teacher a Admin.

Přístup do výukové části je umožněn i bez uživatelského účtu. Všichni přihlášení uživatelé s rolí Student mohou vyplňovat a odesílat testy, a prohlížet výsledky vlastních testů.

Uživatelé s rolí Teacher mohou vytvářet, měnit a odstraňovat veškerý obsah portálu s výjimkou již odeslaných testů studenty. Také mají možnost prohlížet odeslané výsledky testů od všech studentů.

Role Admin má možnost zobrazovat údaje o všech existujících uživatelských účtech a měnit jejich členství v roli Teacher.

Aplikace na neoprávněný pokus o přístup do omezené části reaguje dvěma způsoby. Pokud žádný účet není přihlášen, přesměruje uživatele na přihlašovací formulář. V případě, že aktuální účet není členem požadované role, se zobrazí speciální stránka informující o nedostatečném oprávnění. K tomu by mohlo dojít např. použitím přímé URL adresy, server však takovou žádost odmítne.

5.4.2 Společné prvky uživatelského rozhraní

5.4.2.1 Záhlaví a zápatí

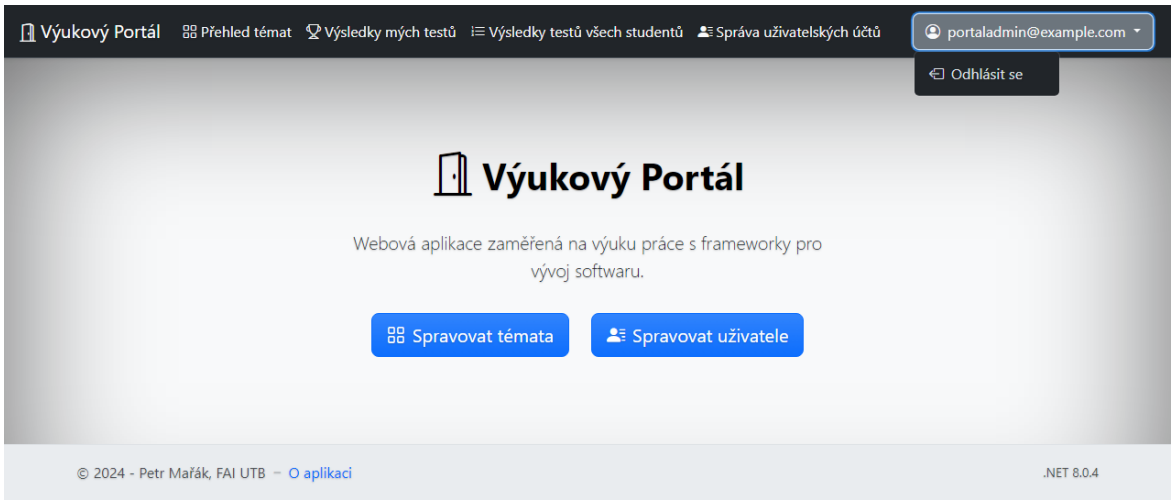
Všechny stránky aplikace obsahují společné záhlaví a zápatí, jejichž vzhled a obsah jsou definovány v souboru _Layout.

Na začátku záhlaví se nachází název a symbol portálu. Pootevřené dveře symbolizují příležitost pro nabytí nových znalostí.

Následuje navigační panel s odkazy pro přístup do různých částí portálu. Každému uživateli se zobrazí pouze ty části, do kterých má oprávnění přistoupit na základě svého členství v rolích. Pokud odkaz v panelu odpovídá aktuálně zobrazené stránce, je vizuálně zvýrazněn.

Tlačítku pro uživatelský účet je vyhrazen prostor na konci záhlaví.

Na obrázku (Obr. 24) je ukázána domovská stránka portálu s viditelným záhlavím a zápatím a otevřeným dropdown menu pro tlačítko uživatelského účtu.



Obrázek 28. Domovská stránka portálu se záhlavím a zápatím

5.4.2.2 Navigační pruh a úvodní sekce

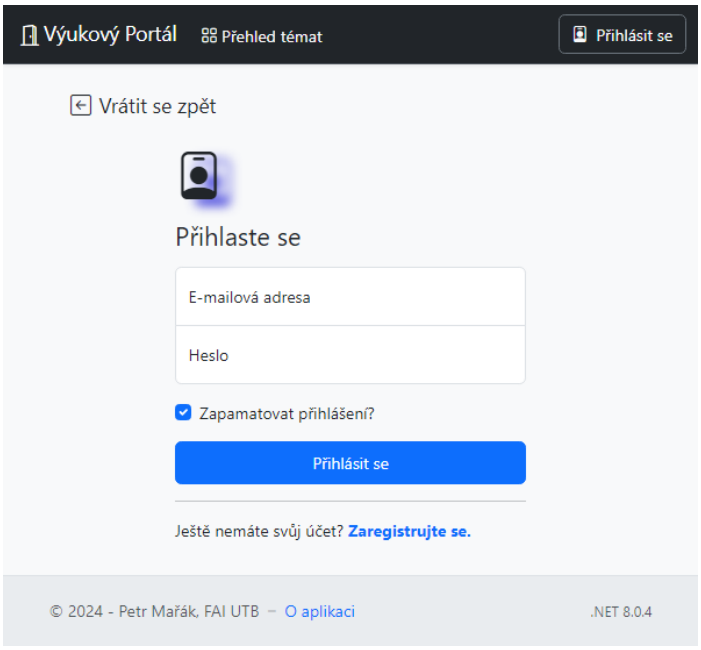
Napříč portálem je využit navigační pruh pro vyjádření současné pozice v rámci hierarchie stránek pro snazší orientaci a možnost návratu k předcházejícím stránkám. Spolu s tím je také použit jednotný design úvodní sekce stránek, zahrnující nadpis a podnadpis.

Oba tyto prvky jsou realizovány jako partial view a vkládány do jednotlivých views. Data jsou nastavována v controllerech a předávána pomocí dynamické ViewBag property do partial views.

5.4.3 Přihlášení a registrace uživatele

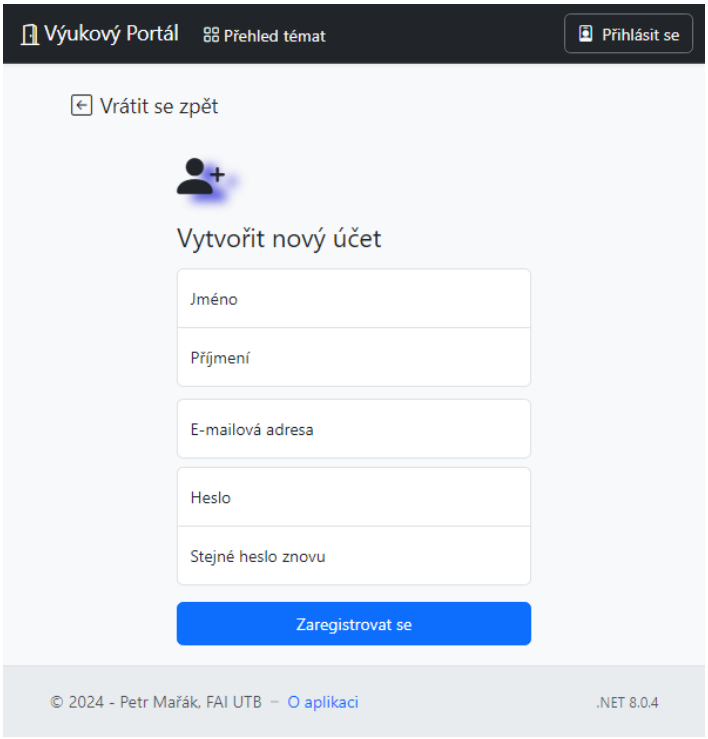
Uživatelé se ke svým účtům přihlašují pomocí e-mailové adresy a hesla zadaných při registraci. Formulář pro přihlášení lze otevřít tlačítkem Přihlásit se v pravé části záhlaví.

Po přihlášení je aktuální uživatelské jméno, odpovídající e-mailové adrese účtu, indikováno v záhlaví na místě původního tlačítka. Po kliknutí je možné zvolit odhlášení.



Obrázek 29. Přihlášení uživatele

Ze stránky pro přihlášení lze také pomocí odkazu „Zaregistrujte se“ otevřít druhý formulář pro vytvoření nového účtu. Po provedení registrace je portál k tomuto účtu automaticky přihlášen.

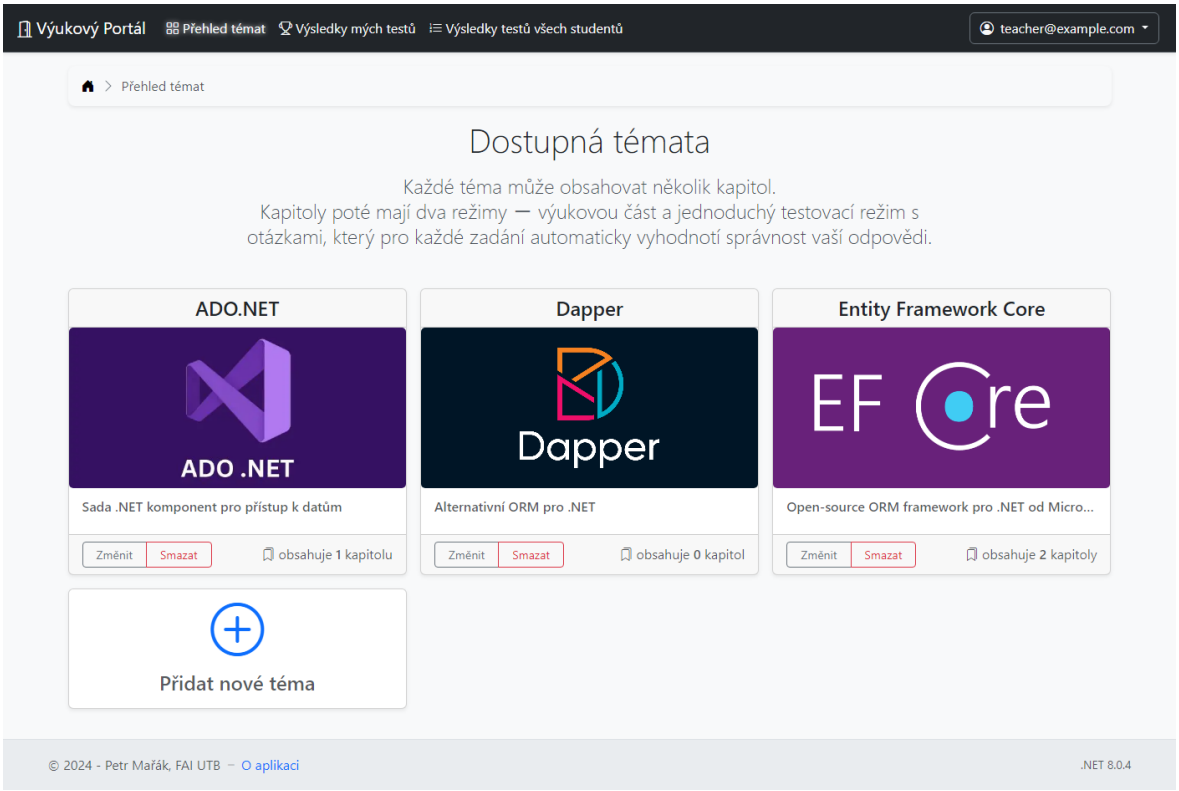


Obrázek 30. Registrace nového uživatele

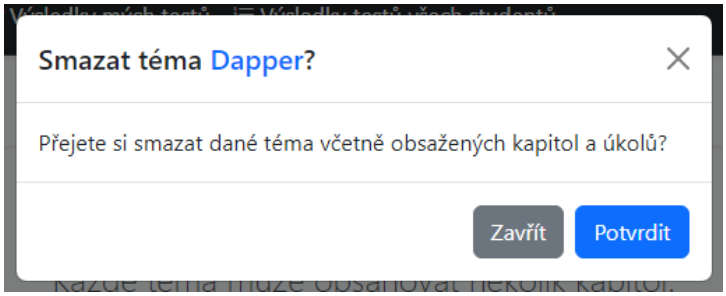
5.4.4 Témata

Témata představují nejvyšší úroveň členění obsahu. Pro vyobrazení témat jsou použity Bootstrap karty s vlastními úpravami. Témata se zobrazují v mřížce a na posledním místě se vždy nachází karta pro přidání nového tématu.

Tlačítka Změnit, Smazat u jednotlivých témat a karta Přidat nové téma jsou viditelná pouze v případě, že aktuální účet je členem role Teacher. Po stisku tlačítka pro smazání se zobrazí modal dialog pro potvrzení nebo zrušení této akce.



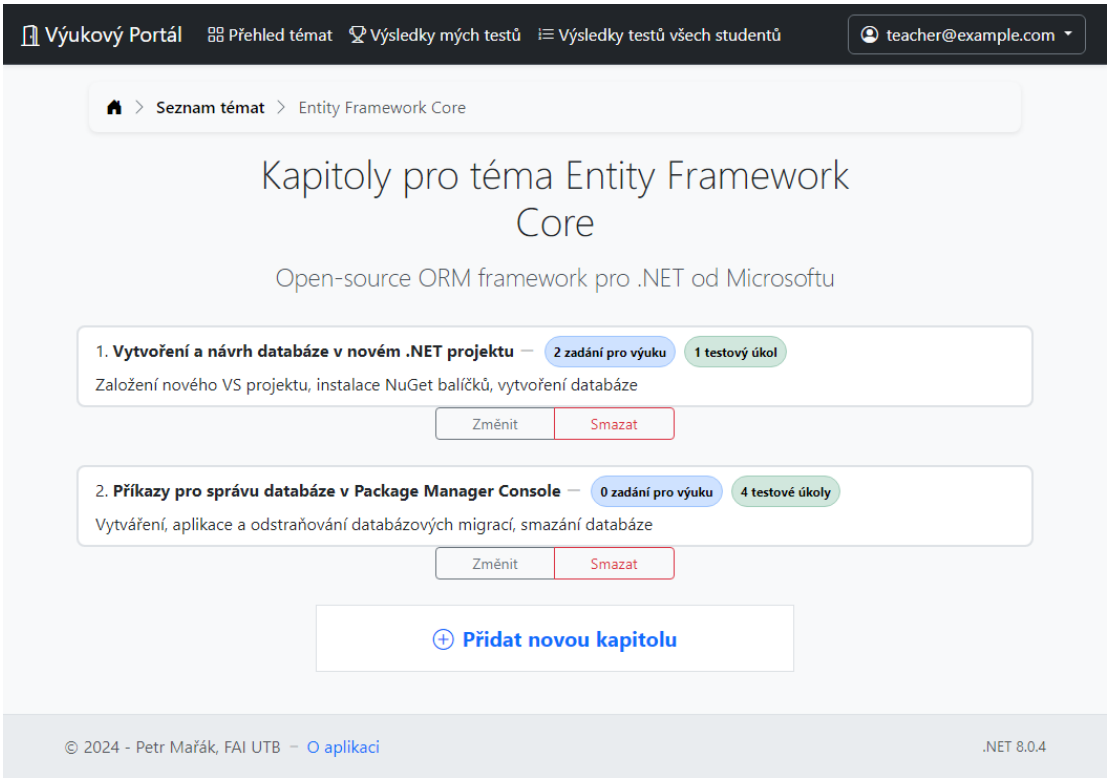
Obrázek 31. Přehled témat



Obrázek 32. Modal dialog pro potvrzení smazání

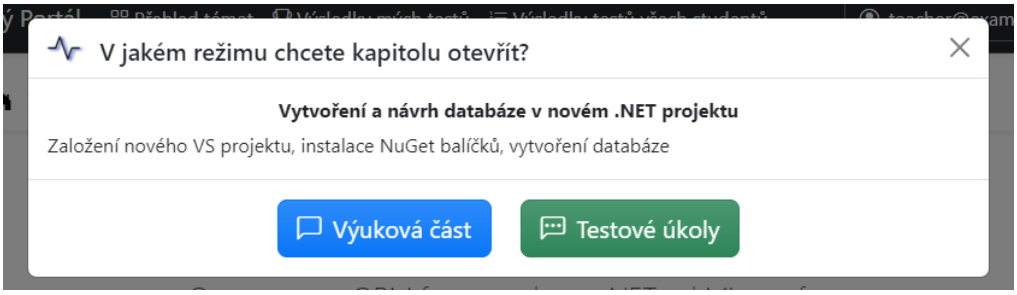
5.4.5 Kapitoly

Každé téma může obsahovat několik kapitol. Jejich cílem je umožnit smysluplně oddělit výukový obsah a testové úkoly dle toho, na jakou konkrétní oblast se v rámci celého tématu zaměřují. Kapitoly jsou vypsány pod sebou jako samostatné položky list-group s vlastním rozvržením obsahu, zahrnujícím informaci o počtu výukových zadání a testových úkolů v dané kapitole.



Obrázek 33. Přehled kapitol spadajících pod téma

Po výběru kapitoly se v podobě Bootstrap modal okna objeví dotaz, v jakém režimu se má kapitola otevřít. Na výběr je Výuková část a Testové úkoly.



Obrázek 34. Modal okno s výběrem varianty pro otevření kapitoly

5.4.6 Výuková část

Výuková část může obsahovat jeden nebo více různých výukových obsahů, resp. úkolů. Pro každý úkol je použit Bootstrap prvek accordion s minimálně jednou až třemi položkami. Tento prvek umožňuje rozbalení vlastního obsahu až po kliknutí na záhlaví konkrétní položky, je tedy ideálním řešením pro potřeby vytvořených úkolů v rámci této práce.

První položka každého výukového obsahu představuje zadání úkolu a je automaticky viditelná po načtení stránky. Je možné také přiložit soubor, který je zpřístupněn ke stažení na

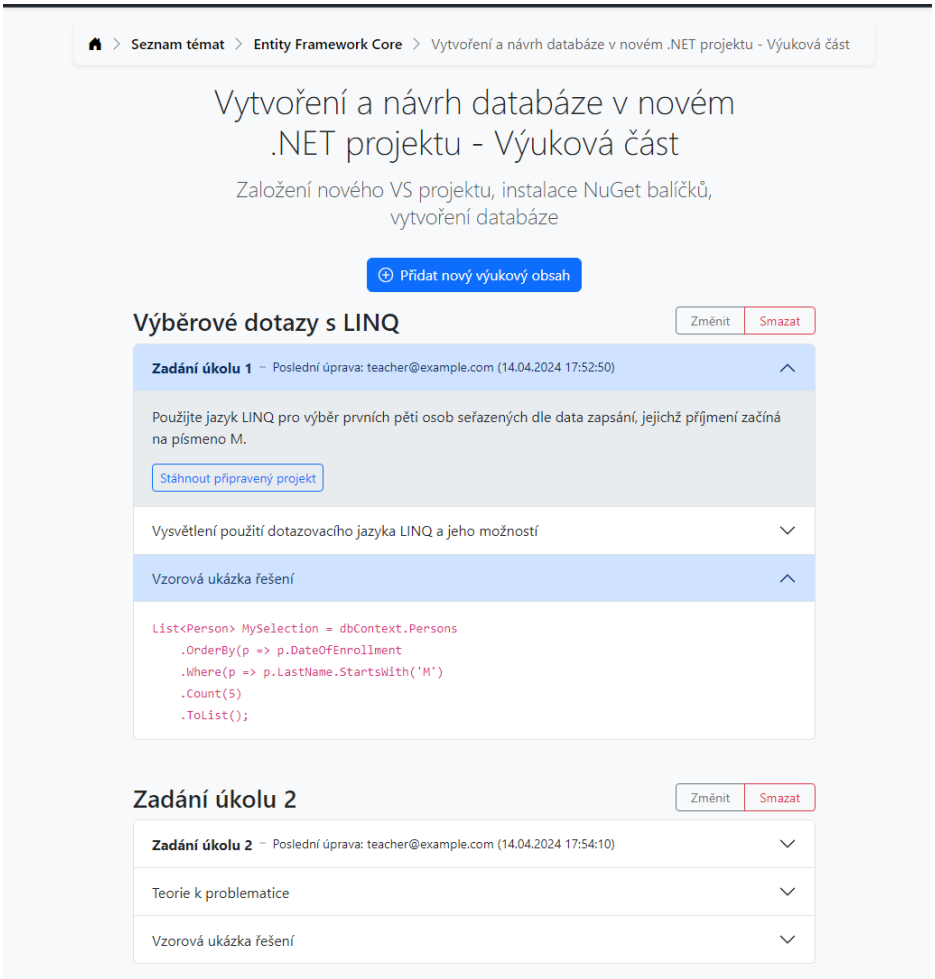
konci textu, a nahrát jeden obrázek. Pro vytvořené úkoly se zde jako soubor obvykle nachází .zip archiv obsahující předpřipravený projekt, do kterého stačí doplnit řešení, a již vyřešený projekt. Tlačítku pro stažení přiloženého souboru lze přidat vlastní název.

Druhá položka může obsahovat teorii, či popis jednotlivých kroků, jak lze zadání splnit.

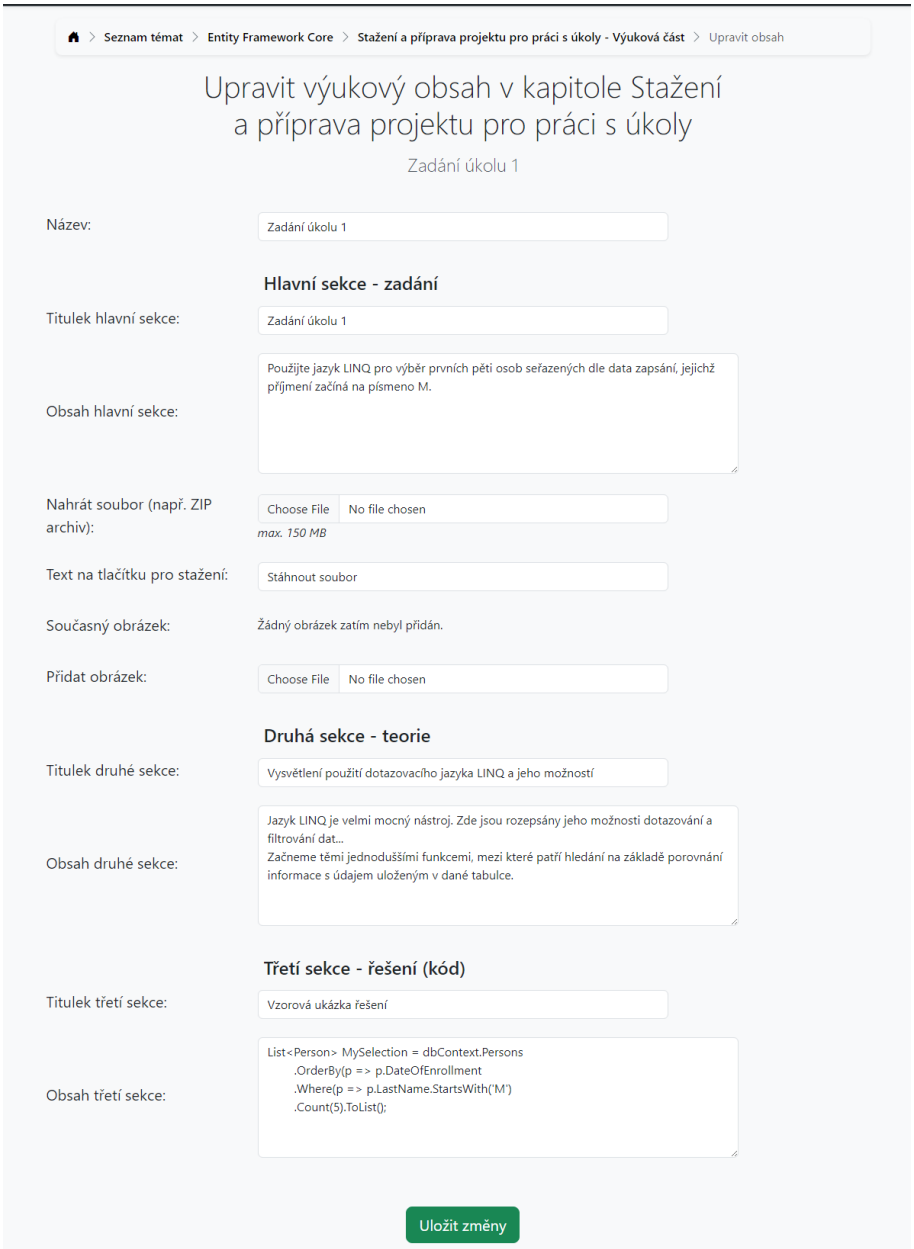
Poslední, třetí položka poté umožňuje nahlédnout na konkrétní možné řešení v podobě úryvku zdrojového kódu.

Prvky na stránce jsou velmi modulární. Pro každý přidaný výukový obsah se zobrazují pouze ty prvky, které nějaký svůj obsah skutečně mají, v opačném případě nebudou vůbec zobrazeny. Jediné povinné položky pro jeden výukový obsah jsou jeho hlavní nadpis a titulek a obsah první položky.

Pro účty s rolí Teacher se vedle tlačítek Přidat, Změnit a Smazat, fungujících obdobným způsobem jako v tématech a kapitolách, u jednotlivých položek také zobrazuje informace o jejich poslední úpravě.



Obrázek 35. Výuková část



Obrázek 36. Úprava výukového obsahu

5.4.7 Testové úkoly

Druhým typem obsahu v rámci kapitoly jsou testové úkoly. Úkolem je ke každému zadání uvést odpověď, která je automaticky zkontrolována na straně serveru. Jsou implementovány dva různé typy odpovědi, otevřený text nebo volba pravda / nepravda. Na základě zvoleného typu testu se způsob zápisu odpovědi liší.

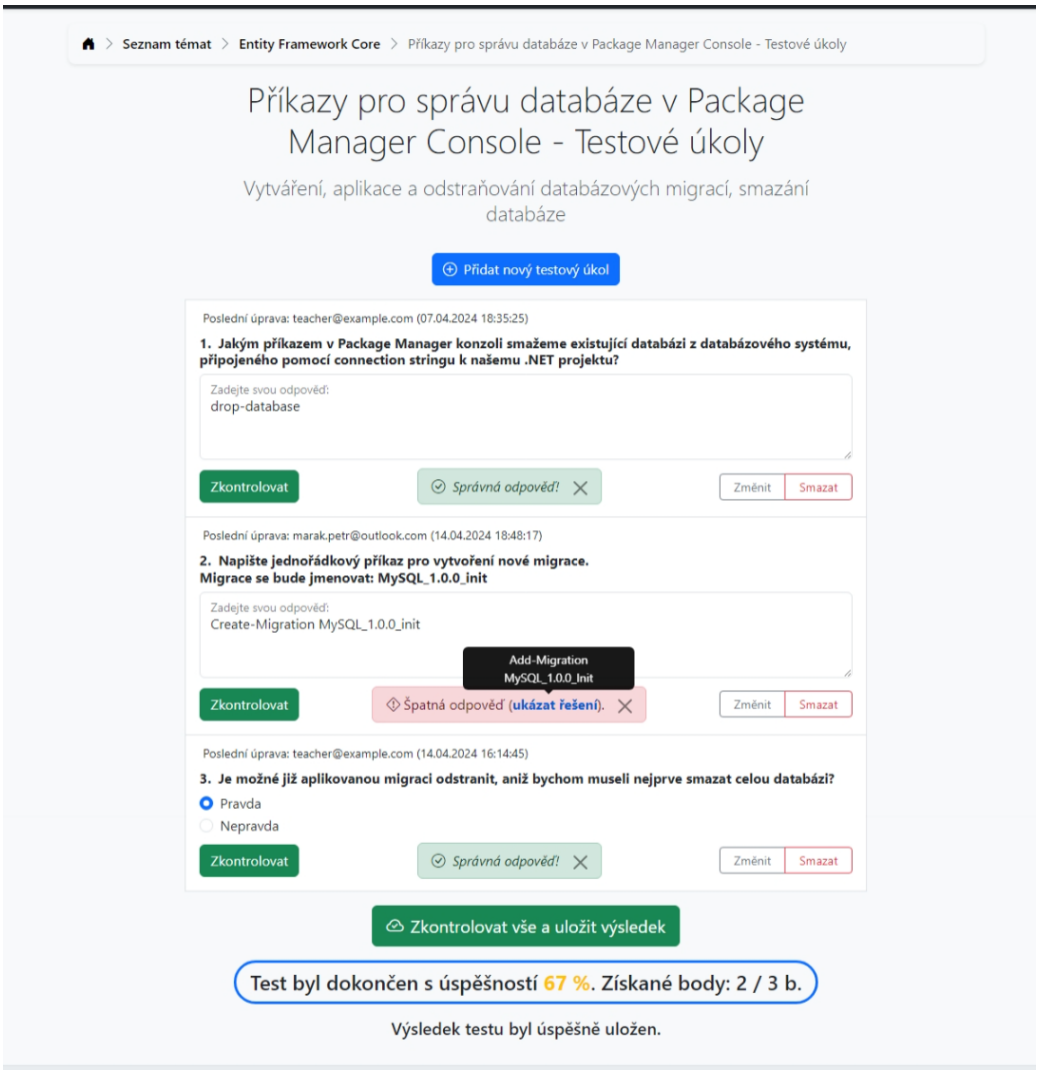
Každou odpověď si student může nechat zkontrolovat samostatně, výsledek se zobrazí v podobě Bootstrap alert proužku pod prostorem na odpověď. Vzhledem k zaměření celého

portálu zejména na podporu získávání znalostí přívětivou metodou, spíše než zkoušení, existuje možnost ukázat řešení po najetí kurzorem v případě špatné odpovědi.

Na konci každého testu se vyskytuje tlačítko pro hromadnou kontrolu všech vyplněných odpovědí a odeslání získaného výsledku na server. Po stisknutí se zobrazí úspěšnost za celý test a zpráva, zda byl výsledek uložen. Nový výsledek lze z jednoho účtu uložit maximálně jednou za 5 minut.

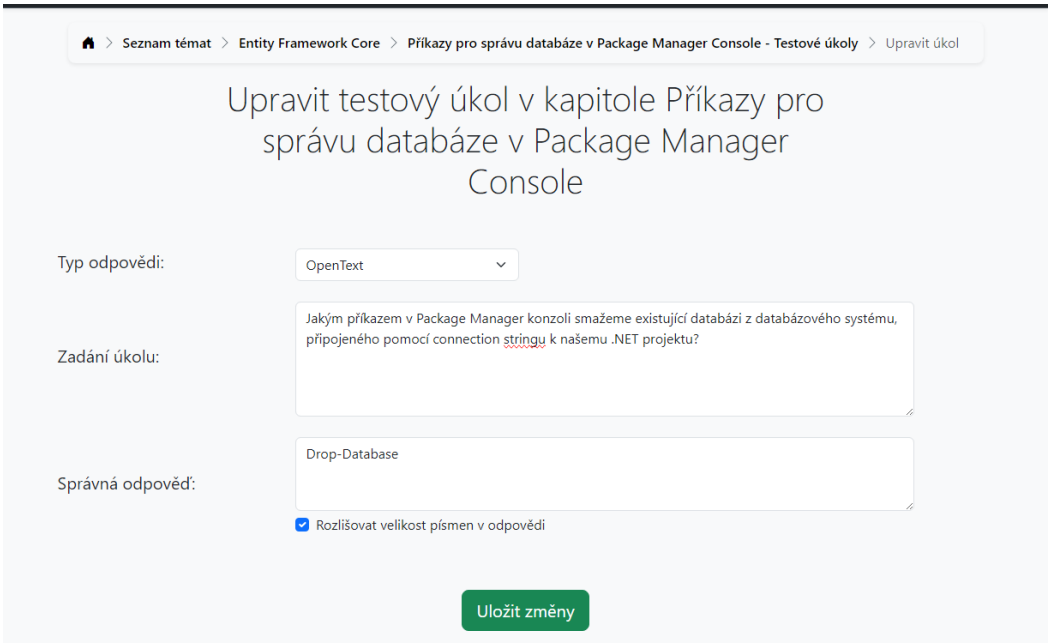
Tato funkcionalita je řízena pomocí Javascriptového kódu, který komunikuje s akčními metodami v controlleru na straně serveru, a na základě obdržených dat interaktivně přidává patřičné prvky a informace na stránku.

Pro účty s rolí Teacher jsou zde dostupné akce a zobrazované informace stejné jako ve výukové části.



Obrázek 37. Testové úkoly s vyplněným a odeslaným testem

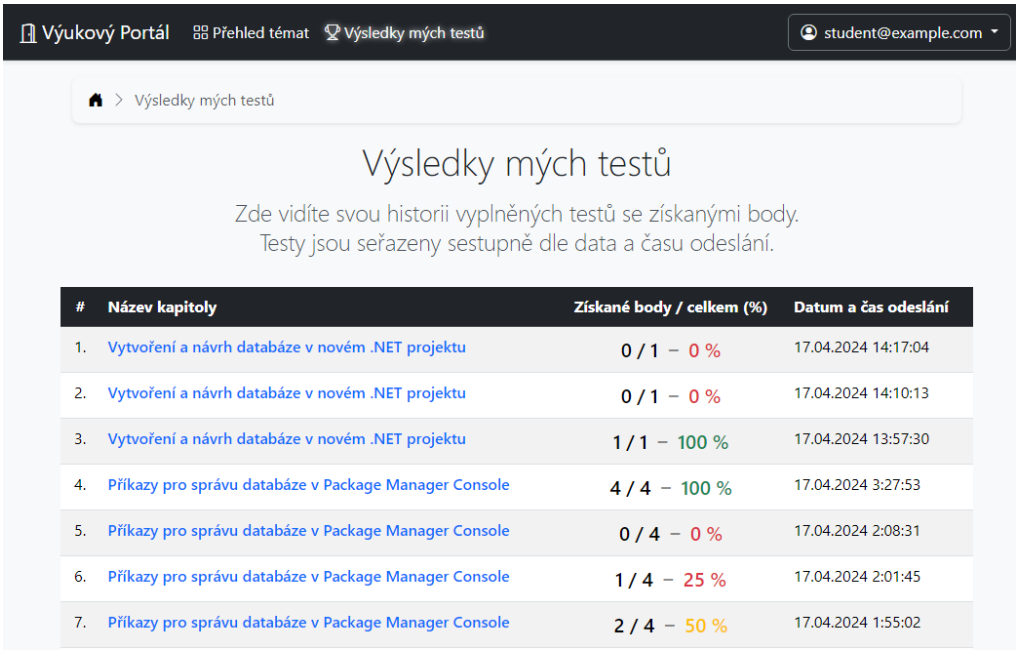
Na stránkách pro přidání a změnu testového úkolu se způsob zápisu správné odpovědi dynamicky mění spolu se změnou vybraného typu odpovědi a odpovídá následnému zobrazení na stránce s testovými úkoly pro studenty.



Obrázek 38. Úprava testového úkolu

5.4.8 Výsledky mých testů

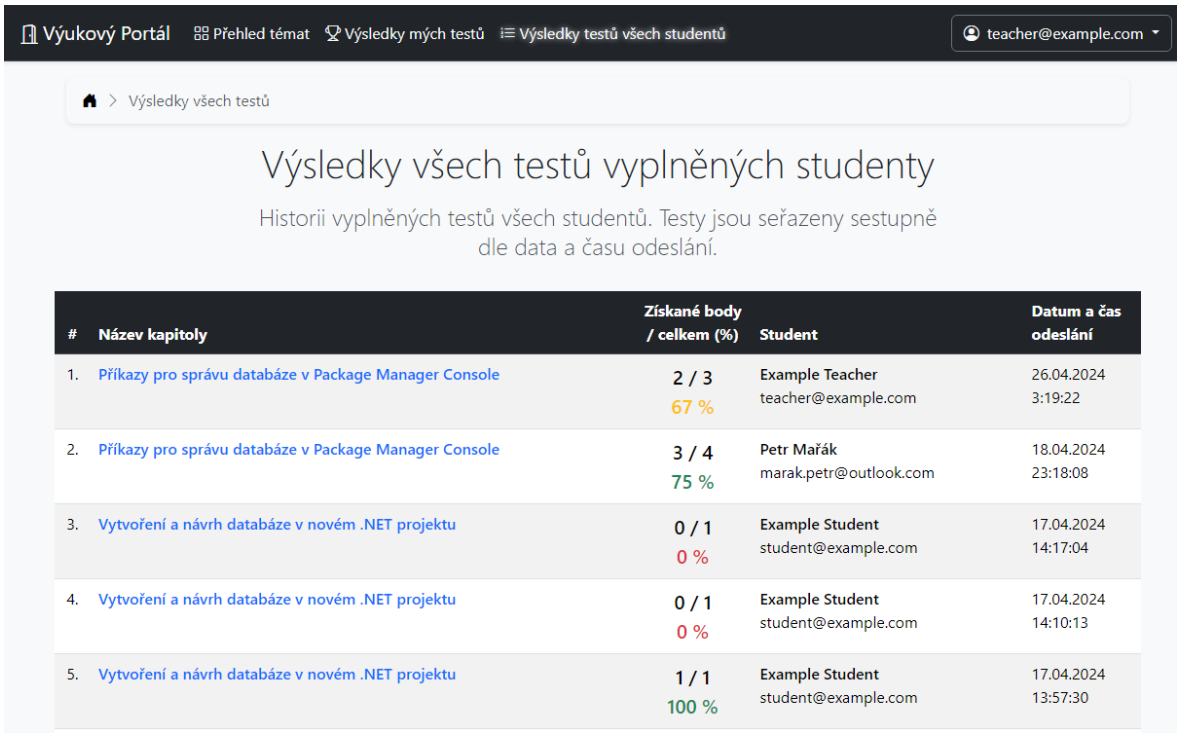
Stránka dostupná všem studentům, kteří si zde mohou prohlížet historii svých dokončených testů. Výpis je seřazen od nejnovějšího záznamu dle data a času odeslání. Počet vypsaných záznamů je omezen na posledních 200. Úspěšnost je uváděna v poměru získaných bodů i v procentech. Hodnota je zabarvena na základě úspěšnosti, kde zelená značí 75 % a více získaných bodů, žlutá alespoň 50 %, a červená barva se použije v ostatních případech.



Obrázek 39. Výsledky mých testů

5.4.9 Výsledky všech testů vyplněných studenty

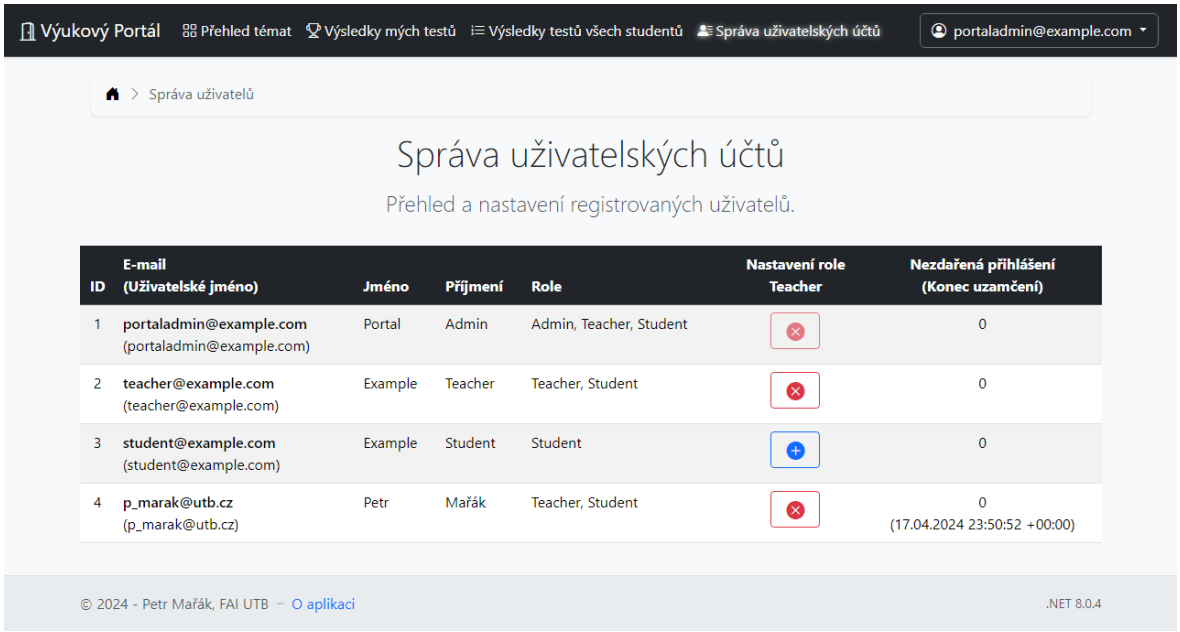
Přehled je dostupný pouze pro učitele s rolí Teacher. Je velmi podobný stránce Výsledky mých testů, s tím rozdílem, že se zde zobrazují odeslané testy od všech studentů, a je zde navíc sloupec se jménem a e-mailovou adresou studenta, který test vyplnil.



Obrázek 40. Výsledky všech testů vyplněných studenty

5.4.10 Správa uživatelských účtů

Do této sekce má přístup pouze účet s rolí Admin, poskytující nejvyšší oprávnění. Stránka slouží pro prohlížení informací o vytvořených uživatelských účtech na portále a umožňuje přidávat a odebírat členství v roli Teacher pro jednotlivé uživatele.



Obrázek 41. Správa uživatelských účtů

5.5 Možnosti dalšího rozvoje aplikace

V této části jsou uvedeny návrhy pro vylepšení výukového portálu, zejména přidání funkcionality v rámci dalšího vývoje.

- Změna hesla pro uživatelský účet administrátorem.
- Odstranění uživatelského účtu administrátorem.
- Změna hesla pro vlastní účet.
- Import a export uložených dat, například s využitím formátu JSON.
- Možnost dále dělit témata dle kategorií. Vhodné pro případ existence velkého množství vzájemně nesouvisejících témat.
- Podpora přidání více obrázků pro jeden výukový obsah.

- Možnost ručně změnit zobrazované pořadí výukových a testových úkolů.
- Funkce označit téma jako dokončené.
- Náhodné generování / výběr testových úkolů z většího celku.
- Více typů odpovědí pro testové úkoly. Například vícenásobný výběr všech správných tvrzení implementovaný pomocí checkboxů. Výběr jediné odpovědi z více možností pomocí radio buttonů.
- Více než jedna správná odpověď pro testové úlohy s otevřeným textem.
- Pro každý odeslaný test uložit také všechny zadané odpovědi studenta.
  Tato funkce by značně zvýšila nároky na velikost databáze, jelikož by bylo třeba ukládat také zadání všech otázek pro případ, pokud by učitel později zadané otázky v daném testu upravil.

ZÁVĚR

Tato bakalářská práce se zabývá tvorbou studijních materiálů demonstrující způsoby práce s databází ve frameworku .NET.

V teoretické části je zpracována charakteristika současného stavu technologií databázových systémů se zaměřením na relační, nerelační a cloudové databáze. Dále je zde popsána softwarová platforma Microsoft .NET a její součásti, zejména ASP.NET Core využívaný v části praktické. Následuje rešerše .NET technologií pro práci s databázemi, konkrétně ADO .NET, Dapper a Entity Framework Core.

Praktická část se věnuje dvěma hlavním cílům. Prvním z nich je návrh praktických úkolů, pomocí kterých si mohou studenti vyzkoušet práci s databází uvnitř připravené ASP.NET Core aplikace. Pro tvorbu úkolů byla vybrána technologie EF Core. Pro každý úkol je připraveno zadání, postup řešení, implementace řešení v podobě zdrojového kódu a možné kontrolní otázky. Úkoly studenta postupně provedou instalací EF Core do projektu aplikace, připojením k databázi, vytvořením entit a vlastního databázového kontextu a databázovými migracemi. Další úkoly následně pokračují výběrovými dotazy, implementací funkce pro vytváření, editaci a odstraňování dat, a nakonec představí možnosti změny konfigurace databázového systému pomocí anotací dat a fluent API.

Druhým cílem v rámci praktické části bylo navrhnout a vyvinout výukovou aplikaci „Výukový Portál“. Aplikace nabízí přívětivé prostředí pro práci s navrženými materiály a úkoly, správu uživatelských účtů a vyhodnocení testů. Také může sloužit jako praktická ukázka implementace vybraných technologií do .NET aplikace s reálnou funkcionalitou. Aplikace pro svůj běh používá Entity Framework Core s MySQL databází.

Vytvořené úkoly spolu s aplikací mohou sloužit jako pomůcka pro výuku předmětů zaměřených na databáze a programování s technologiemi .NET.

Tato práce poskytuje ucelený rámec, jenž velmi usnadňuje vzdělávání online cestou, a to jak pro pedagogy, tak i studenty.

SEZNAM POUŽITÉ LITERATURY

[1] ORACLE. *What Is a Database?* Online. C2024. Dostupné z: https://www.oracle.com/database/what-is-database. [cit. 2024-03-01].

[2] HAIGH, Tom. *A.M. TURING AWARD.* Online. C2019. Dostupné z: https://amturing.acm.org/award_winners/bachman_1896680.cfm. [cit. 2024-03-01].

[3] MICROSOFT CORPORATION. *Databáze NoSQL – co je NoSQL?* Online. C2024. Dostupné z: https://azure.microsoft.com/cs-cz/resources/cloud-computing-dictionary/what-is-nosql-database. [cit. 2024-03-01].

[4] GOOGLE. *What is a relational database?* Online. C2024. Dostupné z: https://cloud.google.com/learn/what-is-a-relational-database. [cit. 2024-03-01].

[5] IBM. *What is a NoSQL database?* Online. C2024. Dostupné z: https://www.ibm.com/topics/nosql-databases. [cit. 2024-03-01].

[6] GOOGLE. *What is NoSQL?* Online. C2024. Dostupné z: https://cloud.google.com/discover/what-is-nosql. [cit. 2024-03-01].

[7] MICROSOFT CORPORATION. *Non-relational data and NoSQL.* Online. 2022. Dostupné z: https://learn.microsoft.com/en-us/azure/architecture/data-guide/big-data/non-relational-data. [cit. 2024-03-01].

[8] ORACLE. *What is a cloud database?* Online. C2024. Dostupné z: https://www.oracle.com/database/what-is-a-cloud-database. [cit. 2024-03-01].

[9] IBM. *What is Database-as-a-Service (DBaaS)?* Online. C2024. Dostupné z: https://www.ibm.com/topics/dbaas. [cit. 2024-03-01].

[10] MICROSOFT CORPORATION. *Build. Test. Deploy.* Online. 2024. Dostupné z: https://dotnet.microsoft.com/en-us. [cit. 2024-05-12].

[11] MICROSOFT CORPORATION. *Introduction to .NET*. Online. 2024. Dostupné z: https://learn.microsoft.com/en-us/dotnet/core/introduction. [cit. 2024-05-12].

[12] MICROSOFT CORPORATION. *ADO.NET*. Online. 2021. Dostupné z: https://learn.microsoft.com/en-us/dotnet/framework/data/adonet. [cit. 2024-03-11].

[13] BOEHM, Anne a MEAD, Ged, MURACH, Mike (ed.). *Murach's ADO. NET 4 Database Programming with C# 2010*. 4th Edition. United States of America: Mike Murach & Associates, 2011. ISBN 978-1-890774-63-9.

[14] *Welcome To Learn Dapper*. Online. Learn Dapper. 2023. Dostupné z: https://www.learndapper.com. [cit. 2024-05-12].

[15] *Querying Data With Dapper*. Online. Learn Dapper. 2023. Dostupné z: https://www.learndapper.com/dapper-query. [cit. 2024-05-12].

[16] *Executing Non-Query Commands With Dapper*. Online. Learn Dapper. 2023. Dostupné z: https://www.learndapper.com/non-query. [cit. 2024-05-12].

[17] MICROSOFT CORPORATION. *Entity Framework Core*. Online. Microsoft Learn. 2021. Dostupné z: https://learn.microsoft.com/en-us/ef/core. [cit. 2024-05-12].

[18] ELLINGWOOD, Justin. *What is an ORM?* Online. Prisma's Data Guide. 2023. Dostupné z: https://www.prisma.io/dataguide/types/relational/what-is-an-orm. [cit. 2024-05-12].

[19] MICROSOFT CORPORATION. *Database Providers*. Online. Microsoft Learn. 2024. Dostupné z: https://learn.microsoft.com/en-us/ef/core/providers. [cit. 2024-05-12].

[20] MICROSOFT CORPORATION. *DbContext Lifetime, Configuration, and Initialization*. Online. Microsoft Learn. 2023. Dostupné z: https://learn.microsoft.com/en-us/ef/core/dbcontext-configuration. [cit. 2024-05-12].

[21] MICROSOFT CORPORATION. *Creating and Configuring a Model*. Online. Microsoft Learn. 2023. Dostupné z: https://learn.microsoft.com/en-us/ef/core/modeling. [cit. 2024-05-12].

[22] MICROSOFT CORPORATION. *Entity Types*. Online. Microsoft Learn. 2023. Dostupné z: https://learn.microsoft.com/en-us/ef/core/modeling/entity-types. [cit. 2024-05-12].

[23] MICROSOFT CORPORATION. *Keys*. Online. Microsoft Learn. 2022. Dostupné z: https://learn.microsoft.com/en-us/ef/core/modeling/keys. [cit. 2024-05-12].

[24] MICROSOFT CORPORATION. *Generated Values*. Online. Microsoft Learn. 2023. Dostupné z: https://learn.microsoft.com/en-us/ef/core/modeling/generated-properties. [cit. 2024-05-12].

[25] MICROSOFT CORPORATION. *Introduction to relationships*. Online. Microsoft Learn. 2023. Dostupné z: https://learn.microsoft.com/en-us/ef/core/modeling/relationships. [cit. 2024-05-12].

[26] MICROSOFT CORPORATION. *Migrations Overview*. Online. Microsoft Learn. 2023. Dostupné z: https://learn.microsoft.com/en-us/ef/core/managing-schemas/migrations. [cit. 2024-05-12].

[27] MICROSOFT CORPORATION. *Scaffolding (Reverse Engineering)*. Online. Microsoft Learn. 2023. Dostupné z: https://learn.microsoft.com/en-us/ef/core/managing-schemas/scaffolding. [cit. 2024-05-12].

[28] GITHUB. *Northwind-SQLite3*. Online. 2023. Dostupné z: https://github.com/jpwhite3/northwind-SQLite3. [cit. 2024-05-12].

[29] MICROSOFT CORPORATION. *Entity Framework Core tools reference - Package Manager Console in Visual Studio*. Online. Microsoft Learn. 2023. Dostupné z: https://learn.microsoft.com/en-us/ef/core/cli/powershell. [cit. 2024-05-12].

[30] MICROSOFT CORPORATION. *Language Integrated Query (LINQ)*. Online. Microsoft Learn. 2023. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/linq. [cit. 2024-05-12].

[31] *Datatypes In SQLite*. Online. SQLite. 2022. Dostupné z: https://www.sqlite.org/datatype3.html. [cit. 2024-05-12].

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

DB	Database.
DBMS	Database Management System.
EF	Entity Framework.
IDE	Integrated Development Environment.
LINQ	Language Integrated Query.
ORM	Object Relational Mapper.
SQL	Structured Query Language.
VS	Visual Studio.

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: CD

Přiložené CD obsahuje soubor prilohy.zip, který zahrnuje zdrojové kódy aplikace vytvořené pro databázové úkoly, zdrojový kód aplikace Výukový Portál a bakalářskou práci ve formátu PDF.