Ukázková aplikace s využitím ASP.NET Core a COSMOS DB

Adam Hamšík

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:	**Adam Hamšík**
Osobní číslo:	**A21054**
Studijní program:	**B0613A140020 Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Ukázková aplikace s využitím ASP.NET Core a COSMOS DB**
Téma práce anglicky:	**Sample Application Using ASP.NET Core and COSMOS DB**

Zásady pro vypracování

1. Seznamte se s technologií ASP.NET Core a Cosmos DB v aktuální verzi.
2. Proveďte rešerši HTML frameworků.
3. Zpracujte vzorové příklady a jejich řešení pro cvičení a přednášky.
4. Vytvořte podpůrné prezentace v Powerpointu.
5. Navrhněte sadu testovacích úkolů včetně řešení ve formátu vhodném pro LMS Moodle.

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

Seznam doporučené literatury:

1. PAZ, Guay a Jose ROLAO. Introduction to azure cosmos db. In: Microsoft Azure Cosmos DB Revealed. Apress, Berkeley, CA, 2018. p. 1-23. ISBN 9781484233504.
2. HILLAR, Gaston C.; YÖNDEM, Daron. Guide to NoSQL with Azure Cosmos DB: Work with the massively scalable Azure database service with JSON, C#, LINQ, and. NET Core 2. Packt Publishing Ltd, 2018. ISBN 1789612896.
3. Dokumentace k Azure Cosmos DB, 2020. Dokumentace k Azure Cosmos DB [online]. USA: Microsoft [cit. 2020-11-25]. Dostupné z: https://docs.microsoft.com/cs-cz/azure/cosmos-db/
4. OWLER, Adam. NoSQL for dummies. John Wiley & Sons, 2015. ISBN: 9788126554904.
5. FREEMAN, Adam. Putting ASP. NET Core in Context. In: Pro ASP. NET Core 3. Apress, Berkeley, CA, 2020. p. 3-7. ISBN: 9781484254400

Vedoucí bakalářské práce: **doc. Ing. Petr Šilhavý, Ph.D.**
Ústav počítačových a komunikačních systémů

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

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

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

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 Adam Hamšík, v.r.

ABSTRAKT

Předmětem bakalářské práce je seznámení s technologiemi pro vývoj webových aplikací ASP.NET Core a Cosmos DB a součástmi, s kterými tyto technologie dokážou pracovat a také vypracování příkladů na základě kterých můžou čtenáři ověřit své znalosti a porozumět více tématu.

V teoretické části se rozebírá způsob fungování popisovaných technologií a vzájemné integrace mezi nimi. Na teoretické poznatky navazuje praktická část, ve které se řeší konkrétní implementace technologií a na základě nich je vytvořena ukázková aplikace a jsou popsány všechny nezbytné náležitosti a také vytvořeny podpůrné materiály, které slouží k jednoduchému a praktickému otestování znalostí čtenáře.

Klíčová slova: ASP.NET Core, Cosmos DB, webové aplikace, HTML framework, NoSQL, databázové systémy

ABSTRACT

The subject of this bachelor's thesis is an introduction web development technologies ASP.NET Core and Cosmos DB, and components these technologies can work with, as well as the development of examples through which readers can verify their knowledge and gain a deeper understanding of the topic.

The theoretical part discusses the functioning of the described technologies and their integration with each other. Build on the theoretical knowledge, the practical part addresses the specific implementation of these technologies, resulting in the creation of a sample application. This section also describes all the necessary details and includes the creation of supporting materials which facilitate easy and practical testing of user knowledge.

Keywords: ASP.NET Core, Cosmos DB, web applications, HTML framework, NoSQL, database systems

Chtěl bych poděkovat mému vedoucímu panu Ing. Petru Šilhavému, Ph.D. za vedení mé práce, konzultace a připomínky, které vedly k řešení problémů v mé bakalářské práci.

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

Prohlašuji, že při tvorbě této práce jsem použil/a nástroj generativního modelu AI ChatGPT https://chat.openai.com za účelem lepší úpravy textu. Po použití tohoto nástroje jsem provedl kontrolu obsahu a přebírám za něj plnou zodpovědnost.

OBSAH

ÚVOD

V době, kdy se se neustále vše digitalizuje a data se rozrůstají do enormních rozměrů je důležité řešit tyto výzvy spojené s dostupností a jejich zpracováním v reálném čase. Tato bakalářská práce se zaměřuje na využití pokročilých technologií pro tvorbu webových aplikací s efektivním databázovým systémem. Tyto technologie představují ASP.NET Core a Cosmos DB.

ASP.NET Core je moderní framework pro tvorbu webových aplikací a služeb, zatímco Cosmos DB představuje globálně distribuovanou databázovou službu s vysokou dostupností a efektivním škálováním. Integrací těchto technologií tak můžeme vytvářet velmi výkonné a robustní aplikace pro webové využití, které budou navíc velmi dostupné.

Cílem práce je tak seznámení čtenářů s použitím těchto technologií, klíčovými vlastnostmi, které tyto technologie nabízejí a způsoby využití s vytvořením jednoduché aplikace, která se užívá k demonstraci základních funkcí a představuje příkladový materiál, díky kterému by čtenář měl lépe pochopit problematiku a být schopen vytvořit vlastní webovou aplikaci s využitím těchto technologií.

V teoretické části je pozornost zaměřena na popis jednotlivých technologií, jejich klíčových komponent, možností využití a principů na kterých tyto technologie pracují.

V praktické části je popsána specifikace vytvářené aplikace, včetně popisu samotné aplikace a způsobů využití. Dále jsou vytvářeny podpůrné materiály, které slouží pro praktické seznámení s funkcemi technologií a otestování znalostí čtenáře.

I. TEORETICKÁ ČÁST

1 ASP.NET CORE

ASP.NET Core je moderní multiplatformní framework přes který můžeme vytvářet webové aplikace a služby s kombinací jazyků C#, HTML, CSS a JavaScriptu. Jedná se o open-source software, což znamená, že je svobodně dostupný a umožňuje volné přispívání vývojářů k jeho rozvoji a optimalizaci [12].

1.1 HISTORIE A SOUČASTNOST

ASP.NET Core vznikl v roce 2016 a je přepisem původního ASP.NET frameworku a kombinací ASP.NET MVC (Model-View-Controller) a ASP.NET Web API.

Představuje tak jednotné programovací rozhraní, které je navíc nezávislé na platformě, protože původní ASP.NET byl pouze pro Windows.

Obrázek 1 Historie verzí ASP.NET Core [26].

Z obrázku je možné si všimnout chybějící verze 4.0, a to z toho důvodu, že dříve existoval .NET Framework – předchůdce .NET Core, který byl využíván ASP.NET. Poté, co byl .NET Framework nahrazen .NET Core, přešlo se stejně i z ASP.NET na ASP.NET Core. Protože .NET Framework byl ve verzi 4, byla úmyslně vynechána verze .NET Core 4 (ASP.NET Core 4), aby se to nepletlo. Pokud tedy v dnešní době hovoříme o ASP.NET Core (nebo .NET Core) je nutné specifikovat slovo "Core" a číslo verze, v případě, že je číslo verze menší než 5. Od verze 5 již můžeme říkat pouze .NET, např: .NET 5 (ASP.NET 5).

V současnosti je ASP.NET Core ve verzi 8.0, která vyšla 14.11.2023. Neustálým vývojem ASP.NET Core potvrzuje svou pozici jako jeden z předních webových frameworků. Nová verze sebou přináší vylepšení výkonu a bezpečnosti, což usnadňuje práci díky lepší integraci s nástroji [19].

1.2 KLÍČOVÉ VLASTNOSTI A FUNKCE

ASP.NET Core má spoustu klíčových vlastností jako například [19]:

- Blazor: Komponenta ASP.NET Core pro sdílení logiky mezi serverem a klientem s použitím jazyka C#. Tato technologie umožňuje zvýšit produktivitu a odstraňuje nutnost opakování kódu. Nahrazuje tak jazyk JavaScript, který bývá používán na straně klienta, jazykem C#.
- Interaktivní uživatelské rozhraní: Nabízí rozsáhlé možnosti pro tvorbu interaktivních a dynamických uživatelských rozhraní. Schopnost vytvářet responzivní a přívětivé webové aplikace díky Razor Pages a MVC (Model-View-Controller) architektury.
- Multiplatformní: Schopnost vývoje webových aplikací na Windows, Linux a Mac.
- Asynchronní programování: Framework plně podporuje asynchronní programování, což umožňuje vytvářet responzivní aplikace bez blokování vláken
- Flexibilní hostování: Schopnost hostovat aplikace na různých platformách jako IIS, Docker a Nginx, což umožnuje širokou škálu možností pro nasazení aplikací.
- Rychlost: ASP.NET Core patří mezí nejrychlejší populární webové frameworky. S téměř 7 milióny požadavků za sekundu dosahuje lepších výsledků než webové frameworky jako Java Servlet nebo Node.js (Tabulka 1).
- Tag Helpers: Umožnují implementaci server-side logiky do HTML elementů.

Tabulka 1 Porovnání rychlosti jednotlivých webových frameworků [20].

Framework	Požadavků/Sekundu	Jazyk
ASP.NET.CORE	7 023 107	C#
DROGON	5 969 800	C++
SERVLET	2 208 344	Java
PHP-NGX	1 589 429	PHP
NODEJS	607 368	JS
DJANGO	79 188	Python

1.2.1 Blazor

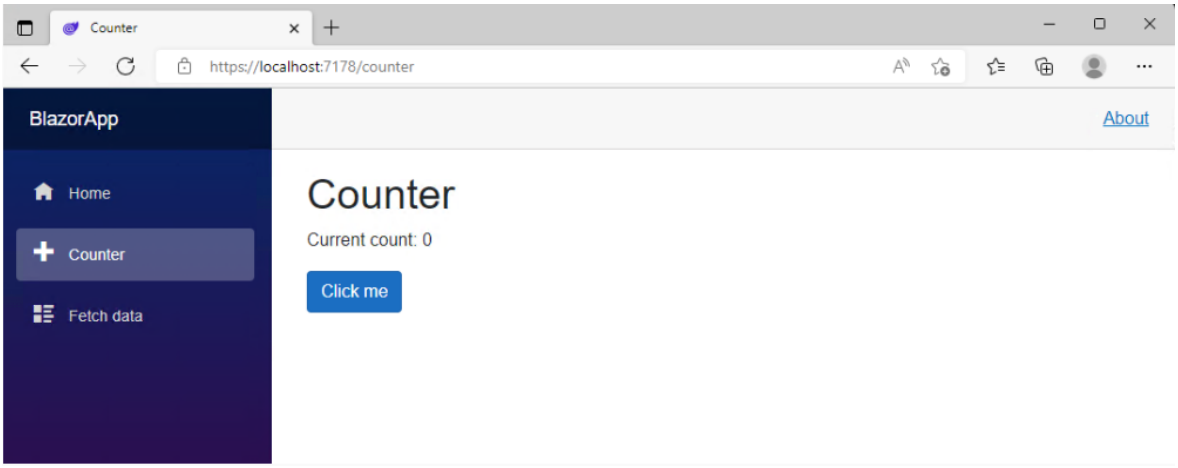
Hlavním cílem Blazoru je sdílení logiky mezi serverem a klientem využitím .NET. S Blazorem můžeme sestavit nejen webové aplikace, ale taky mobilní a desktopové.

S využitím Razor stránek můžeme přidat do HTML také C# a můžeme tak kombinovat společnou logiku a usnadnit vytváření dynamických uživatelských rozhraní.

```
<PageTitle>Counter</PageTitle>

<h1>Counter</h1>

<p role="status">Current count: @currentCount</p>

<button class="btn btn-primary" @onclick="IncrementCount">Click me</button>

@code {
    private int currentCount = 0;

    private void IncrementCount()
    {
        currentCount++;
    }
}
```

Obrázek 2 Využití Razor v Blazor aplikaci [27].

C# kód tak přidáváme k html pomocí “@”. Následují pak složené závorky, v kterých můžeme psát řádků kódu. Funkce „IncrementCount“ se zavolá při kliknutí na tlačítko a dojde tak ke zvýšení hodnoty počítadla.



Obrázek 3 Blazor stránka s počítadlem [27].

1.2.2 MVC

Pomocí architektury MVC (Model-View-Controller) dokážeme vytvářet přehledné webové aplikace a API (Application-User-Interface).

MVC je návrhový vzor, který rozděluje aplikaci do tří komponent – Modely, View a Controllery. Pomocí této struktury je aplikace přehlednější pro uživatele a snadnější k testování.

- Model: Model představuje návrh dat v naší aplikaci. Můžeme mít například model „uživatel“, který představuje všechny uživatele a jejich atributy – které mohou být například: jméno, příjmení, datum narození, telefonní číslo (Obrázek 4). Modely jsou následně předávány do databáze, kde jsou k uchovávány již jednotlivé záznamy. V ASP.NET Core vytváříme modely pomocí tříd v C# kódu.

```
1. public class User
2. {
3.     public string FirstName { get; set; }
4.     public string LastName { get; set; }
5.     public string PhoneNumber { get; set; }
6. }
```

Obrázek 4 Model uživatele

- View: View znázorňuje stránku, kterou uživatel vidí ve webové aplikaci – například domovská stránka uživatele s jeho údaji (Obrázek 5). Do view tak můžeme předat jednotlivé modely a zobrazit jejich data v jednotlivých formulářích. Ve view kombinujeme pomocí Razor stránek propojení HTML s modelem zapsaným v C# kódu.

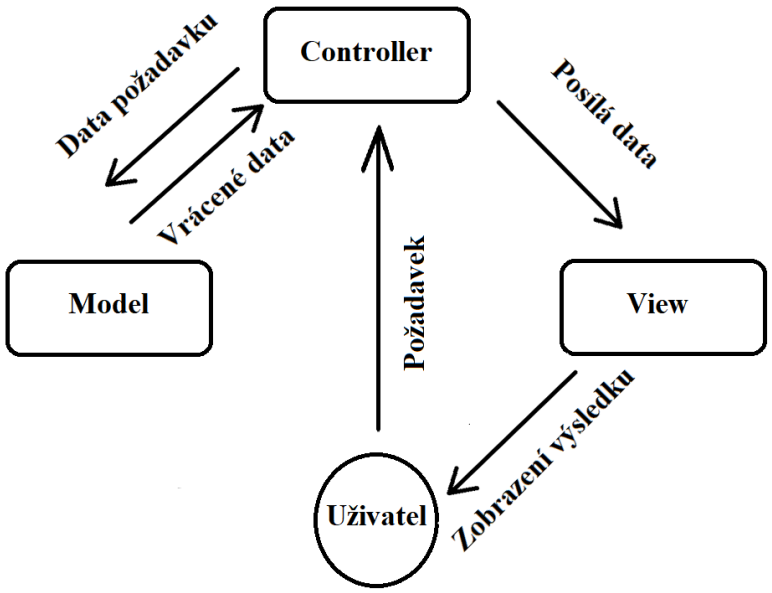
```
 1. @model User;
 3. @{
 4.     ViewData["Title"] = "My Profile";
 5. }
 7. <div class="container my-5">
 8.     <div class="row">
 9.         <div class="col-md-8 offset-md-2">
10.             <div class="card shadow">
11.                 <div class="card-body">
12.                     <h1 class="card-title text-center mb-4">@ViewData["Title"]</h1>
13.                     <dl class="row">
14.                         <dt class="col-sm-4">Name</dt>
15.                         <dd class="col-sm-8">@Model.FirstName</dd>
16.
17.                         <dt class="col-sm-4">Surname</dt>
18.                         <dd class="col-sm-8">@Model.LastName</dd>
19.
20.                         <dt class="col-sm-4">Email</dt>
21.                         <dd class="col-sm-8">@Model.PhoneNumber</dd>
22.                     </dl>
23.                 </div>
24.             </div>
25.         </div>
26.     </div>
27. </div>
```

Obrázek 5 View stránky uživatele

- Controller: Abychom docílili propojení mezi modelem a view, je potřeba to celé řídit. K tomu se užívá controller, který zpracovává požadavky stránky a na základě nich posílá nebo přijímá modely z nebo do view. Slouží jako propojení mezi těmito dvěma částmi webové aplikace. Controller nakonec odešle zpracovaný požadavek zpátky a view bude zobrazený na stránce (Obrázek 7).

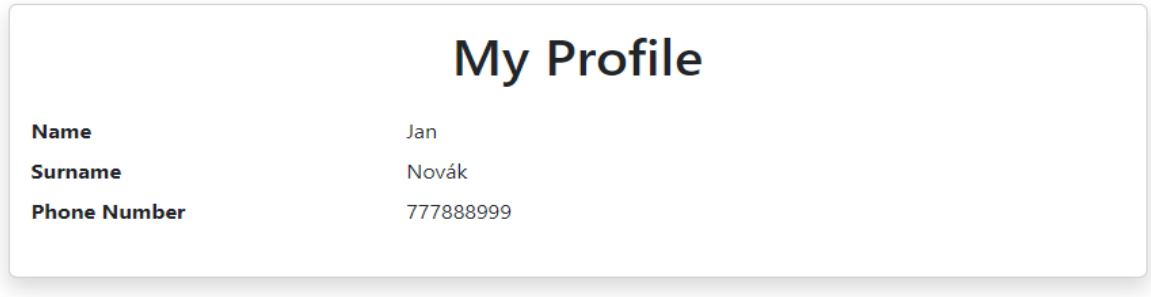
```
1. public class ProfileController : Controller
2. {
3.     [HttpGet]
4.     public IActionResult MyPage()
5.     {
6.         User user = new User { FirstName = "Jan", LastName = "Novák", PhoneNumber =
"777888999" };
7.         return View(user);
8.     }
9. }
```

Obrázek 6 Controller pro předání modelu do view



Obrázek 7 Princip fungování návrhového vzoru MVC [28].

Výsledná stránka (která je reprezentována pomocí view) se následně zobrazí uživateli po úspěšném zpracování požadavku.



Obrázek 8 Zobrazená profilová stránka

1.2.3 Tag Helpers

Tag helpers přidávají na základně speciálních HTML prefixů - „asp-“ integraci s server-side kódem. Vytvářejí tak dynamické komponenty přímo v Razor views. Tento přístup umožnuje efektivní manipulaci s HTML elementy a atributy na serveru [36].

Můžeme například ve formuláři pomocí Tag Helpers nastavit akci, která se zavolá po odeslání formuláře.

```
1. <form asp-action="MyPage" asp-controller="Me" asp-area="User" method="get">
2.     <input type="hidden" name="userName" value="@friend.UserName" />
3.     <button type="submit" class="link-primary" id="profileLink">Profile</button>
4. </form>
```

Obrázek 9 Příklad zobrazení určité stránky na základě metody v controlleru

Tento příklad říká, že se nasměrujeme na metodu „MyPage“ nacházející se v controlleru „MeController“ a oblasti „User“. Správně se tak dostaneme na požadované místo.

2 COSMOS DB

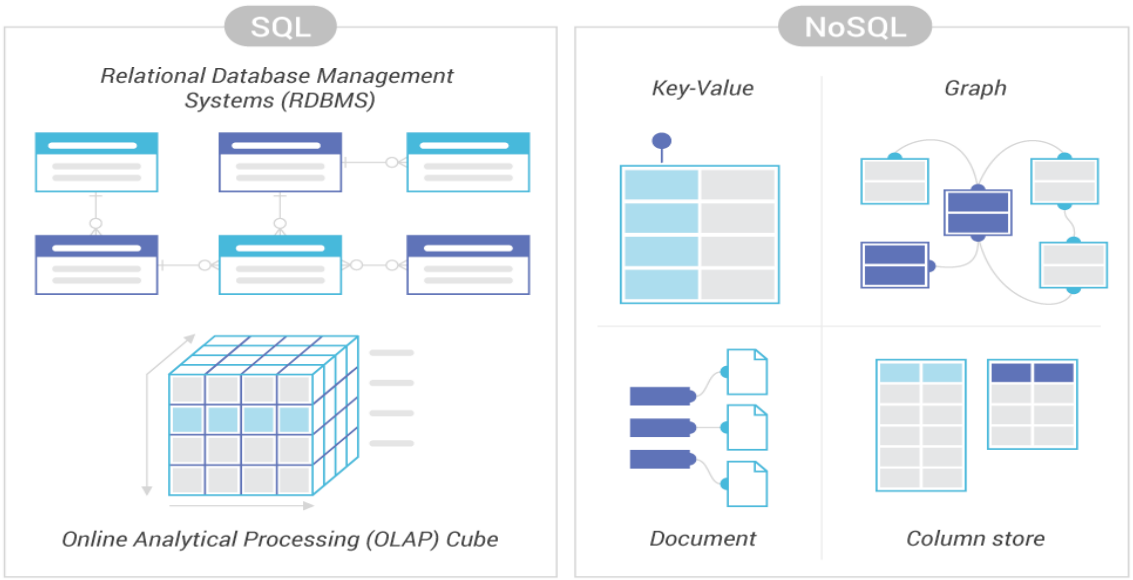
Cosmos DB je globálně distribuované databázová služba od Azure, cloudové platformy spadající pod společnost Microsoft. V Cosmos DB je možnost zvolení libovolné NoSQL databázové API, to umožňuje různorodost pro práci s databází a využití pro různá aplikační řešení.

2.1 NOSQL

NoSQL (not only SQL) je způsob provedení databáze, který zahrnuje různé datové modely, které jsou odlišné od tradičního tabulkového modelu relačních databází (Obrázek 10). Dokáže tak využívat různých způsobů ukládání dat, a to například dokumentů přes JSON (JavaScript object notation) nebo XML (Extensinle Markup Language) [1].

Při využití NoSQL tak může docházet k denormalizaci, protože je zde brán důraz především na zjednodušení. Ve velkých databázových aplikacích, kde je potřeba zpracovávat obrovské množství dat může docházet k problémům s rychlostí přístupu.

V případě RDBMS je použito velké množství různých tabulek, které mezi sebou mají vztahy, na jejichž základě jsou pomocí SQL (structured query language) vytvářeny tabulky, které spojují vzájemně více tabulek a dávají tak jeden celek. Nicméně při delších skriptech a náročnějších datech je spojováno až příliš mnoho tabulek a výsledný skript tak může být až příliš náročný. Tento problém odstraňuje NoSQL, kdy data nenabývají vertikálně, nýbrž horizontálně [1].



Obrázek 10 Vzhled SQL a NoSQL [21].

2.1.1 Typy NoSQL Databází

Spousta společností neustále usiluje o vytvoření „své“ nejlepší databáze. Existuje tak celá řada NoSQL a jejich využití je tak potřeba zvážit pro konkrétní účely aplikačního řešení.

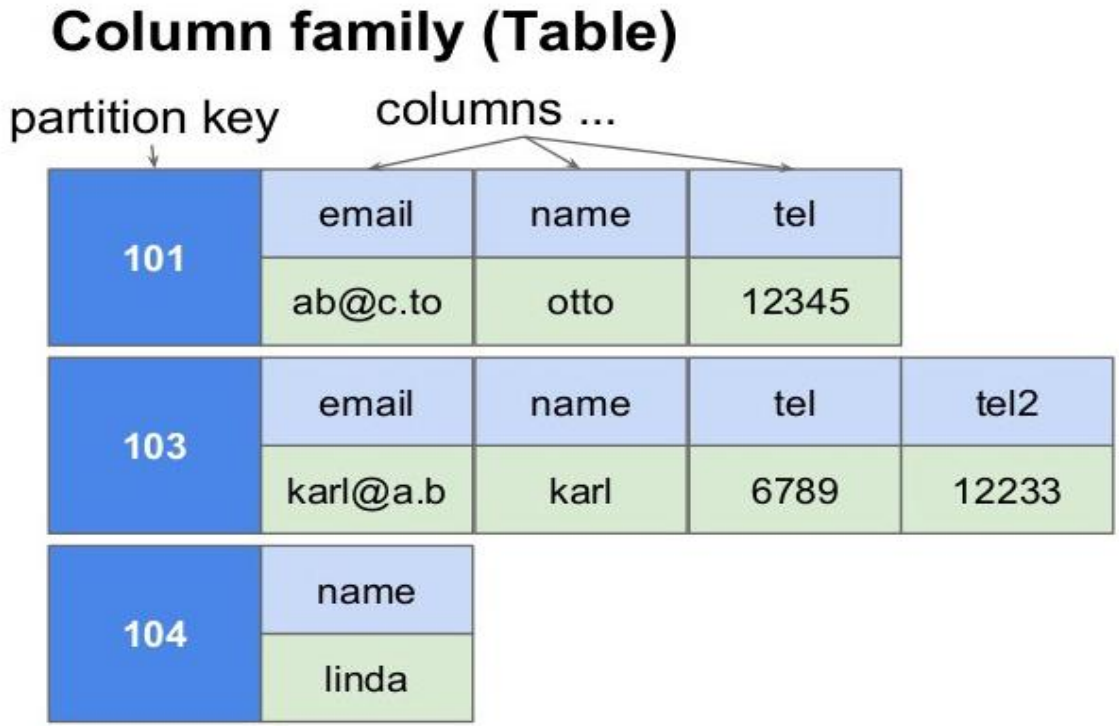
Způsoby jednotlivých provedení jsou:

- Dokumentové databáze – data jsou organizována v dokumentech. Tyto dokumenty mohou být například ve formátu XML nebo JSON. Získávání dat z jednotlivých dokumentů je jednoduché a nevyžaduje komplexní propojování tabulek jako je tomu u RDBMS [1]. JSON ukládá data jako objekt, který může obsahovat celou řadu údajů jako jsou například jméno, příjmení, datum narození nebo i další pod objekty. Velmi často je využíván hlavně ve webových API.

```
[
  {
    "id": 1,
    "jmeno": "Adam",
    "mesto": "Zlin",
    "vek": 22,
    "pohlavi": "muz",
    "email": "adam@email.cz",
    "kamaradi": [
      { "jmeno": "Filip", "id": 2 },
      { "jmeno": "David", "id": 3 },
      { "jmeno": "Pepa", "id": 4 },
      { "jmeno": "Petr", "id": 5 }
    ],
    "zaliby": ["sport", "programovani", "spanek"]
  },
  {
    "id": 2,
    "jmeno": "Filip",
    "mesto": "Zlin",
    "vek": 20,
    "pohlavi": "muz",
    "email": "filip@email.cz",
    "kamaradi": [
      { "jmeno": "Adam", "id": 1 },
      { "jmeno": "Roman", "id": 26 },
      { "jmeno": "Marek", "id": 12 }
    ],
    "zaliby": ["posilovani", "serialy", "beerpong"]
  }
]
```

Obrázek 11 Příklad JSON – Dokumentová databáze

- Sloupcové databáze – data jsou ukládány ve sloupcích. Každý řádek obsahuje klíč, kterému se přiřadí záznamy dat reprezentované sloupci, kterých může být mnoho (Obrázek 12). Každý sloupec obsahuje jméno a samotné hodnoty související s tímto jménem. Díky tomuto způsobu organizace dat umožňuje sloupcová databáze flexibilitu a rychlost. Je tak možné data v průběhu měnit a různě organizovat. Nemusí tak být dán pevný návrh a každý záznam může vypadat jinak. Tento přístup je rozdílný od RDBMS, kde je návrh pevně stanoven.



Obrázek 12 Sloupcová databáze [29].

Na tomto obrázku jsou znázorněny tři záznamy v databázi, které se všechny odlišují svým klíčem a svými sloupci se záznamy. Zatímco záznam s klíčem „101“ obsahuje tři sloupce s hodnoty emailu, jména a telefonního čísla, tak například záznam s klíčem „104“ obsahuje pouze jméno, to charakterizuje flexibilitu návrhu a různorodost v databázi.

- Databáze typu klíč-hodnota – nejjednodušší z NoSQL databází. Ukládá data v podobě klíče a na něj navazující hodnoty (Obrázek 13). V případě, že známe klíč, tak můžeme získat hodnoty. Unikátní klíče jsou typicky realizovány jako řetězce a navazující hodnoty jsou realizovány různými datovými typy, kterými mohou být celé čísla, řetězce, pravdivostní hodnoty anebo pole hodnot. Tento jednoduchý způsob umožňuje velmi snadný přístup k jednotlivým datům.

Phone directory

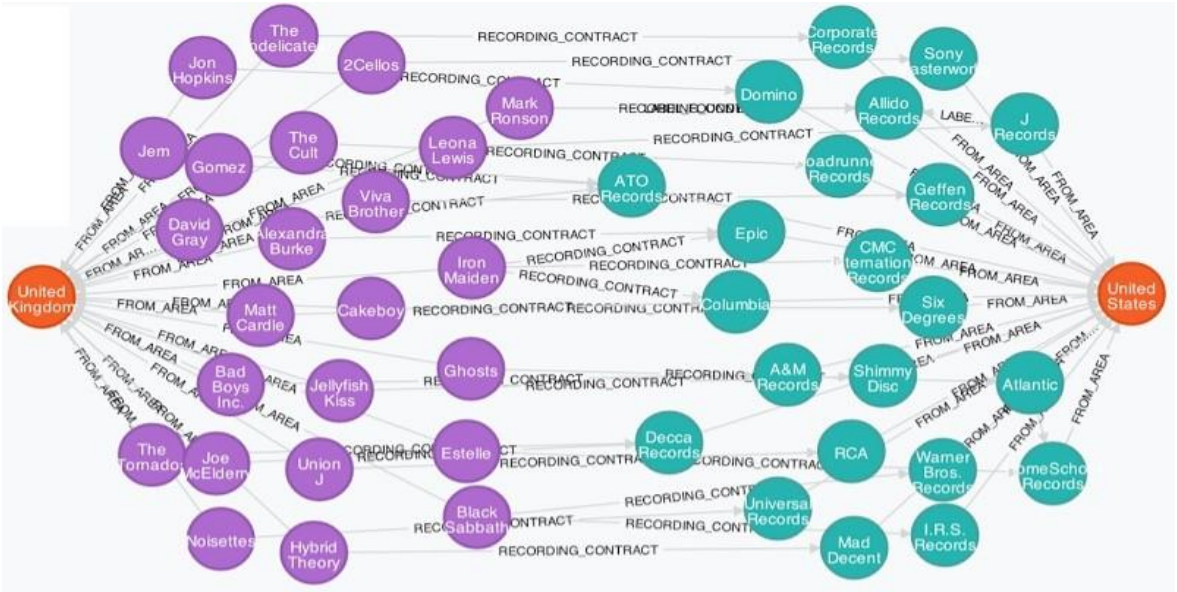
Key	Value
Paul	(091) 9786453778
Greg	(091) 9686154559
Marco	(091) 9868564334

MAC table

Key	Value
10.94.214.172	3c:22:fb:86:c1:b1
10.94.214.173	00:0a:95:9d:68:16
10.94.214.174	3c:1b:fb:45:c4:b1

Obrázek 13 Klíč-hodnota databáze [30].

- Grafické databáze – na rozdíl od RDBMS databází, které organizují strukturu do propojených tabulek vytvářejí grafické databáze propojenou síť charakterizující objekty a jejich vztahy mezi dalšími objekty. Taková síť může být velmi složitá a dokáže nabývat extrémních rozměrů. Databáze je tak navržená na komplexní vtahy. Data jsou uložena jako hrany a uzly. Uzly reprezentují entity a hrany reprezentují vztahy mezi těmito entity [2].

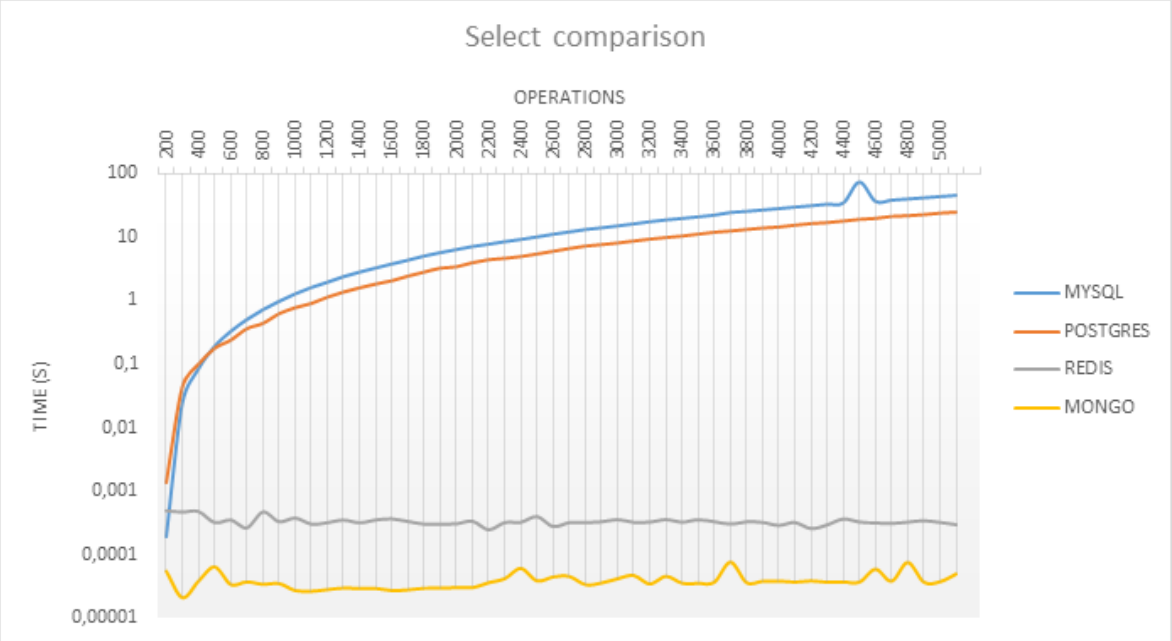


Obrázek 14 Grafická databáze [31].

2.1.2 Výhody

- Rychlost

NoSQL databáze jsou výhodné díky své rychlosti k přístupu k datům. Ve velkých aplikacích reálného světa je potřeba velké množství dat a začíná být velmi obtížné zpracovávat je způsobem, jak jsme zvyklí – relační databází. Dotazování se přes velké množství tabulek může trvat velmi dlouho a SQL příkazy mohou být velmi složité [4].

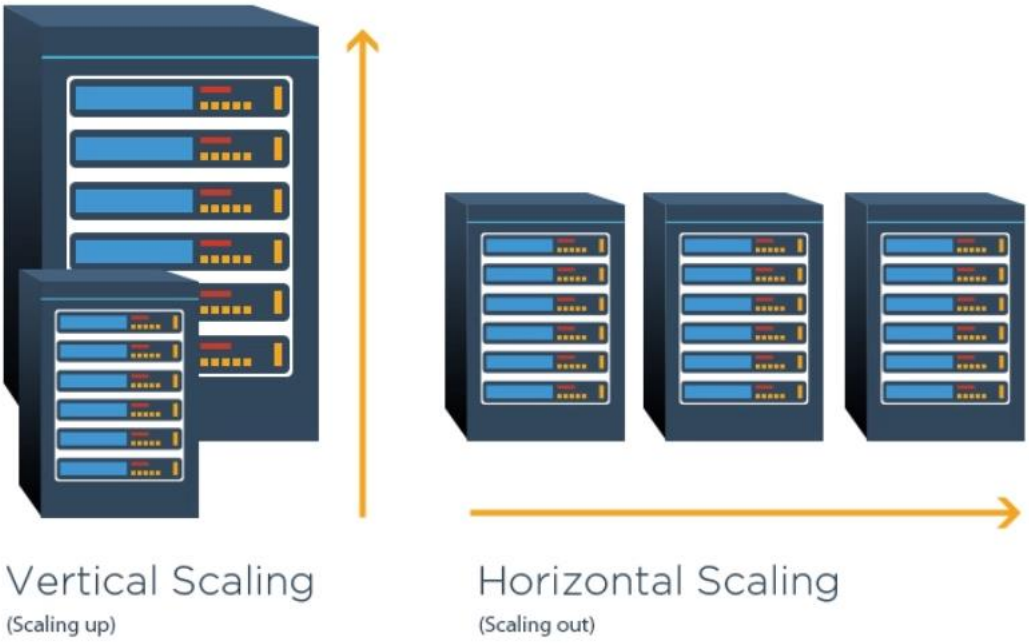


Obrázek 15 Porovnání rychlostí operací jednotlivých databází [4].

Z obrázku lze vidět porovnání rychlostí operací jednotlivých populárních databází. Relační databáze jako MySQL a PostgreSQL si vedly poměrně hůře v porovnáním s NoSQL. Na větší množství operací potřebovaly až desítky sekund. Oproti tomu dokumentová databáze MongoDb a databáze Redis využívající klíč-hodnotu měly téměř konzistentní časový přístup, a to i při větším množství operací.

- Horizontální růst

NoSQL databáze mají schopnost růst horizontálně, to znamená, že data nerostou do hloubky, ale do šířky (Obrázek 16). Databáze jsou tak schopné fungovat na více oddělených strojích, zatímco pro SQL je potřeba lepší hardware – silnější a výkonnější, což může být časem velmi drahé. Relační databáze tak ideálně potřebuje jeden superpočítač s větším výkonem.



Obrázek 16 Horizontální a vertikální růst [32].

- Flexibilní návrh

Další výhodou může být flexibilní návrh databáze. Zatímco u relační databáze je stanovený návrh s danými tabulkami, vztahy mezi nimi a snahy normalizovat způsobem, kterým by nedocházelo k redundantním datům, tak u NoSQL databáze se může kdykoliv přidat nový sloupec, položka nebo objekt [1]. Změny tedy mohou nastat téměř okamžitě bez složitého předělávání.

2.1.3 Nevýhody

- Omezená podpora transakcí

Oproti relačním databázím nepodporují NoSQL databáze transakce a jejich vlastnost ACID (Atomicity-Consistency-Isolation-Durability), jež je hlavním aspektem transakcí. NoSQL neobsahují všechny tyto vlastnosti nebo někdy dokonce žádnou. Data tak vypadají méně důvěryhodně [3]. Při zápisu a čtení tak nemusíme okamžitě dostat stejná data. To nemusí být problém například při zobrazování příspěvků na sociálních sítích, nicméně pro transakci velmi velké částky to problém může být [1].

- Omezená podpora

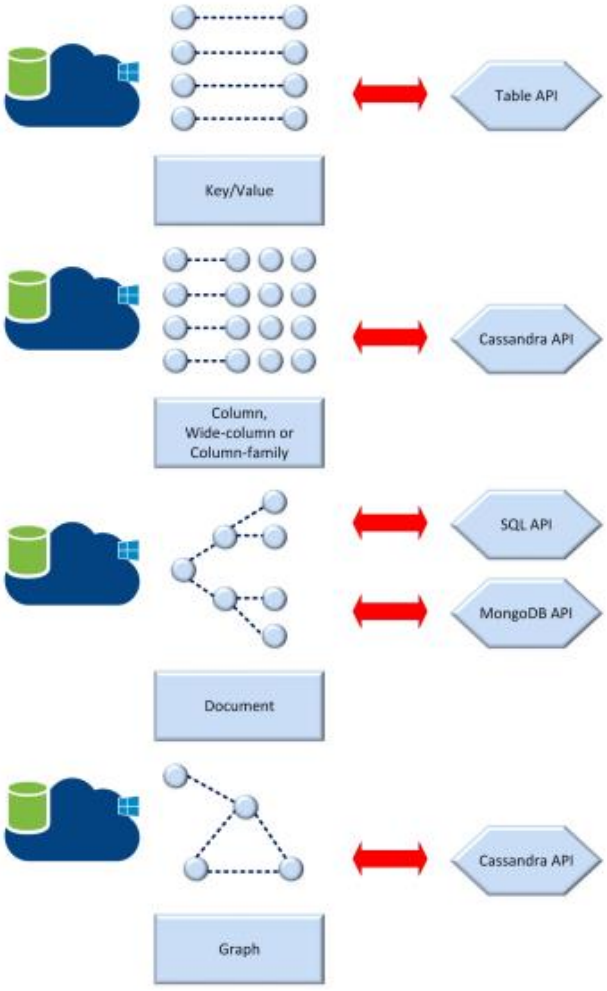
Relační databáze tady byly mnohem dříve a spousta lidí si taky představí databázi standardním způsobem – jako relační. Existují již více než 50 let a mají velké množství podpory, expertů a zdrojů, z kterých lze čerpat. NoSQL oproti tomu je novější a snaží se dělat věci jiným způsobem než relační databáze, může proto být obtížnější a dražší najít někoho na konzultaci projektu [5].

- Omezenost dotazů

Dotazy v NoSQL nejsou tak komplexní jako v SQL, nelze je úplně optimalizovat. Nemají univerzální dotazovací jazyk jako je SQL. Spojování tabulek pomocí joinů jako v SQL tak není možné a je potřeba využít jiné alternativy [6].

2.2 KLÍČOVÉ VLASTNOSTI COSMOS DB

- Globální distribuce – data z Cosmos DB je snadné šířit po celém světě.
- Schopnost více modelů – Cosmos DB má schopnost využití různých NoSQL modelů jako jsou dokumenty, klíč-hodnota, sloupcové a grafické databáze (Obrázek 17), které lze dokonce využít i všechny dohromady. Využívá různých API pro práci s populárními databázemi jako jsou Cassandra, MongoDB nebo Gremlin. Můžeme taky využít SQL pro práci s modelem ve formátu dokumentů. Sami tak můžeme vybrat, co je pro nás vhodné a jakým způsobem s databází budeme pracovat [10].



Obrázek 17 Modely v Cosmos DB [10].

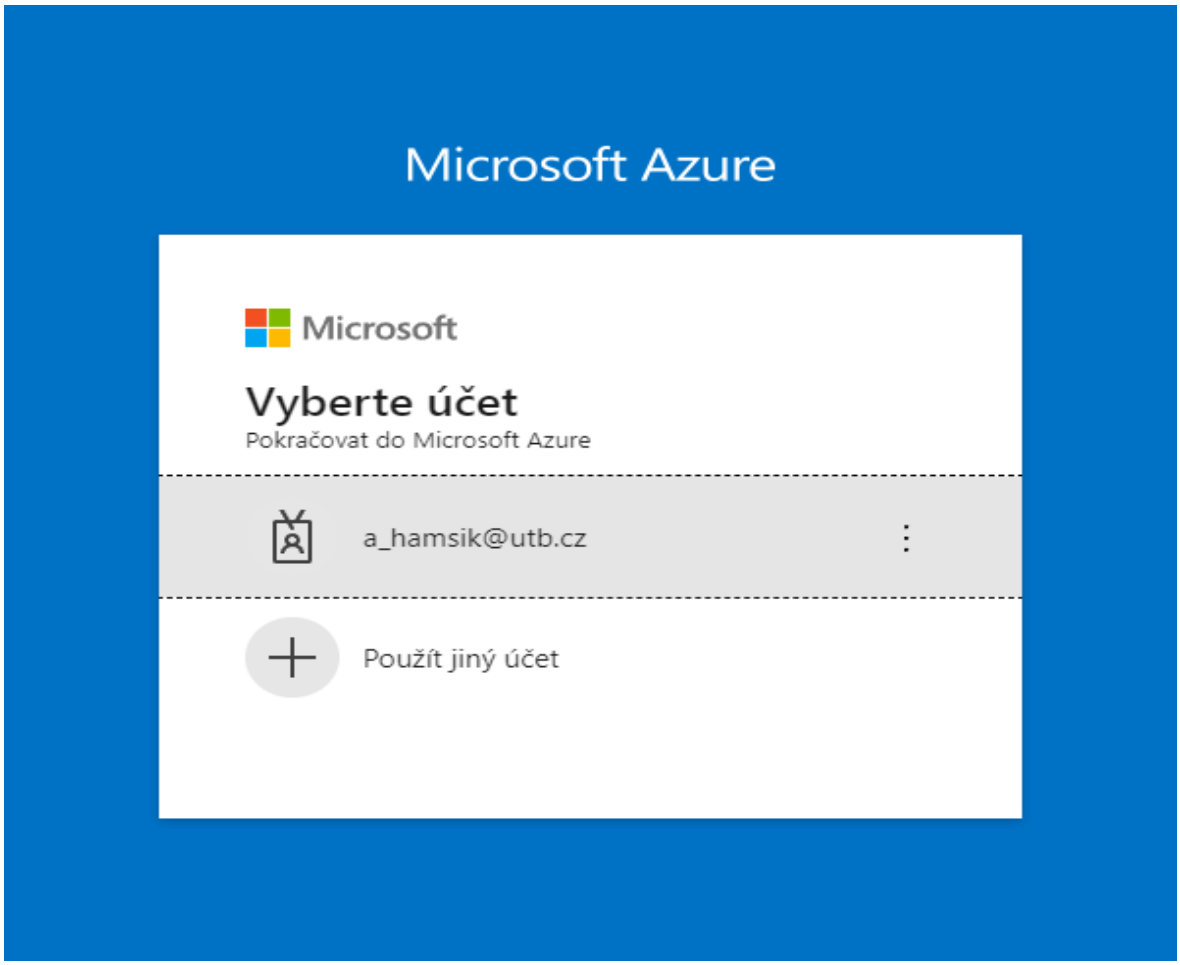
- Vysoká propustnost a dostupnost – možné zvětšit nebo zmenšit kapacitu databáze dle libosti přidáním nebo odebráním fyzických uzlů, které obsahují data. Aplikace tak splňuje požadavky na výkon. Jednotlivé operace jsou vyjádřeny v RU (Request Unit) – jednotky požadavku. RU lze měnit, ale v Azure mají svůj ceník [7]. Služba pracuje s vysokou dostupností. V jedné regionální oblasti zaručuje dostupnost až 99.99 % SLA (Service Level Agreement) a 99.999 % SLA při distribuci ve více regionech [10].
- Jedinečnost klíčů – v Azure Cosmos DB se nacházejí jedinečné klíče a tím je zabráněno vytváření duplicit v rámci jednotlivých logických oddílů. Tyto logické klíče jsou nastavovány při vytváření kontejneru a nelze je již měnit. Můžou být složeny z jednoho nebo více atributů. Tím se může vytvořit logický oddíl, zajišťující jedinečnou kombinaci v databázi. Například spojení kombinace jména, příjmení nebo emailové adresy [48].

2.3 INTEGRACE ASP.NET CORE S COSMOS DB

Propojením ASP.NET Core a Cosmos DB můžeme vytvořit výkonnou webovou aplikaci s rychlou databází.

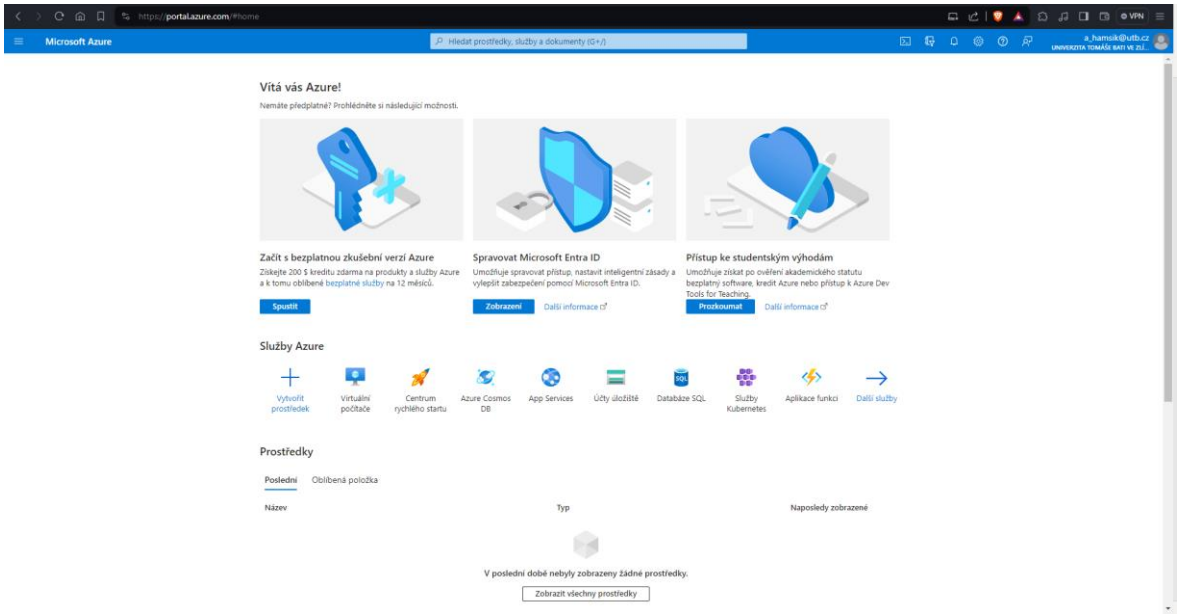
Pro využití Cosmos DB je potřeba vytvořit si účet na Microsoft Azure (Obrázek 20). Azure je cloudová služba, která obsahuje celou řadu služeb pro vývoj, testování aplikací, práci s daty a jejich analýzu [22]. Cloud obsahuje servery, na kterých běží nejrůznější databáze. Tyto servery jsou rozmístěny v data centrech na různých místech po celém světě. Uživatelé nebo společnosti tak nemusejí provozovat své vlastní fyzické servery, ale využívají služeb Azure [23].

Do Azure je možné se přihlásit již existujícím účtem Microsoft nebo vytvořit nový účet (Obrázek 18).



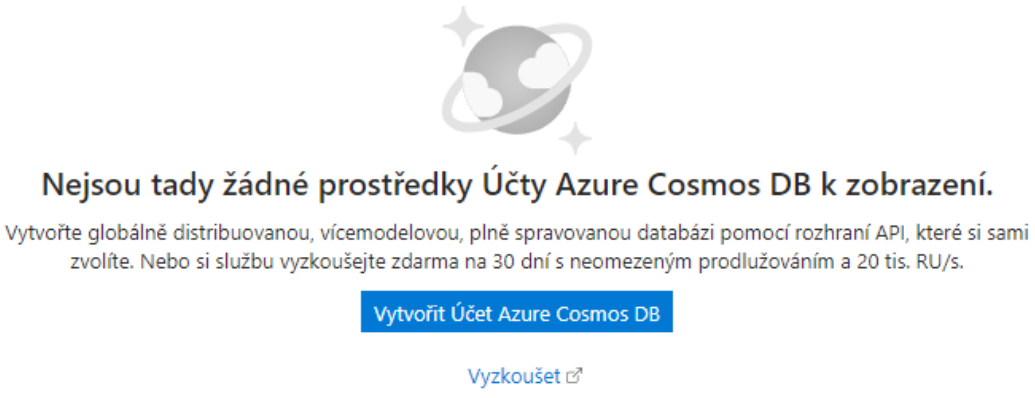
Obrázek 18 Přihlášení do Azure

Portál poté vyžaduje vytvoření účtu Azure. Abychom mohli přistupovat k jeho vice než 200 službám, je potřeba vytvořit bezplatný nebo studentský účet (Obrázek 19).



Obrázek 19 Portál Azure

Při vytváření bezplatného účtu je potřeba zadat své osobní údaje a potvrzení karty. Azure si účtuje za své služby na základě využití. V případě, že bychom přečerpali bezplatné prostředky pro naší Cosmos DB službu, museli bychom si další zaplatit.

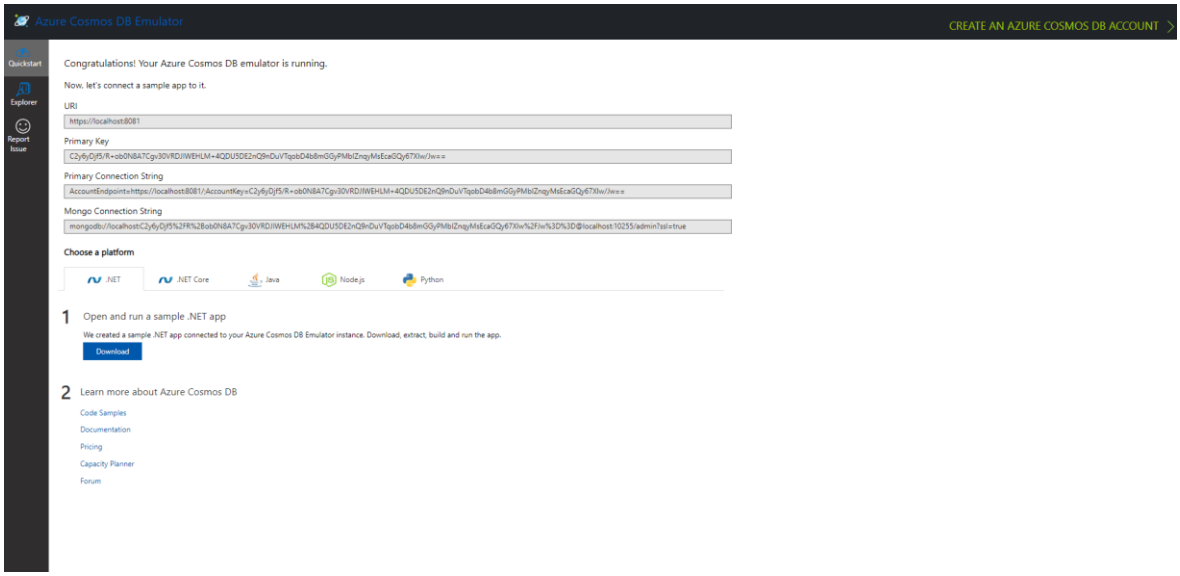


Obrázek 20 Účet Azure Cosmos DB

Alternativou je vyzkoušet službu zdarma na 30 dní neomezeně. Po zvolení je potřeba vybrat API pro náš účet, tento výběr není možné dále měnit.

Další možností je využití bezplatného emulátoru (Obrázek 21).

Tento emulátor simuluje cloudové prostředí Azure na lokálním počítači. Neúčtuje si tak žádné poplatky a je výhodný pro snadné testování a vývoj. Má nicméně několik nevýhod a nemá tak široké použití jako má cloud. Podporuje například jen 2 API – NoSQL API a MongoDB API. Nelze pro něj vygenerovat nový klíč za běhu nebo využívat nekonečně mnoho kontejnerů s neomezeným RU. V případě spokojenosti s aplikací je potom možné se přesunout na účet Azure [24].

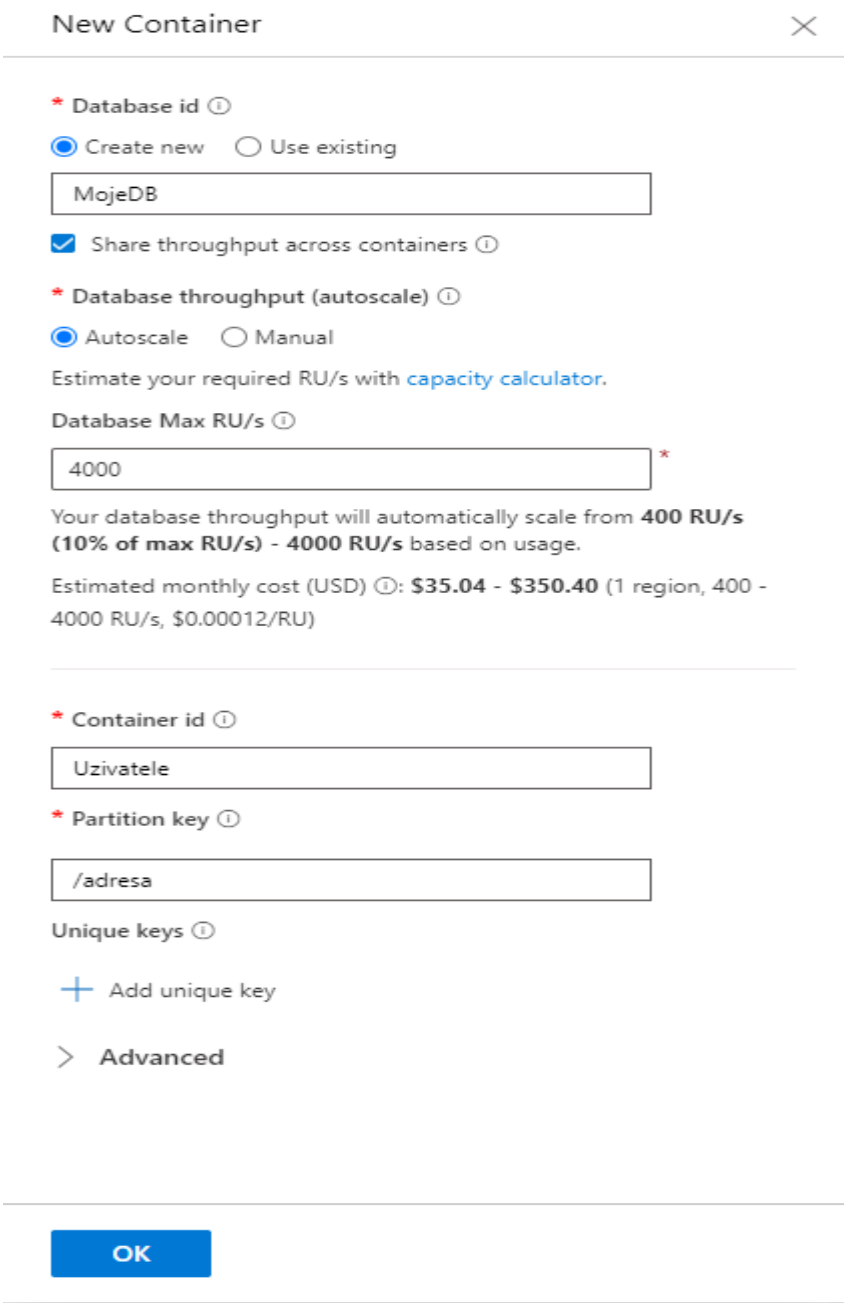


Obrázek 21 Azure Cosmos DB Emulátor

V emulátoru můžeme vytvořit novou databázi s kontejnery (Obrázek 22). Kontejnerů může být v databázi více, představují sadu jednotlivých dat.

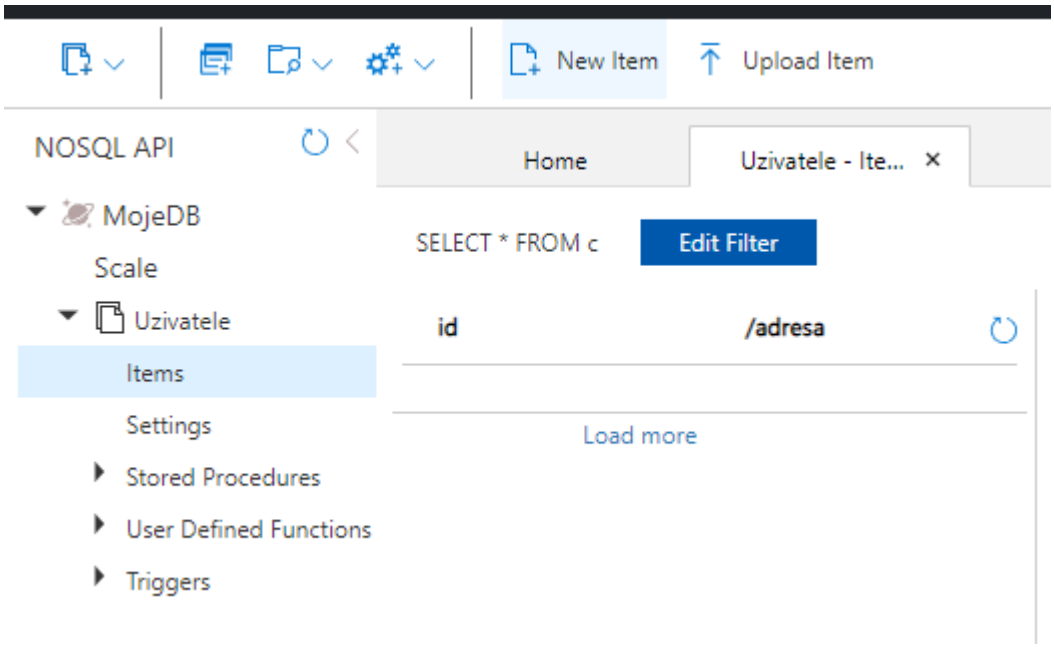
Při vytváření databáze musíme určit identifikátor databáze a nastavit propustnost na základě našeho využití. Propustnost můžeme nastavit automaticky nebo manuálně. Při automatickém nastavení bude systém za nás měnit RU na základě našeho použití. Při čtení dokumentu, který má 1 kB (kilobyte) se využije 1 RU.
Poté je potřeba nastavit maximální RU do kterého se bude propustnost ze všech kontejnerů v databázi měnit.

Po vytvoření databáze v ní můžeme vytvořit různé kontejnery. Kontejner potřebuje unikátní identifikátor a partition key. Partition key určuje data, podle kterých se vytváří logické oddíly. Pokud je například partition key adresa, tak se vytvoří logické oddíly na základě adres všech záznamů. Pokud se nějaká adresa opakuje, bude více záznamů se stejnou adresu spadat pod jeden logický oddíl. Takový přístup umožňuje lepší škálovatelnost a paralelní zpracování [25].



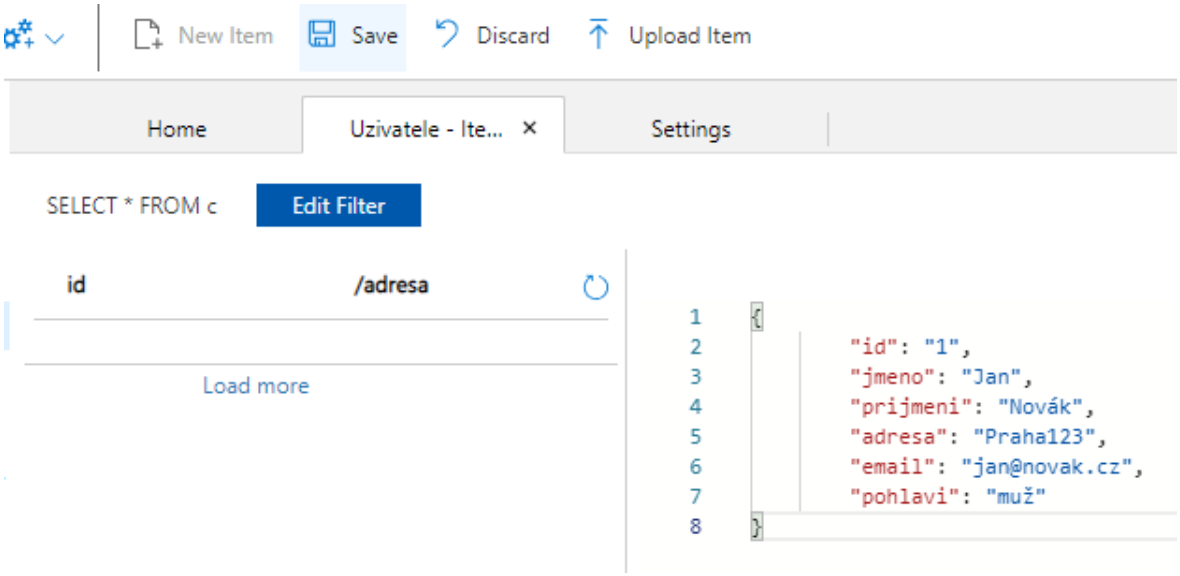
Obrázek 22 Vytvoření databáze a kontejneru v emulátoru

Po vytvoření databáze s kontejnerem můžeme následně provádět základní operace s daty jako je vytváření (Obrázek 23), čtení (Obrázek 26), mazání a upravování (Obrázek 24) přímo v prostředí emulátoru.



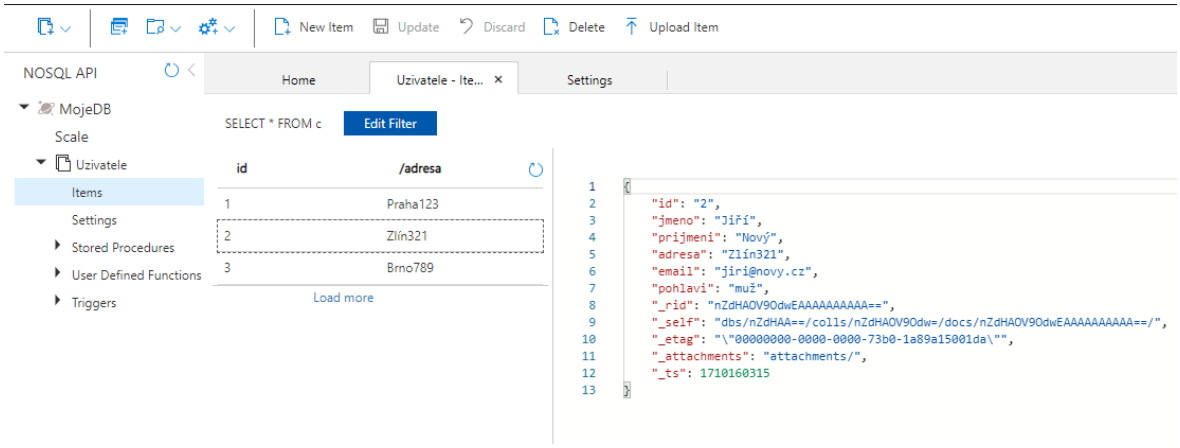
Obrázek 23 Vytvoření nové položky

Po kliknutí na „New Item“ můžeme přidat náš dokument a uložit ho.



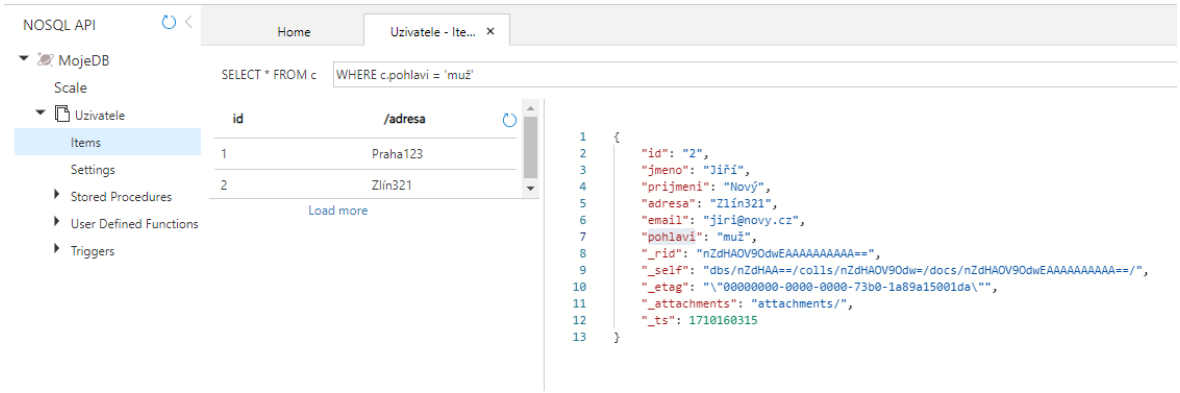
Obrázek 24 Uložení nové položky

Jednotlivé záznamy jsou pak viditelné pod sebou a můžeme na nich provádět různé dotazy. NoSQL API využívá formátu dokumentů JSON a umožňuje volání SQL na nich.



Obrázek 25 Vytvořené záznamy v kontejneru uživatelů

Je tak možné například vybrat všechny záznamy, kde pohlaví má hodnotu „muž“.



Obrázek 26 Výběr záznamů na základě dotazu

3 REŠERŠE HTML FRAMEWORKŮ

Vývoj frontendu může být zdlouhavý a náročný. Snažit se vytvořit uživateli přívětivé a přizpůsobivé rozhraní je důležité, a proto je potřeba vybrat vhodný HTML framework.

HTML framework je sada předepsaného kódu obsahující prvky jako například tlačítka, ikony, formuláře nebo další. Zjednodušují tak práci, protože mají určité standardy a jsou velmi přizpůsobivé. Velmi často jsou taky responzivní, takže změna formátu stránky neovlivní vzhled stránky, což je velmi důležité v době různých velikostí zařízení [8].

Frameworky mají v sobě součást HTML, CSS a JavaScript kódu. Kombinují tak komponenty, jejích přizpůsobení různým stylům, barvám, postavení na stránce a reakci uživatele s nimi. HTML obsahuje základní strukturu prvků, jazyk CSS definuje styl a vzhled, JavaScript přidává funkcionalitu a manipulaci se stránkou.

Obrázek 27 Znázornění klíčových součástí HTML frameworku [37].

3.1 PŘEHLED MODERNÍCH FRAMEWORKŮ

V této kapitole jsou popsány některé z populárních moderních HTML frameworků a jejich vlastnosti, které nabízejí.

3.1.1 Bootstrap

Bootstrap je jeden z nejpoužívanějších moderních frameworků. Je využíván na více než 100 tisíc stránkách na světě [9]. Nabízí důležité komponenty pro webový vývoj. Mezi jeho klíčové vlastnosti patří:

- JavaScriptové komponenty – Bootstrap má řadu již načtených JavaScriptových komponent, umožňujících snadné otevírání vyskakujících oken, jako jsou například modaly, dropdown prvky, tooltipy anebo alerty – tedy zobrazení nápovědy, upozornění nebo výběru z možností na stránce [14].
- Responzivní návrh – je velmi populární pro svůj responzivní design. Komponenty se automaticky přizpůsobují zobrazení stránky a jednotlivým zařízením s různými rozlišeními.
- Rozšiřitelnost – má možnost přidání dalších komponent a funkcionalit do projektu. Díky velké komunitě vývojářů existuje spousta unikátních pluginů.
- Kompatibilita v rámci různých prohlížečů – Bootstrap je kompatibilní s velkým množstvím prohlížečů a jejích nejnovějšími verzemi. Spousta uživatelů používá různé zařízení s různými webovými prohlížečemi a je tak potřeba dodržet kompatibilitu [15].

Obrázek 28 Logo Bootstrap [16].

3.1.2 Foundation

Foundation je další populární framework vhodný pro jakékoliv zařízení. Je přizpůsobivý a responsivní. Je pravidelně aktualizován o nové vlastnosti [11]. Je to open source, založený společností ZURB a udržovaný dobrovolníky [13].

Vlastnosti:

- Možnost úpravy podle vlastních potřeb – vývojáři si můžou upravit podle svých potřeb jednotlivé komponenty, což vytváří jedinečnost webových stránek [17].
- Podpora komunity – Foundation má silnou základnu podporovatelů, kteří pomáhají s jednotlivými problémy.
- Mřížkový systém – flexibilní, responzivní, užitečný při použití na různých zařízení o různých rozlišeních.
- Znovupoužitelné komponenty – často používané komponenty jako jsou tlačítka, formuláře, výběr z možností s vylepšenými možnostmi [13].
- Jednoduchý – vhodný pro vývojáře, kteří již znají HTML a CSS.

Obrázek 29 Logo Foundation [11].

3.1.3 Metro UI

MetroUI je open-source framework pro vývoj s HTML, CSS a JavaScriptem. Je zajímavý tím, že vývojář nemusí znát pro vývoj JavaScript. Je responsivní a využívá způsobu mobile-first – vyvíjeno je první na mobilní zařízení a až později se adaptuje na větší zařízení [18].

Vlastnosti:

- Rychlý začátek – MetroUI je velmi snadný pro přidání do projektu. Pro přidání CSS stačí přidat odkaz na knihovnu a budou využity jeho styly. Pro přidání JavaScriptu přidáme odkaz na scripty [26].
- Technická podpora – tato možnost je nicméně placená [33].
- Velká řada komponent – nabízí širokou řadu předdefinovaných komponent, které mohou být snadno použity podle potřeb projektu [18].
- Barevné schéma a motivy – obsahuje spoustu barevných schémat a motivů pro personalizaci webových stránek a jejich vzhledu [18].
- Kompatibilita s prohlížeči – MetroUI je kompatibilní s většinou moderních prohlížečů.

Obrázek 30 Logo Metro UI [40].

3.1.4 Semantic UI

Semantic UI je moderní framework, který se soustředí na lidsky přívětivé použití HTML. Byl navržen, aby byl přátelský a intuitivní ve snaze rychlejšího a efektivního vývoje uživatelských rozhraní [38].

Vlastnosti:

- Responzivní mřížkový systém – podobně jako Boostrap a Foundation, tak i Semantic UI nabízí flexibilní a responzivní mřížkový systém, který umožňuje snadné a efektivní uspořádání obsahu na různé velikosti obrazovek [38].
- Komunitní podpora – Semantic UI má silnou a aktivní komunitní podporu na GitHubu, kde i noví uživatelé přispívají k vývoji nových komponent a funkcí [39].
- Sémantický a intuitivní design – využívá přirozeného jazyka pro třídy a strukturu, udržuje tak čitelnost kódu a je tak rychlejší na naučení pro nové uživatele [38].
- Integrace s knihovnami – je snadné jej integrovat s jinými JavaScriptovými knihovnami a framework, což usnadňuje vývoj složitějších aplikací.
- Tématizace a přizpůsobení – poskytuje pokročilé možnosti přizpůsobení témat, což umožňuje vytvářet specifický vzhled na základě požadavků.

Obrázek 31 Logo Semantic UI [41].

3.1.5 Materialize

Materialize je moderní framework založený na Material Designu od Google. Klade si za cíl umožnit vývoj uživatelsky přívětivé a vizuálně atraktivní webové aplikace s použitím předdefinovaných komponent inspirovaných materiál designem [42].

Vlastnosti:

- Koncept Material Design - Materialize vychází z jazyka Material Design, který zdůrazňuje použití hloubkových efektů, jako jsou světelné nebo stínové efekty. Tím se dosáhne realističtějšího uživatelského rozhraní [42].
- Responzivní mřížka – responzivní mřížkový systém, přizpůsobený na různě velká zařízení a obrazovky, aplikace tak vypadá dobře na jakémkoliv zařízení-
- Snadná customizace – možnost snadného přizpůsobení témat, komponent a barev na základě specifických designových požadavků [43].
- JavaScript komponenty – podporuje širokou škálu komponentů JavaScriptu pro dynamické efekty a funkcionalitu.
- Parallax – Parallax efekt pro obrázkové komponenty. Vytváří působivý efekt hloubky a pohybu při posouvání stránky [42].

Obrázek 32 Logo Materialize [44].

3.2 SROVNÁNÍ JEDNOTLIVÝCH FRAMEWORKŮ

Na základě jednotlivých frameworků a jejich vlastností byla pro porovnání vytvořena tabulka (Tabulka 2).

Tabulka 2 Porovnání jednotlivých vlastností frameworků

Vlastnost / Framework	Bootstrap	Metro UI	Semantic UI	Foundation	Materialize
Komponenty	Rozsáhla knihovna komponent	Specifické pro styl Metro	Sémantické rozsáhlé komponenty	Profesionální, minimalistické	Inspirované materiál Designem
JavaScript Knihovny	Závislost na jQuery	Méně závislostí, podpora jQuery	jQuery závislosti, integrované chování	jQuery nezávislé, modulární funkce	Široká škála JS komponent
Dokumentace a podpora	Rozsáhlá dokumentace a komunita	Přehledná, komunita menší	Podrobná, aktivní komunita	Rozsáhlá dokumentace a silná komunita	Dobrá dokumentace a rozvoj komunity
Responzivní design	Integrované třídy pro různé velikosti	Responzivní zobrazení pro všechna zařízení	Responzivní grid a UI komponenty	Mobile-first, responzivní zobrazení	Responzivní design, automatické přizpůsobení

V tabulce jsou porovnávány důležité vlastnosti jednotlivých frameworků a jak se mezi sebou liší.

3.3 PRAKTICKÉ VYUŽITÍ JEDNOTLIVÝCH FRAMEWORKŮ

Při vytváření nového projektu je důležité vybrat vhodný framework pro specifické požadavky uživatele. Na této stránce je popsán výběr jednotlivých frameworků pro projekty.

Bootstrap

- Projekty s omezenou časovou lhůtou – v případě, že náš vyvíjený projekt je pod časovým nátlakem a prioritou není vytvoření vizuálně unikátního designu, může být Boostrap vhodným řešením. Rozsáhlá knihovna stylů umožňuje sestavení v rekordním čase [45].
- Projekty s důrazem na backend a interaktivitu – pro projekty, kde je důraz kladen na funkčnost, interaktivní prvky a designové aspekty jsou vedlejší, je Boostrap vhodným řešením [45].
- Prototypování webů – při vývoji první verze webové stránky, která má dále projít celou řadou změn a úprav [45].

Metro UI

- Vhodný pro projekty, které vyžadují jeho specifický Metro styl, který je inspirovaný Windows rozhraním.
- Projekty, které vyžadují mobile-first implementaci, to znamená, že se vyvíjí nejprve na mobilní zařízení [18].

Semantic UI

- Optimální pro aplikace s vysokou úrovní interaktivity a vizuálního designu [46].
- Snadná integrace s JavaScriptovými knihovnami umožňuje flexibilitu různých typů webových aplikací [46].
- Projekty vyžadující sémantický přístup k designu [46].

Foundation

- Komplexní webové aplikace, které vyžadují pevný a stabilní základ.
- Stejně jako Boostrap i Foundation může být využit pro rychlý a snadný vývoj [47].

Materialize

- Projekty vyžadující estetický design, který využívá prvků Material Designu [42].
- Aplikace s více soubory a textury, komplexní 3D scény.

II. PRAKTICKÁ ČÁST

4 NÁVRH A IMPLEMENTACE APLIKACE

Cílem tohoto projektu je vytvořit webovou aplikaci s využitím technologií ASP.NET Core a databázové služby Cosmos DB. Jako projekt byla vybrána aplikace pro správu úkolů, plánování událostí a osobního portfolia.

4.1 SPECIFIKACE PROJEKTU

4.1.1 Účel aplikace

Hlavním účelem naší aplikace je poskytnout uživateli snadné a intuitivní řešení organizace jejich každodenního života, práce a svého osobního rozvoje. Aplikace efektivně kombinuje správu úkolů, událostí a s nimi spojené osobní portfolio pro zobrazení svých statistik dokončených nebo rozdělaných úkolů a nadcházejících událostí. Díky funkcím jako je osobní kalendář nebo zobrazení dokončených úkolů tak umožňuje uživatelům zvýšit svou produktivitu a motivuje je ke konání činností a zvyšuje organizovanost a efektivitu jejích životů.

4.1.2 Cílová skupina

Cílovou skupinou naší aplikace je široké spektrum uživatelů. Jedná se ale především o ty, kteří nechtějí žít v nepořádku, nebýt neinformovaní a žít ve strachu zmeškání důležitých nadcházejících událostí. Zahrnuje tak jednotlivce, kteří chtějí udržet svůj profesní anebo osobní život organizovaný a plánovaný v souladu s jejich ambicemi anebo ambicemi jejich nejbližších – přátel, rodiny nebo kolegů.

4.1.3 Funkční požadavky

V této sekci jsou podrobněji specifikovány funkční požadavky naší aplikace (Obrázek 33). Definují konkrétní chování, které aplikace musí splňovat a operace, které uživatel může provádět, a na které aplikace reaguje. Každý požadavek má jasný popis, který charakterizuje jeho funkcionalitu a jeho očekávané vstupy a výstupy. Jsou tak zásadní pro popis aplikace a jejích vlastností.

Funkční požadavky byly rozděleny do jednotlivých oblastí, ke kterým se vztahují, jako například „Autentizace“ pro funkcionalitu přihlašování anebo „Správa úkolů“ pro příslušné funkce pro práci s úkoly.

4.1.4 Jednotlivé sekce s požadavky

Autentizace – přihlašování a vytváření nových uživatelů a související operace

- REQ-AUT-1: Uživatel se může registrovat
- REQ-AUT-2: Uživatel se může přihlásit pomocí username a hesla
- REQ-AUT-3: Uživatel se může odhlásit

Dokončené úkoly – přehledné zobrazení uživateli jeho dokončených úkolů

- REQ-DÚ-1: Zobrazení grafu s dokončenými úkoly za aktuální rok
- REQ-DÚ-2: Zobrazení statistiky dokončených úkolů (počet, priorita, zadavatel)
- REQ-DÚ-3: Poskytnutí seznamu všech dokončených úkolů

Funkce nepřihlášených uživatelů – odlišení od přihlášených uživatelů a znemožnění využívat aplikaci plnohodnotným způsobem

- REQ-NU-1: Možnost přidávání veřejných úkolů
- REQ-NU-2: Možnost aktualizace úkolů na základě klíče
- REQ-NU-3: Možnost smazání všech veřejných úkolů na základě klíče

Kalendář – možnost pro uživatele přívětivě v čase procházet své úkoly a události

- REQ-K-1: Zobrazení dokončených a privátních úkolů
- REQ-K-2: Navigace kalendářem (procházení měsíců a roků)

Profil uživatele – zobrazení uživatelských profilů a dat souvisejícími s nimi

- REQ-PU-1: Možnost aktualizace osobních údajů
- REQ-PU-2: Zobrazení profilů přátel, včetně jejich posledních 5 úkolů pro přátele

Správa úkolů – umožňuje uživatelům spravovat jejich úkoly a jejich manipulaci

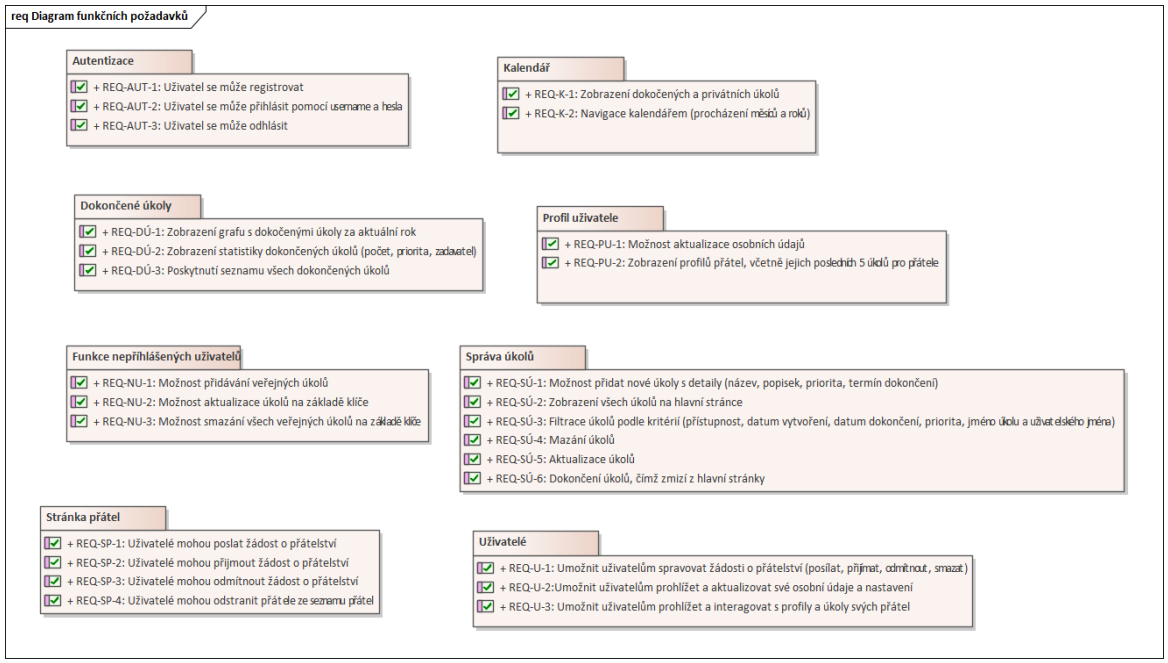
- REQ-SÚ-1: Možnost přidat nové úkoly s detaily (název, popisek, priorita, termín dokončení)
- REQ-SÚ-2: Zobrazení všech úkolů na hlavní stránce
- REQ-SÚ-3: Filtrace úkolů podle kritérií (přístupnost, datum vytvoření, datum dokončení, priorita, jméno úkolu a uživatelského jména)
- REQ-SÚ-4: Mazání úkolů
- REQ-SÚ-5: Aktualizace úkolů
- REQ-SÚ-6: Dokončení úkolů, čímž zmizí z hlavní stránky

Stránka přátel – stránka umožňuje uživatelům rozšiřovat a spravovat své sociální sítě tím, že poskytuje nástroje pro vzájemné propojení mezi uživateli.

- REQ-SP-1: Uživatelé mohou poslat žádost o přátelství
- REQ-SP-2: Uživatelé mohou přijmout žádost o přátelství
- REQ-SP-3: Uživatelé mohou odmítnout žádost o přátelství
- REQ-SP-4: Uživatelé mohou odstranit přátele ze seznamu přátel

Uživatelé – poskytuje přehled o tom, jakým způsobem mohou uživatelé spravovat svůj profil a jeho zobrazení a interakci s ostatními uživateli.

- REQ-U-1: Umožnit uživatelům spravovat žádosti o přátelství (posílat, přijímat, odmítnout, smazat)
- REQ-U-2: Umožnit uživatelům prohlížet a aktualizovat své osobní údaje a nastavení
- REQ-U-3: Umožnit uživatelům prohlížet a interagovat s profily a úkoly svých přátel



Obrázek 33 Diagram funkčních požadavků vytvořených v programu Enterprise Architect

4.1.5 Nefunkční požadavky

Nefunkční požadavky tvoří důležitou součást aplikace (Obrázek 34). Definují kvalitativní atributy jako je výkon, bezpečnost nebo uživatelská přívětivost, které zajišťují spolehlivý chod aplikace. Nefunkční požadavky aplikace byly rozděleny do jednotlivých sekcí:

Výkon

- REQ-NF-1: Aplikace by měla reagovat na uživatelské akce do 2 sekund při běžném zatížení

Bezpečnost

- REQ-NF-2: Šifrování dat chránící uživatelská data před neoprávněným vstupem

Uživatelská přívětivost

- REQ-NF-3: Aplikace by měla být uživatelsky přívětivá a přehledná pro uživatele

Dostupnost

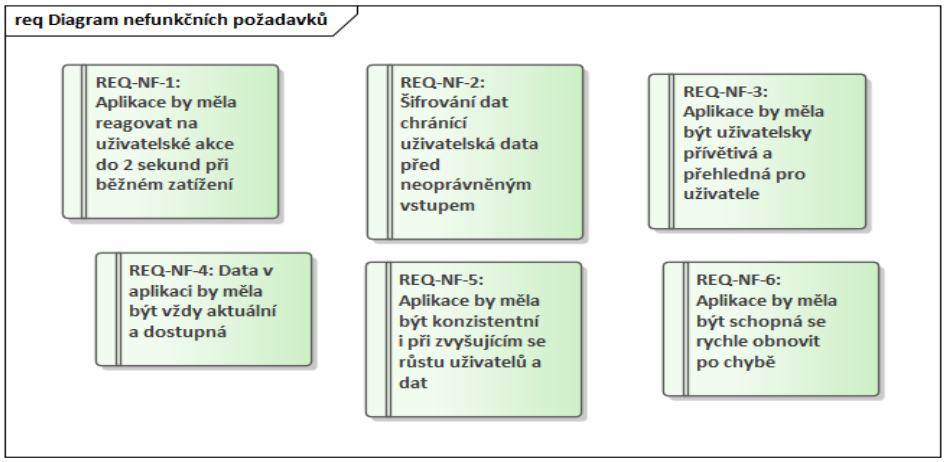
- REQ-NF-4: Data v aplikaci by měla být vždy aktuální a dostupná

Škálovatelnost

- REQ-NF-5: Aplikace by měla být konzistentní i při zvyšujícím se růstu uživatelů a dat

Spolehlivost

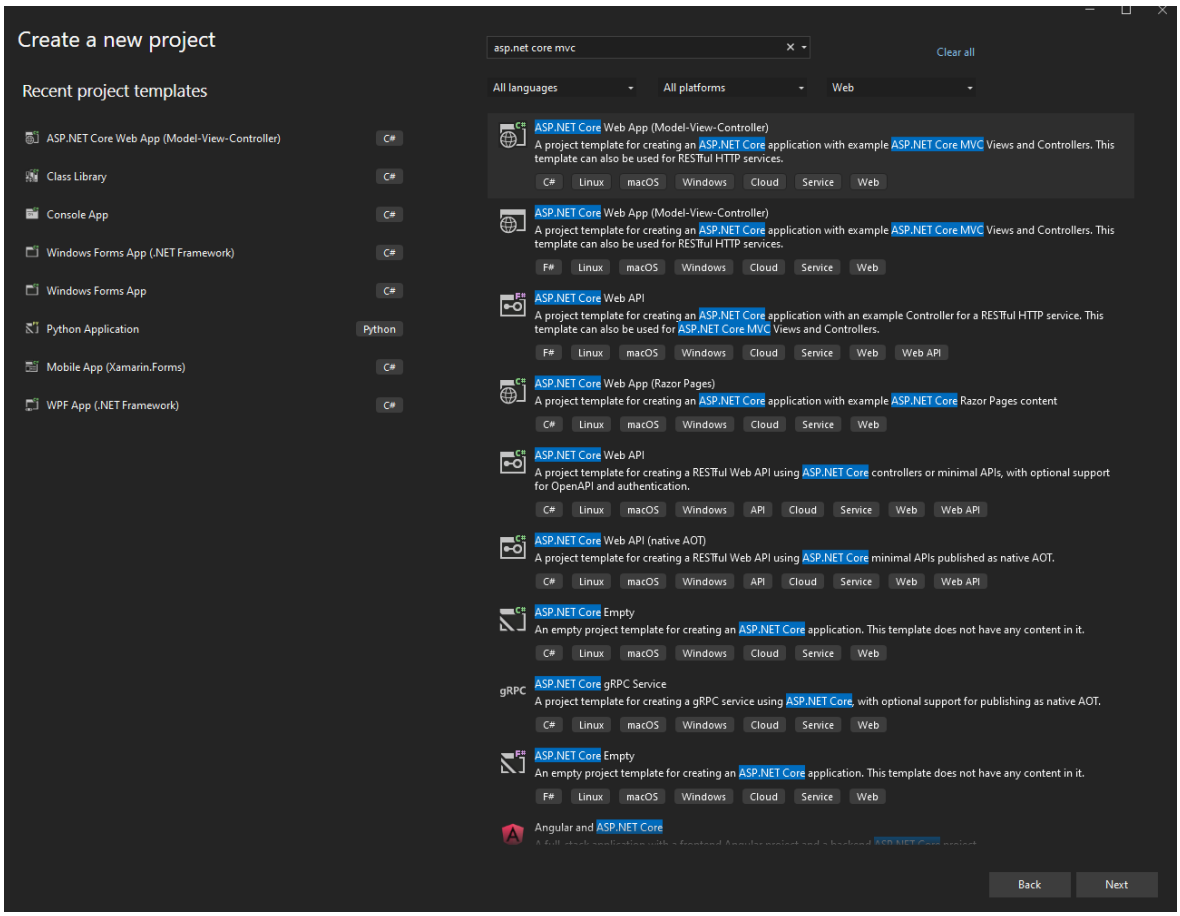
- REQ-NF-6: Aplikace by měla být schopná se rychle obnovit po chybě



Obrázek 34 Nefunkční požadavky aplikace vytvořené v programu Enterprise Architect

4.2 VÝVOJ V IDE

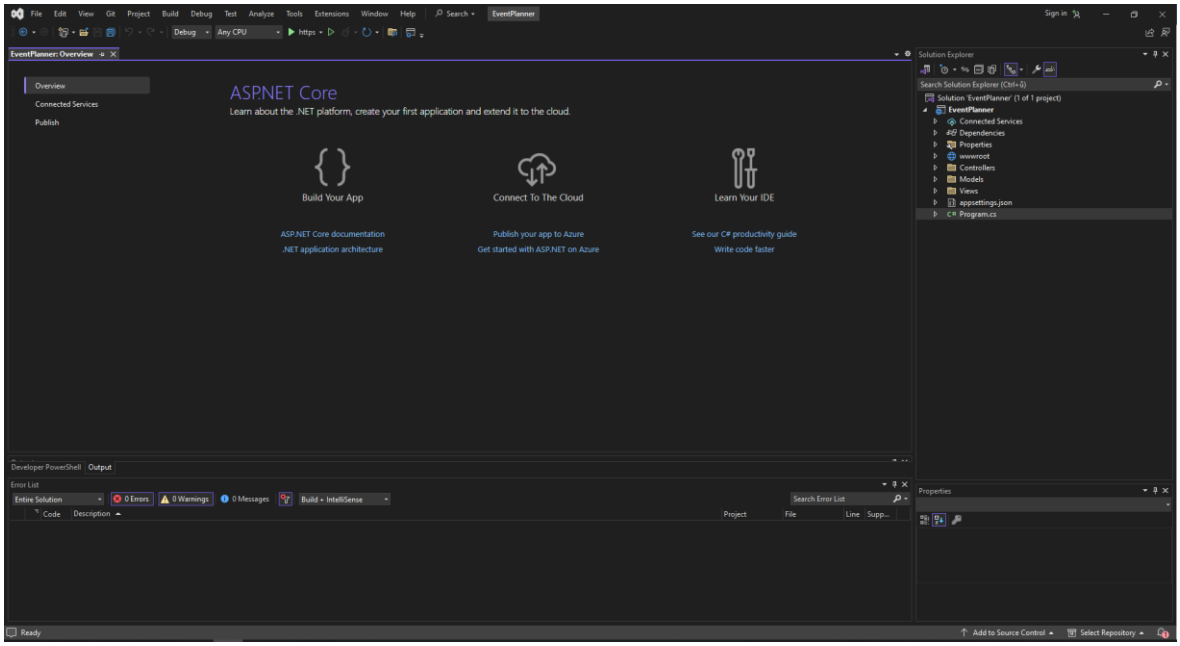
Pro vývoj aplikace bude použito Microsoft Visual Studio, které umožňuje použití technologie ASP.NET Core MVC. Prostředí je velmi moderní a obsahuje velikou řadu možností a funkcionalit potřebnou k vývoji.



Obrázek 35 Vytvoření nového projektu ASP.NET Core

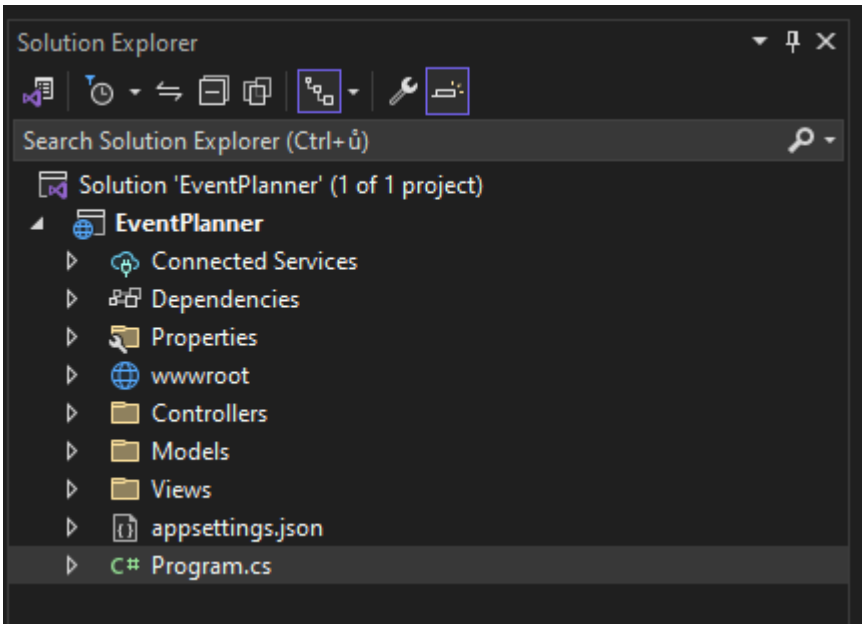
Jako název aplikace bude použit „EventPlanner“ charakterizující webovou aplikaci a .NET framework 8.0 – nejaktuálnější framework s dlouhodobou podporou.

Po vytvoření projektu se dostaneme do vývojového prostředí (Obrázek 36).



Obrázek 36 Vývojové prostředí Microsoft Visual Studio 2022

Důležitým oknem je pro nás Solution Explorer (Obrázek 37), v kterém vidíme naše složky a soubory související s vývojem webové aplikace.



Obrázek 37 Solution Explorer

V Solution Exploreru se nachází:

- Složka s Controllery: V této složce se nachází soubor HomeController.cs (Obrázek 38). Tento controller je hlavním domovským controllerem v naší aplikaci, jeho cílem je zpracovávat požadavky na domovskou stránku. Je odpovědný za správné zobrazení view ve složce Home, a to Index.cshtml a Privacy.cshtml.

```
 1. using EventPlanner.Models;
 2. using Microsoft.AspNetCore.Mvc;
 3. using System.Diagnostics;
 4. using System;
 5. using Microsoft.Extensions.Logging;
 6.
 7. namespace EventPlanner.Controllers
 8. {
 9.     public class HomeController : Controller
10.     {
11.         private readonly ILogger<HomeController> _logger;
12.
13.         public HomeController(ILogger<HomeController> logger)
14.         {
15.             _logger = logger;
16.         }
17.
18.         public IActionResult Index()
19.         {
20.             return View();
21.         }
22.
23.         public IActionResult Privacy()
24.         {
25.             return View();
26.         }
27.
28.         [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore =
true)]
29.         public IActionResult Error()
30.         {
31.             return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpCon-
text.TraceIdentifier });
32.         }
33.     }
34. }
```

Obrázek 38 HomeController

- Složka s Modely: Bude obsahovat modely použité v naší aplikaci.
- Složka s Views: Obsahuje složky Home a Shared. V Home se nachází view pro zobrazení domovské stránky a ve složce Shared

Jsou to složky s našimi controllery, modely a view, složka properties s nastavením spouštěcích profilů pro různých režimech. Kromě toho, zde taky nalezneme soubor Program.cs, který slouží pro nasazení aplikace a její spuštění.

4.3 DATABÁZOVÁ ARCHITEKTURA

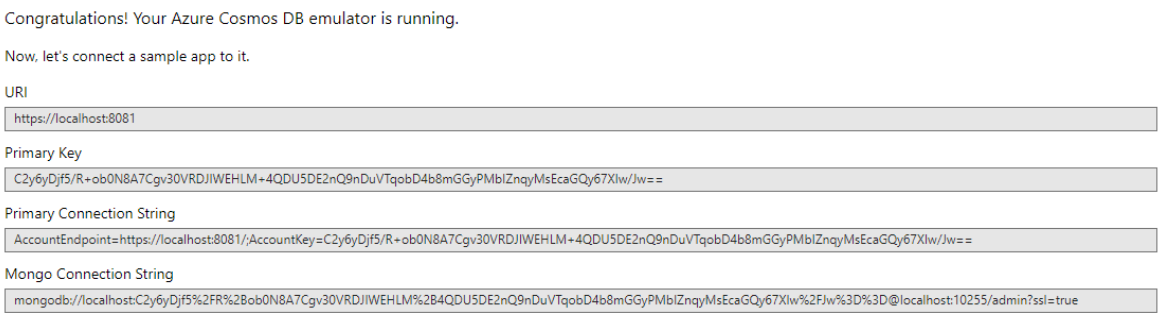
Tato část je věnována konkrétní implementaci databáze pro účely vyvíjené aplikace, která je zásadní pro funkčnost naší aplikace vytvořené v ASP.NET Core a používající Cosmos DB emulátor jako hlavní databázové řešení. Je zde představen proces propojení aplikace s Cosmos DB emulátorem a konfigurace potřebné pro správnou komunikaci mezi databází a aplikací. Je zde rozebrán postup vytvoření databáze a kontejnerů v Cosmos DB, v kterých budou organizována a ukládána naše data.

4.3.1 Propojení ASP.NET Core a Cosmos DB emulátoru

Propojení naší aplikace s Azure Cosmos DB Emulátorem vyžaduje několik kroků, přičemž základem je mít nainstalovaný Cosmos DB emulátor.

Emulátor je možné stáhnout zdarma z oficiálních stránek Microsoftu na adrese emulátoru. Po jeho nainstalování je nutné emulátor spustit s právy administrátora. Po spuštění emulátor automaticky otevře webovou stránku na adrese „https://localhost:8081/explorer/index.html“, která slouží jako uživatelské rozhraní pro správu databáze. První načtení stránky může trvat déle a pokud se zdá, že emulátor nefunguje, doporučuje se stránku po několika sekundách obnovit [34].

Po načtení stránky můžeme vidět rozhraní, které již bylo znázorněno v teoretické části v kapitole 1.3. Klíčovým prvkem pro úspěšné propojení naší ASP.NET Core aplikace s Cosmos DB emulátorem jsou připojovací řetězce (Connection Strings) (Obrázek 39). Tyto řetězce obsahují informace nezbytné ke komunikaci s databází jako jsou autentizační klíče a adresa endpointu.



Obrázek 39 Připojovací řetězce v Cosmos DB emulátoru

Pro propojení s naší aplikací využijeme „Primary Connection String“. Tento připojovací řetězec je univerzální a slouží k navázání spojení přes různé dostupné API, které Cosmos DB nabízí. V případě, že bychom chtěli využít MongoDB API, zvolili bychom místo toho „Mongo Connection String“.

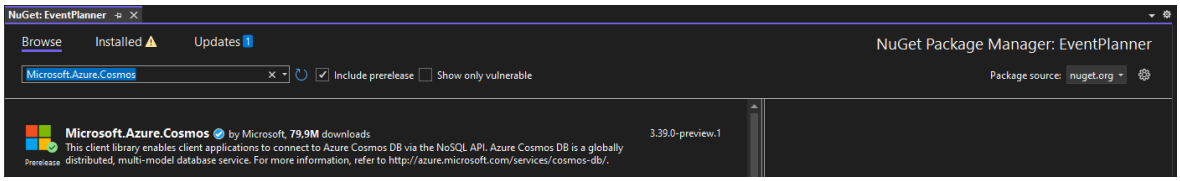
Dalším klíčovým krokem je správné nastavení konfiguračního souboru „appsettings.json“ (Obrázek 40) v kterém bude náš připojovací řetězec zaznamenán. Můžeme přidat taky další nastavení jako je například název naší databáze nebo pravdivostní hodnotu, která určuje, zda má databáze při spuštění vytvořit určité kontejnery.

```
1. {
2.   "SetupCosmosDb": "false",
3.   "CosmosIdentityDbName": "EventPlannerDatabase",
4.   "ConnectionStrings": {
5.     "CosmosDBConnectionString": "AccountEndpoint=https://localhost:8081/;Account-
Key=C2y6yDjf5..."
6.   }
7. }
```

Obrázek 40 Ústřižek kódu obsažený v souboru „apsettings.json“ obsahující nastavení databáze a připojovací řetězec

Toto nastavení využijeme později při vytváření identity pomocí ASP.NET Core identity. Samotný připojovací řetězec je uložený uvnitř „ConnectionStrings“ v „CosmosDBConnectionString“.

Následně nainstalujeme balíček „Microsoft.Azure.Cosmos“ vyhledáním v NuGet Package Manageru (Obrázek 41) nebo přes, Package Manager Console použitím příkazu „ Install-Package Microsoft.Azure.Cosmos“.



Obrázek 41 Balíček „Microsoft.Azure.Cosmos“ v NuGet Package Manageru

Tento balíček je nezbytný pro práci Cosmos DB a slouží k vytváření databáze a kontejnerů a manipulaci s nimi a jejich daty využitím CRUD (Create, Read, Update, Delete) operací.

4.3.1.1 Vytvoření databáze a kontejnerů

Nyní můžeme v „Program.cs“ vytvořit databázi a kontejnery s ní související. Nejdříve je potřeba vytvořit proměnou s připojovacím řetězcem a názvem databáze, kterou získáme z konfiguračního souboru aplikace. V dalších krocích je potřeba vytvořit novou instanci „CosmosClient“, které připojovací řetězec poskytneme.

```
1. var builder = WebApplication.CreateBuilder(args);
2.
3. string? connectionString = builder.Configuration.GetConnectionString("CosmosDBConnection-
String");
4.
5. CosmosClient client = new CosmosClient(connectionString);
```

Obrázek 42 Získání připojovacího řetězce a vytvoření instance CosmosClient

Následně můžeme na instanci „client“ zavolat metodu „CreateDatabaseIfNotExistsAsync“, která vytvoří databázi, pokud již taková databáze neexistuje. Id databáze si opět vytáhneme z konfiguračního souboru.

```
1. var cosmosIdentityDbName = builder.Configuration.GetValue<string>("CosmosIdentityDbName");
2.
3. await client.CreateDatabaseIfNotExistsAsync(cosmosIdentityDbName);
4.
5. Database database = client.GetDatabase(cosmosIdentityDbName);
```

Obrázek 43 Získání id databáze a vytvoření nové databáze

Také teď můžeme vytvořit instanci třídy „Database“, která umožnuje vytvořit jednotlivé kontejnery.

```
 1. Database database = client.GetDatabase(cosmosIdentityDbName);
 2.
 3. Container events = await database.CreateContainerIfNotExistsAsync(
 4.     id: "Events",
 5.     partitionKeyPath: "/DateCreated",
 6.     throughput: 400
 7. );
 8.
 9. Container tasks = await database.CreateContainerIfNotExistsAsync(
10.     id: "Tasks",
11.     partitionKeyPath: "/DateCreated",
12.     throughput: 400
13. );
14.
15. Container friendRequests = await database.CreateContainerIfNotExistsAsync(
16.     id: "FriendRequests",
17.     partitionKeyPath: "/CompositePartitionKey",
18.     throughput: 400
19. );
20.
21. Container userStatistics = await database.CreateContainerIfNotExistsAsync(
22.     id: "UserStatistics",
23.     partitionKeyPath: "/UserName",
24.     throughput: 400
25. );
```

Obrázek 44 Vytvoření jednotlivých kontejnerů

Při vytváření kontejnerů musíme specifikovat identifikátor kontejneru, partition klíč kontejneru, podle kterého jsou formovány jednotlivé logické oddíly a propustnost, která charakterizuje míru požadavků za sekundu.

4.3.1.2 CRUD Operace v databázi

Pro přidávání nových záznamů do Azure Cosmos DB kontejneru využíváme metodu „CreateItemAsync“, kterou voláme na existující instanci kontejneru (Obrázek 45). Tato asynchronní metoda efektivně přidává instance objektů, jako jsou položky úkolů, do specifikovaného kontejneru bez blokování hlavního vykonávacího vlákna aplikace.

```
1. public async Task CreateTaskAsync(ToDoItem toDoItem)
2. {
3.     ItemResponse<ToDoItem> response = await _container.CreateItemAsync(toDoItem);
4. }
```

Obrázek 45 Příklad vytvoření nového záznamu v kontejneru úkolů

Příklad demonstruje způsob přidání nového záznamu v kontejneru úkolů, kde „toDoItem“ je objekt obsahující data úkolu, který ukládáme. Jako výsledek se vrátí „response“, který je typu „ItemResponse<ToDoItem>“, obsahující důležité informace o operace, včetně identifikátoru, spotřebovaných jednotek RU a samotného objektu úkolu, což umožnuje další zpracování výsledku.

Pro načtení dat z kontejnerů můžeme využít metodu „ReadItemAsync“. Tato metoda je opět volána na kontejneru a vrací opět jako výsledek typ „ItemResponse“. Pro vyhledání správného objektu musíme charakterizovat typ, který bude vrácen, jednoznačný identifikátor dat a hodnotu „PartitionKey“ daného objektu, to umožňuje efektivní lokalizaci dat.

```
1. public async Task<ToDoItem> GetTaskAsync(ToDoItem toDoItem)
2. {
3.     ItemResponse<ToDoItem> response = await _container.ReadItemAsync<ToDoItem>(toDoItem.Id,
new PartitionKey(toDoItem.DateCreated));
4.     return response.Resource;
5. }
```

Obrázek 46 Získání objektu typu „ToDoItem“ z kontejneru

Na tomto příkladě již využijeme výstup z metody, kdy jako návratovou hodnot vrátíme „response.Resource“, která v sobě obsahuje hledaný objekt.

Pokud bychom chtěli načíst objektů více najednou, existuje několik možností, které se liší v závislosti na naších potřebách. Efektivním způsobem načtení dat s možností určité filtrace a bez potřeby předávání „PartitionKey“ je využití SQL dotazu.

```
 1. public async Task<List<ToDoItem>> GetAllPublicTasksAsync()
 2. {
 3.     var queryDefinition = new QueryDefinition("SELECT * FROM c WHERE (c.UserName = @Anony-
mous OR c.Accessibility = 'Public') AND NOT c.IsCompleted")
 4.         .WithParameter("@Anonymous", "Anonymous");
 5.
 6.     List<ToDoItem> toDoItems = new List<ToDoItem>();
 7.
 8.     using (FeedIterator<ToDoItem> iterator = _container.GetItemQueryIterator<ToDoI-
tem>(queryDefinition))
 9.     {
10.         while (iterator.HasMoreResults)
11.         {
12.             FeedResponse<ToDoItem> result = await iterator.ReadNextAsync();
13.             foreach (ToDoItem item in result)
14.             {
15.                 toDoItems.Add(item);
16.             }
17.         }
18.     }
19.
20.     return toDoItems;
21. }
```

Obrázek 47 Vrácení všech úkolů z kontejneru využitím SQL dotazu

Využitím třídy „QueryDefinition“ můžeme specifikovat nový SQL dotaz, který v tomto případě vybere všechny záznamy, které splňují podmínku, že pokud je daný úkol vytvořen anonymním uživatelem anebo je veřejný, ale zároveň nesmí být dokončený. Tento dotaz potom dáme jako parametr metodě „GetItemQueryIterator“ společně s typem objektu, kterého data budou. Pokud budou existovat záznamy přidáme je do naší kolekce úkolů.

Pro aktualizaci a mazání dat v Azure Cosmos DB kontejneru můžeme použít metody „ReplaceItemAsync“ pro aktualizaci (Obrázek 48) a „DeleteItemAsync“ pro mazání dat (Obrázek 49). Tyto operace vyžadují specifikaci „PartitionKey“.

```
 1. public async Task UpdateTaskAsync(ToDoItem toDoItem)
 2. {
 3.     ItemResponse<ToDoItem> response = await _container.ReplaceItemAsync(toDoItem, toDoI-
tem.Id, new PartitionKey(toDoItem.DateCreated));
 4. }
```

Obrázek 48 Aktualizace dat v kontejneru

Při aktualizaci je nutné předat celý objekt „toDoItem“, který v databázi nahradí existující objekt s odpovídající hodnotou identifikátoru. Důležité je, aby tento objekt obsahoval všechny požadované změny, protože „ReplaceItemAsync“ nahradí celý objekt v databázi nově předaným objektem.

```
1.  public async Task DeleteTaskAsync(ToDoItem toDoItem)
2.  {
3.      ItemResponse<ToDoItem> response = await _container.DeleteItemAsync<ToDoItem>(toDoI-
tem.Id, new PartitionKey(toDoItem.DateCreated));
4.  }
```

Obrázek 49 Mazání dat v kontejneru

A při mazání metoda „DeleteItemAsync“ vyžaduje pouze identifikátor objektu a „Partition-Key“ pro jeho identifikaci a odstranění z kontejneru. Důležité je taky správné ošetření potencionální výjimky, například pokud objekt k odstranění neexistuje.

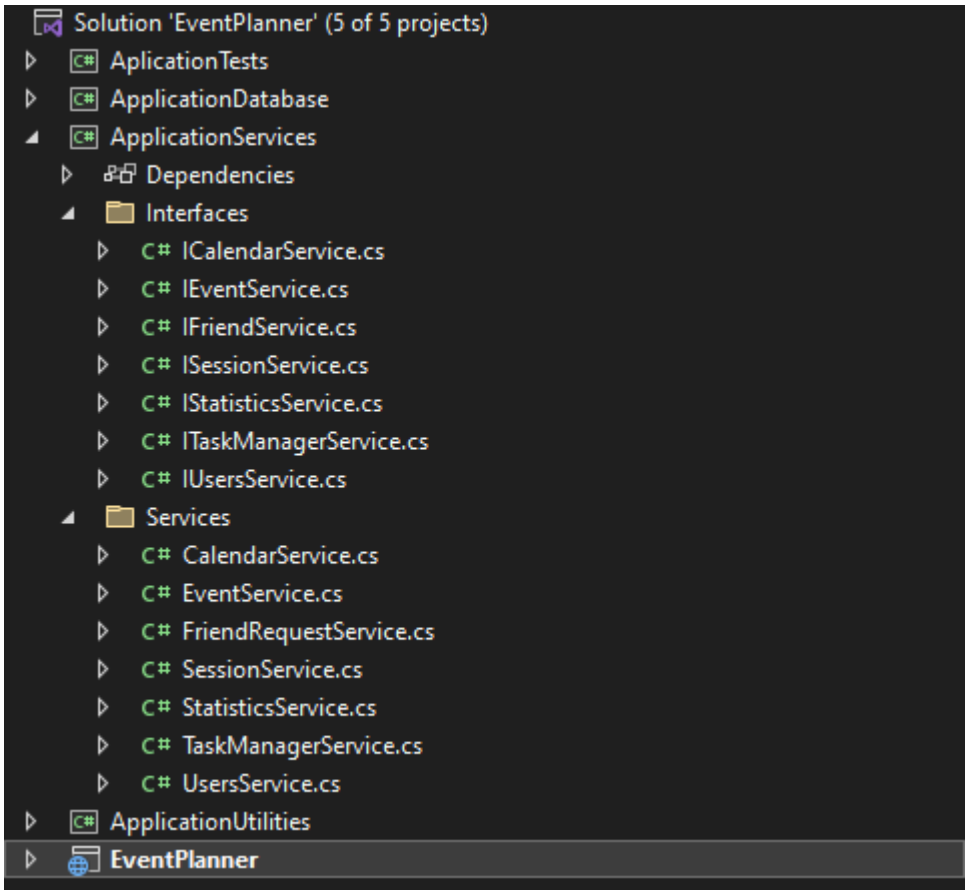
4.4 STRUKTURA APLIKACE

Jednotlivé komponenty v aplikaci jsou rozvržené do více projektů knihovny tříd, ve kterých se nacházejí v jednotlivých oddělených složkách potřebné funkcionality pro práci v aplikaci.

Uvnitř aplikace byly vytvořeny čtyři dodatečné projekty, které by měly splňovat následující funkcionality:

4.4.1 ApplicationServices

Zde jsou definované jednotlivé služby reprezentovány třídami a názvem, který jednoznačně určuje jejich funkcionalitu. Dále jsou zde jejich rozhraní, které předepisují, které metody budou v jednotlivých službách použity – jaké budou mít parametry, název a návratový typ. Jednotlivé služby budou dále využívány v controllerech, do kterých jsou injectovány pomocí návrhového vzoru Depedency injection. Služby jsou registrovány ve třídě „Program.cs“ (Obrázek 51), kde se při spuštění aplikace vytváří a přidávají do kontejneru společně s jejich závislostmi a životním cyklem. To zajišťuje správnou správu instancí služeb podle potřeby aplikace.



Obrázek 50 Projekt „ApplicationServices“ a jeho služby s příslušnými rozhraními

```
builder.Services.AddScoped<TaskManagerService>(provider => {
    var cosmosClient = provider.GetRequiredService<CosmosClient>();
    var databaseId = "EventPlannerDatabase";
    var containerId = "Tasks";
    var sessionService = provider.GetRequiredService<ISessionService>();
    return new TaskManagerService(cosmosClient, databaseId, containerId, sessionService);
});

builder.Services.AddScoped<UsersService>(provider => {
    var cosmosClient = provider.GetRequiredService<CosmosClient>();
    var databaseId = "EventPlannerDatabase";
    var containerId = "Users";
    var userManager = provider.GetRequiredService<UserManager<ApplicationUser>>();
    var friendService = provider.GetRequiredService<IFriendService>();
    return new UsersService(cosmosClient, databaseId, containerId, userManager, friendSer-
vice);
});

builder.Services.AddScoped<FriendService>(provider => {
    var cosmosClient = provider.GetRequiredService<CosmosClient>();
    var databaseId = "EventPlannerDatabase";
    var containerId = "FriendRequests";
    var userManager = provider.GetRequiredService<UserManager<ApplicationUser>>();
    return new FriendRequestService(cosmosClient, databaseId, containerId, userManager);
});

builder.Services.AddScoped<StatisticsService>(provider => {
    var cosmosClient = provider.GetRequiredService<CosmosClient>();
    var databaseId = "EventPlannerDatabase";
    var containerId = "UserStatistics";
    return new StatisticsService(cosmosClient, databaseId, containerId);
});

```

Obrázek 51 Registrace jednotlivých služeb v „program.cs“

Zde jsou registrovány jednotlivé služby s životním cyklem „Scoped“ při kterém bude vytvářena nová instance služby při každém novém http požadavku. Jednotlivé služby také vyžadují i název databáze a kontejneru, s kterým pracují v Cosmos DB databázi. Aby takový přístup byl realizovatelný, je nutné také přidat instanci CosmosClienta. Tyto konkrétní služby jsou potom určeny pro správu a manipulaci s daty uvnitř těchto kontejnerů. Umožnují tak například přidání nové statistiky nebo žádosti o přátelství (Obrázek 52).

```
public async Task CreateStatisticForUserAsync(UserStatistic statistic)
{
    if (statistic != null)
    {
        await _container.CreateItemAsync(statistic);
    }
}

public async Task UpdateStatisticForUserAsync(UserStatistic statistic)
{
    if (statistic != null)
    {
        await _container.ReplaceItemAsync<UserStatistic>(statistic, statistic.Id, new Par-
titionKey(statistic.UserName));
    }
}
```

Obrázek 52 Metody pro přidání a aktualizace statistiky ve službě „StatisticsService“

Tyto dvě konkrétní metody služby „StatisticsService" umožnují vytvoření a aktualizaci statistiky uživatele podle modelu „UserStatistic". Při volání je potřebná instance kontejneru vytvářená na základě identifikátoru kontejneru, který je charakterizován při registraci služby.

4.4.2 ApplicationDatabase

V tomto projektu jsou definované základní prvky poskytující rozhraní pro interakci s databází. Jsou zde složky, které představují objekty, s kterými se pracuje v rámci naší aplikace.

DbContext

- Složka obsahuje třídu „EventPlannerDbContext" (Obrázek 57), která byla vytvořena pro naše specifické účely propojení ASP.NET Core Identity s Cosmos DB. Tato třída poskytuje základ pro autentizaci a správu uživatelských účtů a rolí přímo Cosmos DB.

Models

- Zde jsou obsaženy třídy, které charakterizují unikátní entity v této aplikaci. Na základě těchto modelů jsou následným způsobem uložena data v databázi podle jejich charakteristických vlastností.

```
1. public class UserStatistic
2. {
3.     [JsonProperty("id")]
4.     public required string Id { get; set; }
5.
6.     public required string UserName { get; set; }
7.
8.     public int FinishedTasks { get; set; }
9.
10.    public int CurrentTasks { get; set; }
11.
12.    public string? MostFinishedByPriority { get; set; }
13. }
14.
```

Obrázek 53 Model „UserStatistic" uložený ve složce „Models"

Na tomto obrázku je jeden z modelů, které jsou potom dále využívány napříč aplikací a pomocí služby „StatisticsService" jsou na jeho základě například vytvářeny nebo aktualizovány nové statistiky v databázi.

ViewModels

- Ve složce ViewModels se nacházejí specifické třídy vytvořené speciálně pro účely uživatelského zobrazení. Kombinují data z více modelů pro optimalizované zobrazení uživateli.

```
1. public class FriendsViewModel
2. {
3.     public List<ApplicationUser>? Users { get; set; }
4.     public List<ApplicationUser>? Friends { get; set; }
5.     public List<FriendRequestViewModel>? Requests { get; set; }
6. }
7.
```

Obrázek 54 ViewModel „FriendsViewModel“

Zde je příklad ViewModelu „FriendsViewModel“, který slouží ke sdružení dat dvou modelů a jejich jednotlivých použití. Při zobrazování stránky se všemi uživateli je potřeba odlišit všechny uživatele od přátel, návrhů na přátelství anebo žádostí o přátelství. Je zde proto přidáno více objektů, které jednoznačně charakterizují tuto problematiku. Tyto objekty jsou naplněny daty pomocí jednotlivé služby a dále předány do související stránky, kde jsou pomocí Razor syntaxe správně zobrazeny.

4.4.3 ApplicationUtilities

Tento projekt má účel přidání pomocných, rozšiřujících prvků v aplikaci. V kontextu naší aplikace je zde pouze složka „Extensions“ ve které je obsažena třída „SessionExtensions“ pomocí které je možné ukládat nebo aktualizovat jakákoliv data různého typu do session.

Zde je příklad generické metody „GetObject“ a metody „SetObject“ (Obrázek 55), které slouží pro ukládání a čtení různých typů objektů, jako jsou uživatelé, úkoly nebo události do session pro použití bez neustálého přístupu k databázi.

```
1. public static T? GetObject<T>(this ISession session, string key)
2. {
3.     var data = session.GetString(key);
4.     if (data == null)
5.     {
6.         return default;
7.     }
8.     return JsonSerializer.Deserialize<T>(data);
9. }
10.
11. public static void SetObject(this ISession session, string key, object value)
12. {
13.     session.SetString(key, JsonSerializer.Serialize(value));
14. }
```

Obrázek 55 Metody třídy „SessionExtensions“ pro práci s objekty v session

Metoda „GetObject“ vyhledá a vrátí objekt specifikovaného typu na základě klíče, který je poskytnut. Pokud jsou data nalezena jako „null“, vrátí metoda výchozí hodnotu daného typu, jinak jsou data deserializována a vrácena jako instance objektu.

Na druhé straně „SetObject“ bere objekt jakéhokoliv typu, serializuje ho do formátu JSON a ukládá do session pod určitým klíčem. Tento proces přeměny objektu na řetězec umožňuje efektivní uložení dat do session a jejich následné obnovení v nezměněné formě.

4.4.4 ApplicationTests

V tomto projektu se nachází testovací metody, které mají za účel ověřit funkcionalitu jednotlivých servisních tříd a jejich metod. Testovací metody tak například ověřují, zda se správně vyhodí výjimka v metodě pro vytvoření úkolu nebo jestli jiná metoda správně načítá úkol (Obrázek 56).

Pro realizaci testů byla aplikována kombinace nástrojů Xunit a Moq. Xunit poskytuje testovací framework, zatímco Moq umožňuje simulaci objektů a jejich instancí, což je klíčové pro izolované jednotkové testy bez závislosti na externích zdrojích nebo stavu aplikace.

```
 1. [Fact]
 2. public async Task GetTaskAsync_ReturnsTask()
 3. {
 4.     //Arrange
 5.     var mockCosmosClient = new Mock<CosmosClient>();
 6.     var mockLogger = new Mock<ILogger<TaskManagerService>>();
 7.     var mockTaskContainer = new Mock<Container>();
 8.     var mockItemResponse = new Mock<ItemResponse<ToDoItem>>();
 9.     var mockSessionService = new Mock<ISessionService>();
10.     var toDoItem = new ToDoItem { Id = "123", Name = "Test" };
11.
12.     mockItemResponse.Setup(x => x.Resource).Returns(toDoItem);
13.     mockTaskContainer.Setup(x =>
14.         x.ReadItemAsync<ToDoItem>("123", It.IsAny<PartitionKey>(), null, default))
15.         .ReturnsAsync(mockItemResponse.Object);
16.     mockCosmosClient.Setup(x => x.GetContainer("EventPlannerDatabase", "Tasks"))
17.         .Returns(mockTaskContainer.Object);
18.
19.     var taskManagerService = new TaskManagerService(
20.         mockCosmosClient.Object, "EventPlannerDatabase", "Tasks", mockSessionService.Ob-
ject, mockLogger.Object);
21.
22.     //Act
23.     var result = await taskManagerService.GetTaskAsync(toDoItem);
24.
25.     //Assert
26.     Assert.NotNull(result);
27.     Assert.Equal("Test", result.Name);
28.     Assert.Equal("123", result.Id);
29. }
```

Obrázek 56 Ukázka testovací metody

Tento obrázek prezentuje ukázkovou testovací metodu. Všechny nezbytné závislosti, které jsou nutné pro testování služby „TaskManagerService“ byly nasimulovány pomocí Moq. Testovací metoda „GetTaskAsync“ je ověřována z hlediska správnosti vrácených výsledků. Ukázka ilustruje schopnost systému vrátit konkrétní úkol založený na unikátním identifikátoru.

4.4.5 Správa účtů a zabezpečení

Vzhledem k tomu, že knihovna pro správu uživatelů - „ASP.NET Core Identity“ je primárně designovaná pro relační databáze s použitím Entity Framework Core, bylo zapotřebí najít alternativní řešení umožňující efektivní integraci s NoSQL databází Azure Cosmos DB. Přidáním externího balíčku „AspNetCore.Identity.CosmosDb“ lze poskytnout adaptér pro tuto komunikaci. Díky tomu je možné využít některých tříd, které poskytují bezpečnou a efektivní správu uživatelských účtů [35].

Pro adaptaci byla vytvořena třída „EventPlannerDbContext“, která dědí od „CosmosIdentityDbContext“ (Obrázek 57). Tato třída tak slouží jako základ pro operace spojené se správou účtů.

```
1. public class EventPlannerDbContext : CosmosIdentityDbContext<ApplicationUser, IdentityRole,
string>
2. {
3.     public EventPlannerDbContext(DbContextOptions options) : base(options)
4.     {
5.     }
6. }
```

Obrázek 57 Nastavení třídy „EventPlannerDbContext“

Poté je možné vytvořit kontejnery spojené s uživateli a jejich správou.

```
 1. if (bool.TryParse(setupCosmosDb, out var setup) && setup && connectionString != null &&
cosmosIdentityDbName != null)
 2. {
 3.     var builder1 = new DbContextOptionsBuilder<EventPlannerDbContext>();
 4.     builder1.UseCosmos(connectionString, cosmosIdentityDbName);
 5.
 6.     using (var dbContext = new EventPlannerDbContext(builder1.Options))
 7.     {
 8.         dbContext.Database.EnsureCreated();
 9.     }
10. }
```

Obrázek 58 Vytvoření kontejnerů spojených s ASP.NET Core Identity

Tento postup je součástí inicializačního procesu aplikace, který ověřuje existenci databáze a příslušných kontejnerů. Pokud databáze nebo potřebné kontejnery neexistují, jsou na základě připojovacího řetězce a názvu databáze poskytnutých v konfiguračním souboru vytvořeny. V rámci tohoto procesu se vytvoří sedm kontejnerů, které jsou klíčové pro správu uživatelských účtů a jejich rolí pomocí ASP.NET Core Identity.

Následně je potřeba registrovat službu DbContextu v dependency kontejneru aplikace se správným připojením k naší databázi (Obrázek 59).

```
1. if (connectionString != null && cosmosIdentityDbName != null)
2. {
3.     builder.Services.AddDbContext<EventPlannerDbContext>(options =>
4.         options.UseCosmos(connectionString, cosmosIdentityDbName));
5. }
```

Obrázek 59 Registrace „EventPlannerDbContext“ s vhodným připojením

Dalším krokem je integrace ASP.NET Core identity s naším „EventPLannerDbContext“ (Obrázek 60). Díky tomu budeme mít přístup k využívání bohatých možností pro správu uživatelů a jejich rolí.

```
1. builder.Services.AddCosmosIdentity<EventPlannerDbContext, ApplicationUser, IdentityRole,
string>(options =>
2.     options.SignIn.RequireConfirmedAccount = false);
```

Obrázek 60 Přidání Cosmos identity

Toto nastavení zahrnuje povolení registrace bez nutnosti ověřování e-mailové adresy, což by bylo vhodné v budoucnu změnit pro bezpečnější ověřování uživatelů.

Pro lepší zabezpečení uživatelských účtů je poté potřeba definovat chování při registraci a přihlašování, toho můžeme docílit nastavení možností identity následujícím způsobem (Obrázek 61).

```
 1. builder.Services.Configure<IdentityOptions>(options =>
 2. {
 3.     // Nastavení silnějšího hesla
 4.     options.Password.RequireDigit = true;
 5.     options.Password.RequiredLength = 8;
 6.     options.Password.RequireNonAlphanumeric = true;
 7.     options.Password.RequireUppercase = true;
 8.     options.Password.RequireLowercase = true;
 9.     options.Password.RequiredUniqueChars = 4;
10.
11.     // Zablokování
12.     options.Lockout.AllowedForNewUsers = true;
13.     options.Lockout.MaxFailedAccessAttempts = 5;
14.     options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(5);
15.
16.     // Unikátní email
17.     options.User.RequireUniqueEmail = true;
18. });
```

Obrázek 61 Nastavení zadávání hesel, zablokování a unikátního emailu

V tomto scénáři bude po uživateli při registraci vyžadováno zadání alespoň jednoho čísla, speciálních znaků, malého a velkého písmena. Čtyři znaky také musí být úplně unikátní a velikost hesla musí být alespoň osm znaků. Nastavení politiky zablokování účtu je v aplikaci konfigurováno s cílem poskytnout nově vytvořeným účtům ochranu před potencionálními útoky hrubou silou. Jestliže dojde při pokusu o přihlášení k pěti po sobě jdoucím chybným

zadáním hesla, tak bude daný účet zablokován na dobu pěti minut. Toto opatření tak zvyšuje bezpečnost uživatelských účtů. Poslední nastavení určuje vyžadování unikátnosti emailové adresy. Při registraci nového uživatele tak nebude možné použít adresu, kterou již má existující uživatel.

Pro správnou autentizaci uživatelů je taky nutné do aplikace integrovat „app.UseAuthentification()“. Tento midleware zodpovídá za zpracování přihlašovacích údajů (jména a hesla) a ověření jejich platnosti. Uživatel je pak považován za autentizovaného a systém mu vytvoří a přiřadí uživatelskou identitu.

Následně „app.UseAuthorization“ pro správnou autorizaci, která následuje po autentizaci. Autorizovaný uživatel má přístup k určitým zdrojům nebo vykonávání určitých akcí (Obrázek 62). Toto je založeno na uživatelských rolích, které je možné libovolně definovat. Pokud uživatel nesplňuje určitá kritéria, bude mu přístup odepřen.

```
1. [Area("User")]
2. [Authorize(Roles = "User,Admin")]
3. public class FriendsController : BaseController
4. {
5. //Vnitřní logika controlleru
6. }
```

Obrázek 62 Nastavení přístupu pro controller

Tento příklad demonstruje nastavení přístupu pro celý controller. „FriendsController“.

Metody, které se uvnitř vykonávají budou přístupné jen pro uživatele v rolích „User“ a „Admin“. Tímto způsobem se zabrání nepřihlášeným nebo uživatelům, kteří nemají tuto roli v přístupu k jednotlivým stránkám aplikace, které jsou součástí této sekce.

4.5 HTML FRAMEWORK POUŽITÝ V APLIKACI

Pro vytvářenou aplikaci byl použit framework Boostrap. Framework byl vybrán na základě jeho vhodných vlastností a komponent vhodných při vytváření uživatelských rozhraní pro plánování úkolů a událostí.

Využity zde byly komponenty jako například karty, tlačítka, zakulacení rohů anebo vytvoření uživatelsky přívětivého rozložení jednotlivých částí stránky do mřížkového systému.

To taky umožnilo a dalo základ vzniku kalendáře. Stránka kalendáře byla rozložena na tři části tak, aby uprostřed se nacházel hlavní obsah s jednotlivými kalendářními dny. Uvnitř vytvořena tabulka dní s požadujícím obsahem.

Dále byla použita komponenta zobrazení obsahu do oddělených části s horizontálním nebo vertikálním scrollováním, kde se nacházejí například dokončené úkoly jednotlivých uživatelů (Obrázek 63). V této komponentě pak byla vytvořena karty s jednotlivými úkoly. Toho bylo možné docílit díky vlastnosti „overflow-auto“, kterou Bootstrap nabízí. Způsob provedení je poté zobrazen na následujícím obrázku:

```
<h3><strong>FINISHED TASKS</strong></h3>
<div class="overflow-auto shadow rounded border-5" style="height: 350px;">
    @foreach (var toDoItem in Model.FinishedToDoItemsOfUser)
    {
        <div class="card">
            <div class="card-body">
                @* Zde se dále nachází obsah dokončených úkolů uživatele *@
            </div>
        </div>
    }
</div>
```

Obrázek 63 Příklad využití Bootstrapu pro zobrazení dokončených úkolů

Obrázek 63 ilustruje taky použití Bootstrap tříd „rounded“ pro zakulacení rohů, „border“ pro výrazné ohraničení a „shadow“ pro přidání stínu. To vytváří vizuálně atraktivní karty, které zobrazují dokončené úkoly.

Díky responsivním vlastnostem frameworku je také možnost zobrazení stránek na menších zařízeních.

Framework tak nabídl uživatelsky přívětivější zobrazené stránky.

5 VYTVOŘENÍ PODPŮRNÝCH MATERIÁLŮ

Na základě vyvíjené aplikace byla vytvořena sada úkolů, v kterých mají studenti implementovat vlastní funkce a ověřit jejich funkčnost. Také byly vytvořeny prezentace v PowerPointu (Příloha P I), které slouží ke grafickému znázornění tématu, popsání jednotlivých částí a ponaučení a způsobu tvorby jednotlivých částí aplikace. Prezentace tak slouží jako podpůrný nástroj ke splnění úkolů. Byl také vytvořen Moodle kurz, ve kterém se nacházejí prezentace i s úkoly a testovací otázky, které mají za úkol ověření znalostí.

5.1 ÚKOL VYUŽITÍ CRUD OPERACÍ V COSMOS DB

Cíl: Poskytnutí studentům praktických příkladů a cvičení založených na CRUD operacích pro lepší pochopení manipulaci s daty v NoSQL databázích. Cílem je rovněž naučit studenty správně ošetřit vstupní a výstupní data a reagovat tak na různé výjimky.

Popis úlohy:

- Vytvoření záznamu: Studenti dostanou zadání vytvořit funkci ve své aplikaci. Funkce bude schopná přidávat nové záznamy do Cosmos DB (Obrázek 64). Záznam je realizován pomocí konkrétního objektu.

```
1. public async Task CreateTaskAsync(ToDoItem toDoItem)
2. {
3.     ItemResponse<ToDoItem> response = await _container.CreateItemAsync(toDoItem);
4. }
```

Obrázek 64 Příklad způsobu přidání nového záznamu

- Čtení záznamu: Studenti vytvoří funkci, která bude schopna načíst specifické položky z databáze na základě zadaného klíče.
- Aktualizace záznamu: Implementace funkce pro nahrazení stávajícího záznamu nově aktualizovaným záznamem.
- Mazání záznamu: Vytvoření metody pro odstranění záznamu.
- Ošetření záznamů: V každém úkolu by studenti měli správně ošetřit vstupní data a vypisovat vhodným způsobem úspěch výstupních dat.

Otázky k diskuzi:

- Jaký význam má ošetření výjimek a chyb v kontextu CRUD operací?
- Jak by se změnil přístup datům, pokud byste přešli na tradiční relační databázi místo NoSQL?

5.2 ÚKOL VYUŽITÍ AUTENTIZAČNÍCH OPERACÍ V ASP.NET CORE IDENTITY

Cíl: Poskytnutí praktických příkladů a cvičení založených na autentizačních operací pro lepší pochopení implementace bezpečnosti v aplikaci. Cílem je také ukázat správné využití ASP.NET Core identity, která slouží pro správu uživatelských účtů.

Popis úlohy:

- Registrace nového uživatele: Studenti budou muset vytvořit funkci, která registruje nové uživatele. Následující příklad kódu je poskytnut jako vzor:

```
1. public async Task<IActionResult> Register(RegistrationViewModel model)
2. {
3.     var user = new ApplicationUser
4.     {
5.         UserName = model.Username,
6.         Email = model.Email,
7.         FirstName = model.FirstName,
8.         LastName = model.LastName,
9.         Role = ApplicationRoles.User
10.    };
11.
12.    var result = await _userManager.CreateAsync(user, model.Password);
13.
14.    if (result.Succeeded)
15.    {
16.        //Další logika v případě, že se podaří vytvořit uživatele
17.    }
18.
19.    return View(model);
20. }
```

Obrázek 65 Vzor metody pro registraci nového uživatele

- Přihlášení uživatele: Vytvoření funkce pro přihlášení.
- Odhlášení uživatele: Funkce, která odhlásí přihlášeného uživatele.
- Ošetření záznamů a uživatelských dat: Všechny funkce budou umožnovat správně ošetřit vstupní uživatelský model. V případě, že data budou neplatná nebo daný uživatel již existuje neměly by se dané operace vykonat.
- Správné nastavení: Nastavte správně způsob, kterým se přihlašuje. V hlavní třídě aplikace „Program.cs“, vhodně určete pravidla pro autentizaci.

Otázky k diskuzi:

- Proč je potřeba ověřovat stav modelu pro přihlašování a registraci?

5.3 ÚKOL VYUŽITÍ AUTORIZAČNÍCH OPERACÍ V ASP.NET CORE IDENTITY

Cíl: Cílem je naučit studenty správně nastavit oprávněné přístupy v aplikaci a zabezpečit aplikaci proti neoprávněným akcím. Studenti mají za úkol správně ošetřit dostupnost určitých funkcí a zabezpečit formuláře proti CSRF útokům.

Popis úlohy:

- Autorizace na základě uživatelských rolí: Vytvořte vhodně role pro uživatele a nastavte autorizační pravidla, která omezí přístup k určitým částem aplikace na základě těchto rolí.
- Přidání rolí uživatelům: Přidejte uživatelům správně role. Ujistěte se, že přidání rolí proběhlo správně.
- Ověření vstupu: Využijte atributy jako „[Authorize]" pro oprávněný přístup k metodám controllerů v rámci vaší aplikace.

```
1. [HttpGet]
2. [Authorize(Roles = "Admin,User")]
3. public async Task<IActionResult> FinishedTasks()
4. {
5.     //Implementace metody
6. }
```

Obrázek 66 Vzorový příklad autorizace rolí pro přístup ke konkrétní metodě

- Zabezpečení tokenů: Zajistěte správné využívání autentizačních tokenů pro ověřování uživatelských požadavků a jejich důvěryhodný zdroj. Ve všech metodách, které přijímají „POST" požadavek definujte validní anti-forgery token pomocí atributu „[ValidateAntiForgeryToken]". Tento token by měl být přístupný v rámci celé aplikace.

```
 1. [HttpPost]
 2. [ValidateAntiForgeryToken]
 3. public async Task<IActionResult> UpdateTask(TaskViewModel model)
 4. {
 5.     if (ModelState.IsValid)
 6.     {
 7.         //Logika pro aktualizaci úkolu
 8.     }
 9.     return RedirectToAction(nameof(HomeController.Index), nameof(HomeController));
10. }
```

Obrázek 67 Vzorový příklad využití atributu „[ValidateAntiForgeryToken]"

Otázky k diskuzi:

- Jaký je účel používání anti-forgery tokenů ve webových aplikacích?

- Co se asi stane, pokud uživatel nemá roli pro přístup k odpovídající stránce?

5.4 ÚKOL VYUŽITÍ SPRÁVY UŽIVATELSKÝCH INTERAKCÍ

Cíl: Vytvoření funkcí pro uživatelské interakce, jako jsou žádosti o přátelství, zobrazení přátel a návrhů na přátele. Implementujte službu pro práci s uživatelskými operacemi.

Popis úlohy:

- Přidání přítele: Vytvořte metodu, která umožňuje uživateli poslat žádost o přátelství. Vytvořte vlastní nebo umožněte funkčnost na základě tohoto vzoru:

```
 1. [HttpPost]
 2. [ValidateAntiForgeryToken]
 3. public async Task<IActionResult> AddFriend(string userName)
 4. {
 5.     ApplicationUser? requestTo = new ApplicationUser(); //Vhodným způsobem získat uživa-
tele -např. skrz userName
 6.     ApplicationUser? requestFrom = new ApplicationUser(); //Získat uživatele, který je
přihlášený a posílá žádost
 7.
 8.     await _friendService.SendFriendRequestAsync(requestTo, requestFrom); //Poslat žádost
uživateli
 9.
10.     return RedirectToAction(nameof(FriendsController.MyFriends), nameof(FriendsCon-
troller).Replace(nameof(Controller), ""));
11. }
```

Obrázek 68 Příklad části kódu pro poslání žádosti uživateli

- Přijmutí žádosti: Funkce, která příjme žádost od uživatele a ten se tak stane novým přítelem.
- Odmítnutí žádosti: Zamítnutí žádosti od uživatele.
- Odstranění přítele: Umožní odstranit přítele ze seznamu přátel.
- Správné ošetření: Vhodně ošetřete data uživatelů a operací –např. uživatel, který již přijmul žádost nebo v seznamu žádostí nebo návrhů na přátelství.
- Ukládání: Data vhodně ukládejte do kontejnerů, kde půjdou vidět vztahy mezi uživateli.

Otázky k diskuzi:

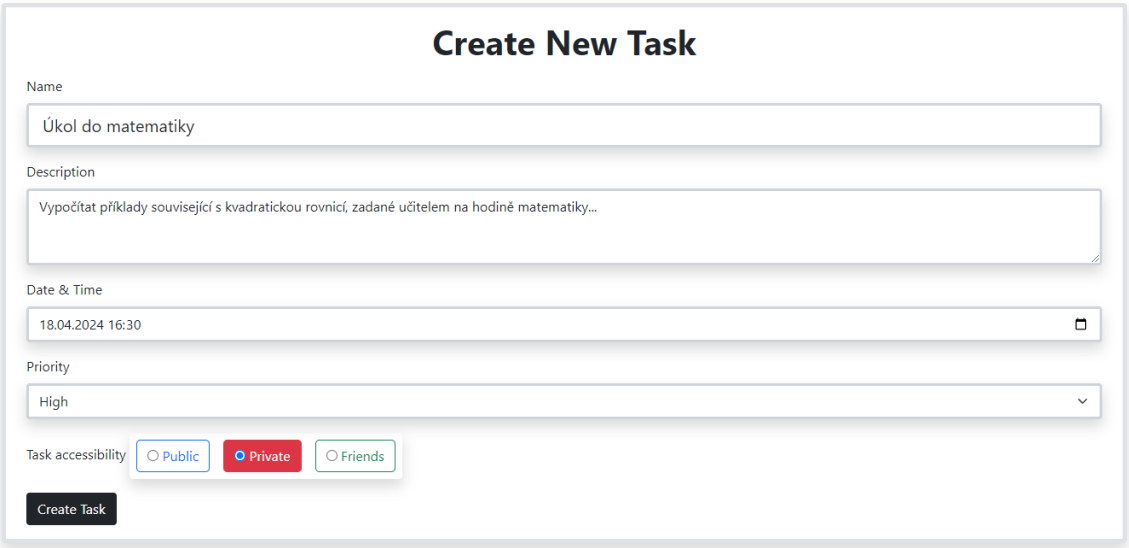
- Jak by se následující funkce daly implementovat bez nutnosti aktualizace webové stránky v prohlížeči?
- Je nutné data pokaždé ukládat do kontejneru nebo načítat z databáze?

5.5 ÚKOL VYTVOŘENÍ UŽIVATELSKÝCH ROZHRANÍ PRO SPRÁVU A ZOBRAZENÍ ÚKOLŮ

Cíl: Vytvoření stránek pro vytváření a zobrazování úkolů, kde bude možné procházet vytvořené úkoly, upravovat je a mazat. Studenti tak propojí funkcionalitu z úkolu 5.1 k zobrazení ve vizuálním provedení.

Popis úlohy:

- Vytvoření nového úkolu: S využitím Boostrapu vytvořte stránku, na které bude moct uživatel aplikace vytvořit nový úkol, který následně bude možné zobrazit v aplikaci. Zde je vzorový příklad stránky, která umožnuje přidat nový záznam na základě uživatelského vstupu a modelu:



Obrázek 69 Vzorový příklad stránky pro přidání úkolu v naší aplikaci

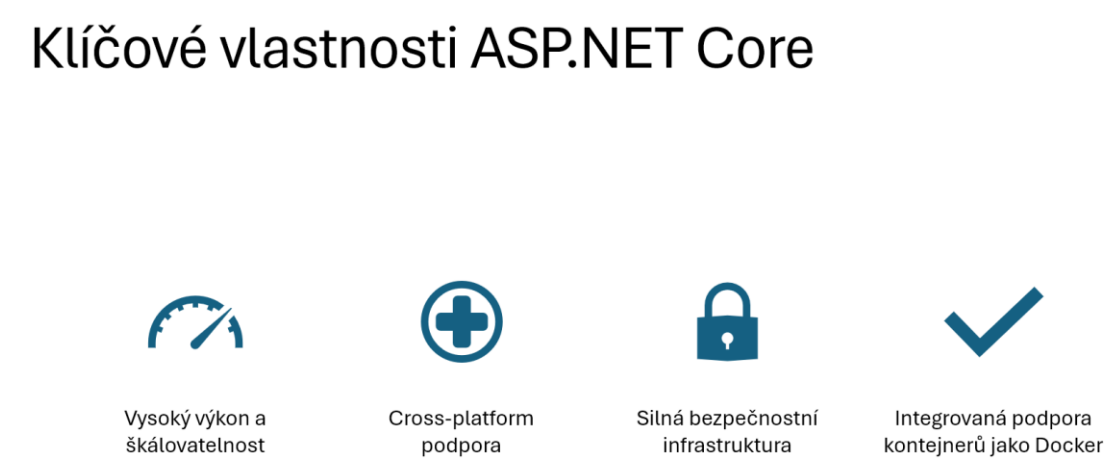
- Zobrazení úkolu: Vhodným způsobem na jiné stránce zobrazte všechny vytvořené úkoly s jejich vlastnostmi – jménem, popiskem a dalšími atributy.

Otázky k diskuzi:

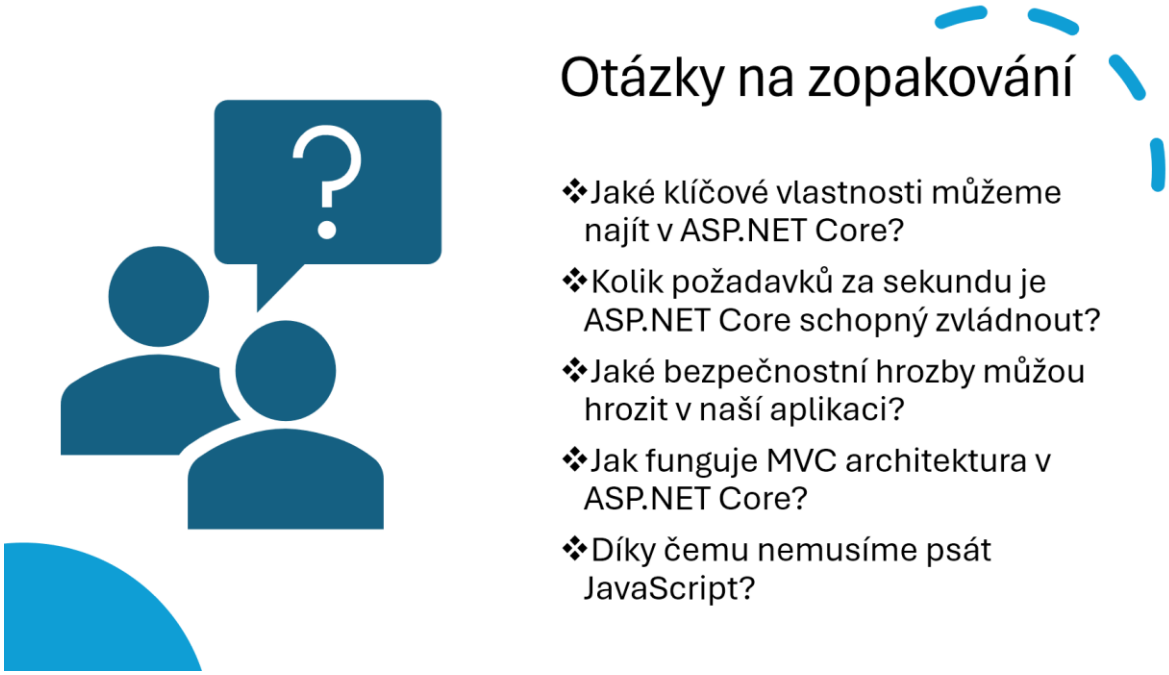
- Proč je vhodné využívat frameworku Bootstrap pro vytváření stránek?
- Co je potřeba ošetřit na stránce?

5.6 UKÁZKA PREZENTACÍ

Bylo celkem vytvořeno devět prezentací v PowerPointu, které popisují jednotlivé části práce. Prezentace mají sloužit jako snadný přehled jednotlivých témat, popisu vlastností (Obrázek 70), způsobu provedení, implementace aplikačního řešení a zopakování dané problematiky (Obrázek 71).



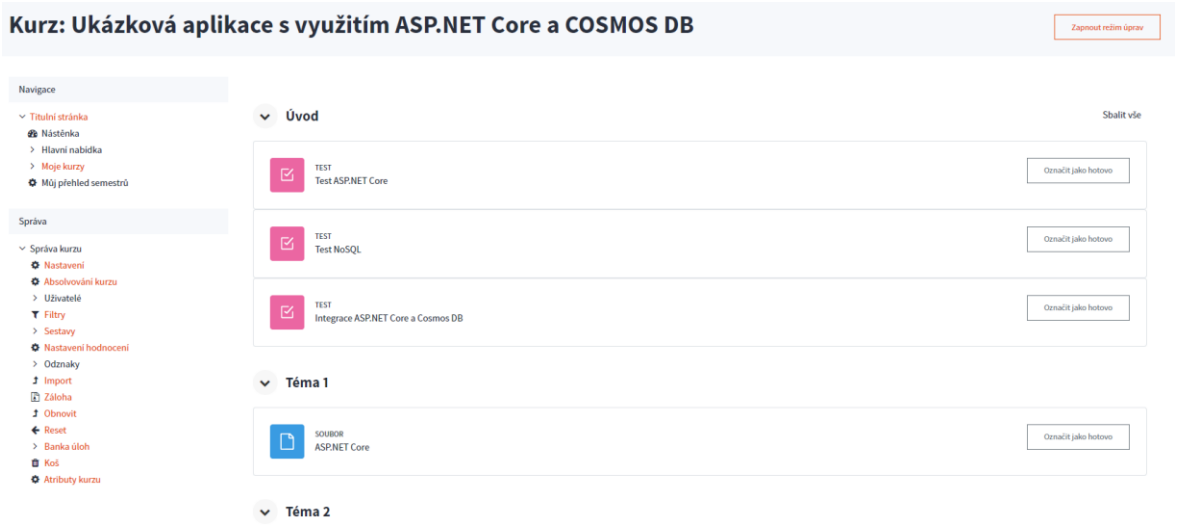
Obrázek 70 Ukázka prezentace vlastností ASP.NET Core



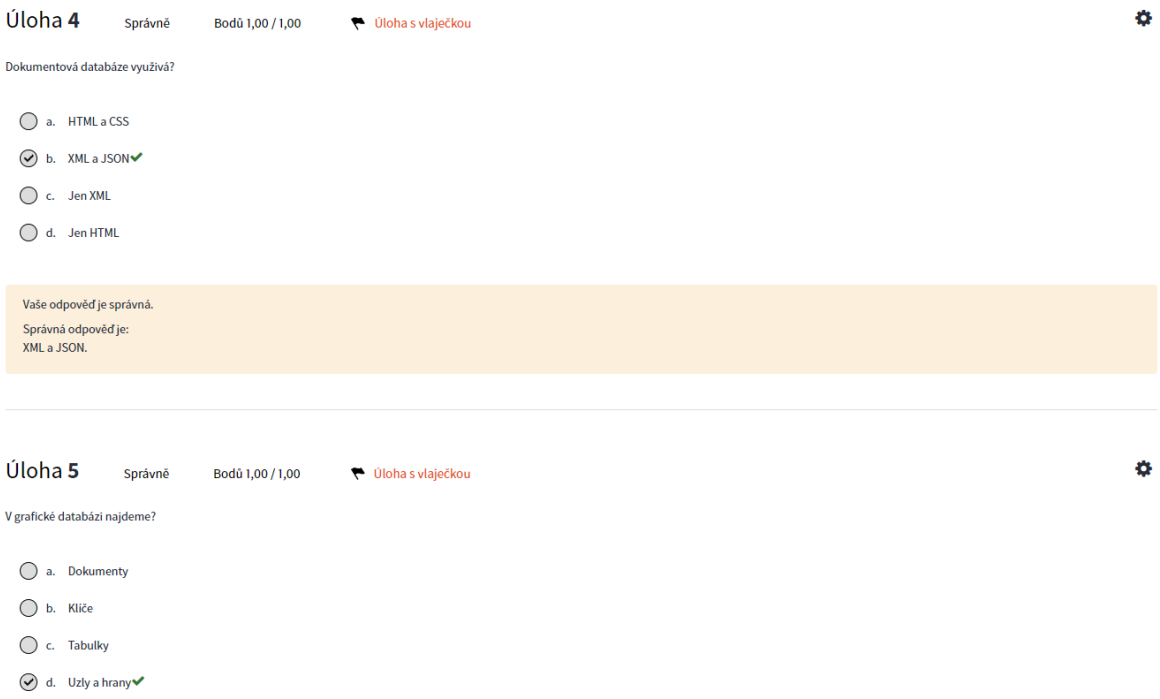
Obrázek 71 Ukázka z prezentací, otázky na zopakování

5.7 TESTOVACÍ ÚKOLY V MOODLE

Na základě prezentací byly vytvořeny testovací úkoly pro ověření znalostí. Testovací úkoly jsou ve formátu výběru jedné správné odpovědi z více možností (Obrázek 73).



Obrázek 72 Kurz v Moodle s testovacími úkoly

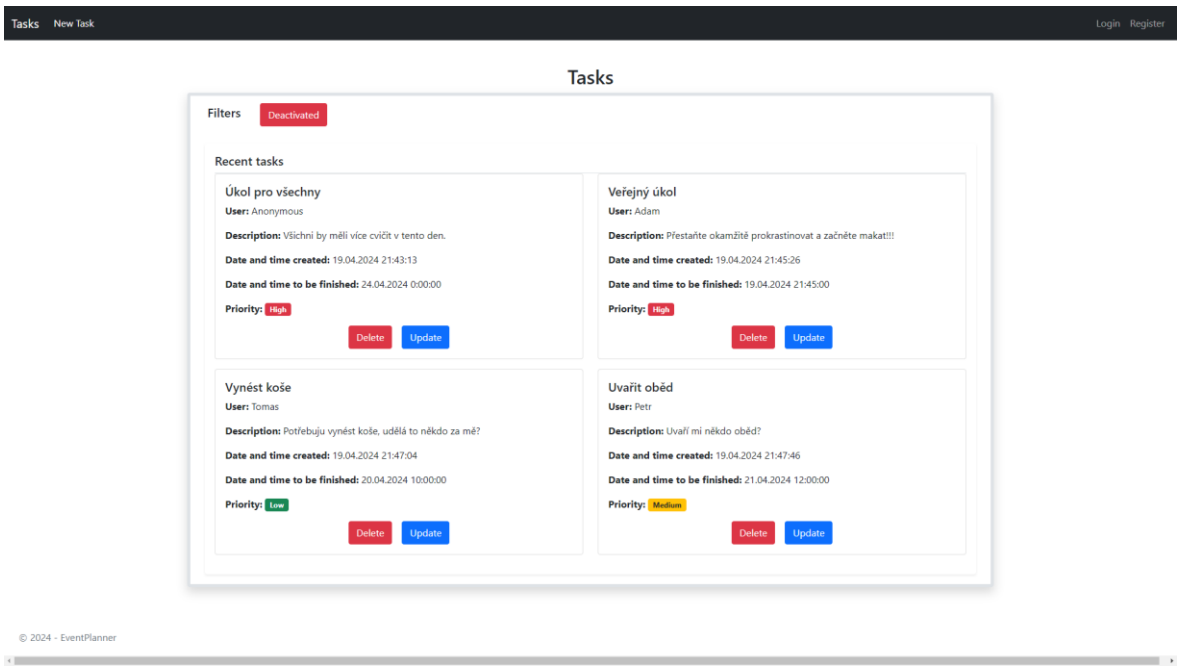


Obrázek 73 Testovací otázky v Moodle

6 POPIS APLIKACE

Vytvořená aplikace obsahuje celkem deset stránek, které každá plní svou vlastní funkcionalitu. Jsou to stránky pro autentizaci – vytvoření nových uživatelů (Obrázek 85) a přihlášení (Obrázek 86), stránka s úkoly – na této stránce uživatel vidí všechny úkoly, které jsou buď privátní, veřejné nebo pro přátele, tato stránka je taky výchozí jako domovská a je zobrazována všem (Obrázek 74). Dále jsou zde stránky s událostmi (Obrázek 78), uživatelským profilem (Obrázek 83), přáteli (Obrázek 82), kalendářem (Obrázek 81), dokončenými úkoly (Obrázek 80) a stránky pro vytváření nových úkolů (Obrázek 79) a událostí. Přístupnost jednotlivých stránek a jejich funkcionalit je dána na základě podmínky, jestli uživatel přihlášený nebo je jako anonym. Uživatel, který není přihlášený má možnost vidět stránku úkolů, možnost vytvářet nové úkoly a může se přihlásit nebo registrovat. Uživatel, který má vytvořený účet není nijak omezen.

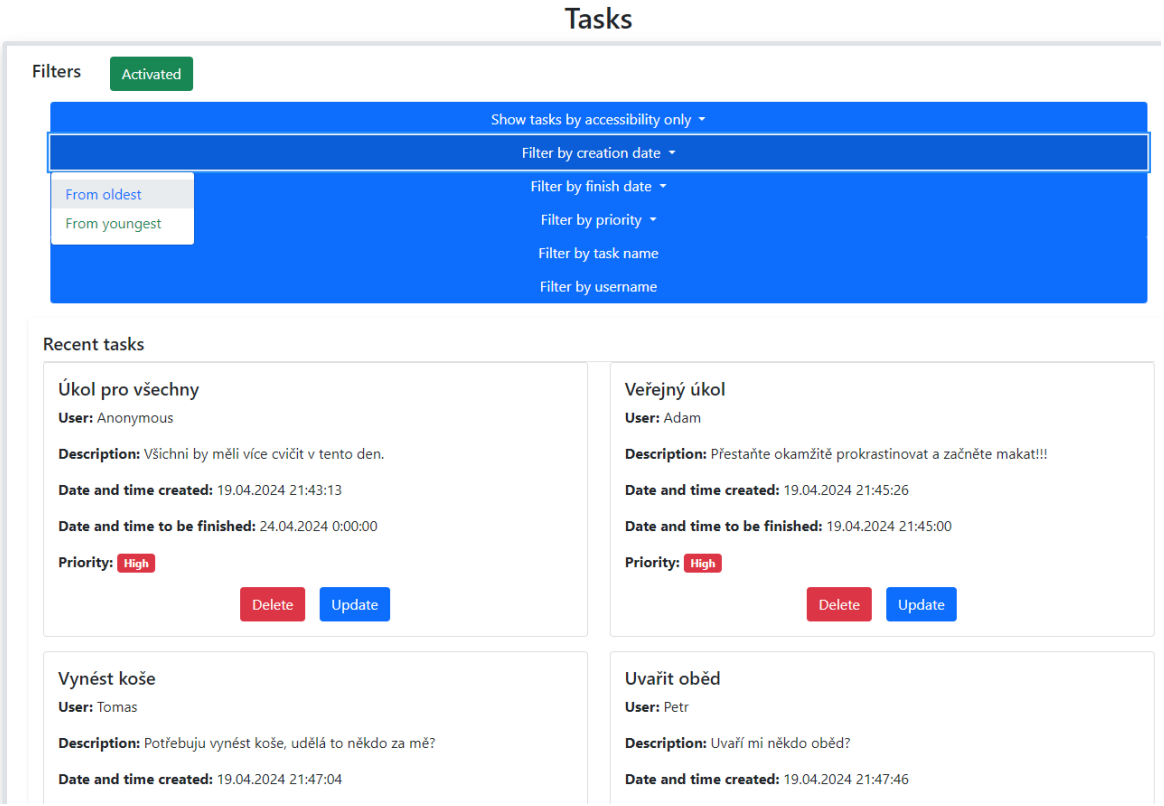
6.1 STRÁNKY ÚKOLŮ A UDÁLOSTÍ



Obrázek 74 Zobrazení aplikace pro nepřihlášeného uživatele

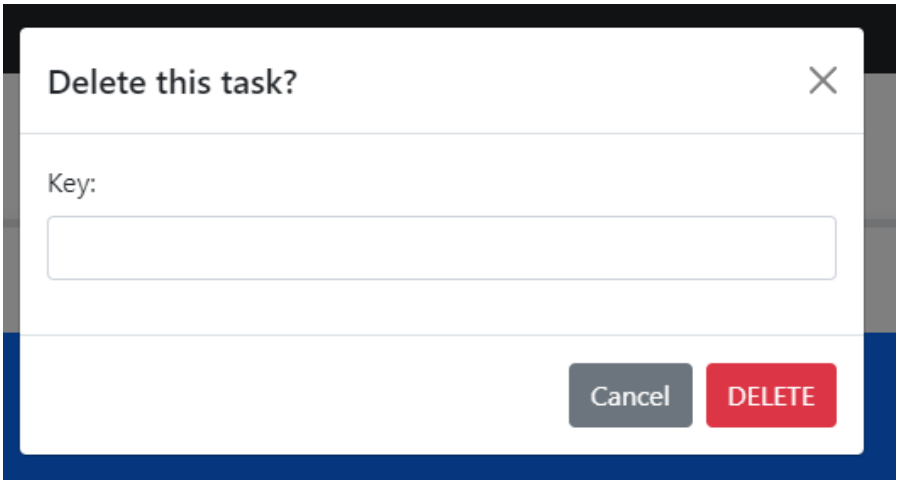
Na obrázku je zobrazená stránka z pohledu nepřihlášeného uživatele. Na stránce se nachází čtyři veřejné úkoly, které byly vytvořeny různými uživateli, ať už přihlášenými nebo anonymními. Každý úkol je znázorněn jako karta a je vytvořen na základě modelu úkolu, který charakterizuje jeho vlastnosti jako jsou – název, popisek, čas vytvoření, čas pro dokončení, jeho priorita a přístupnost (zobrazená jen pro přihlášené uživatele).

Všechny úkoly je možné na stránce filtrovat na základě jejich vlastností. Na následujícím obrázku je znázorněn jeden z filtrů úkolů:



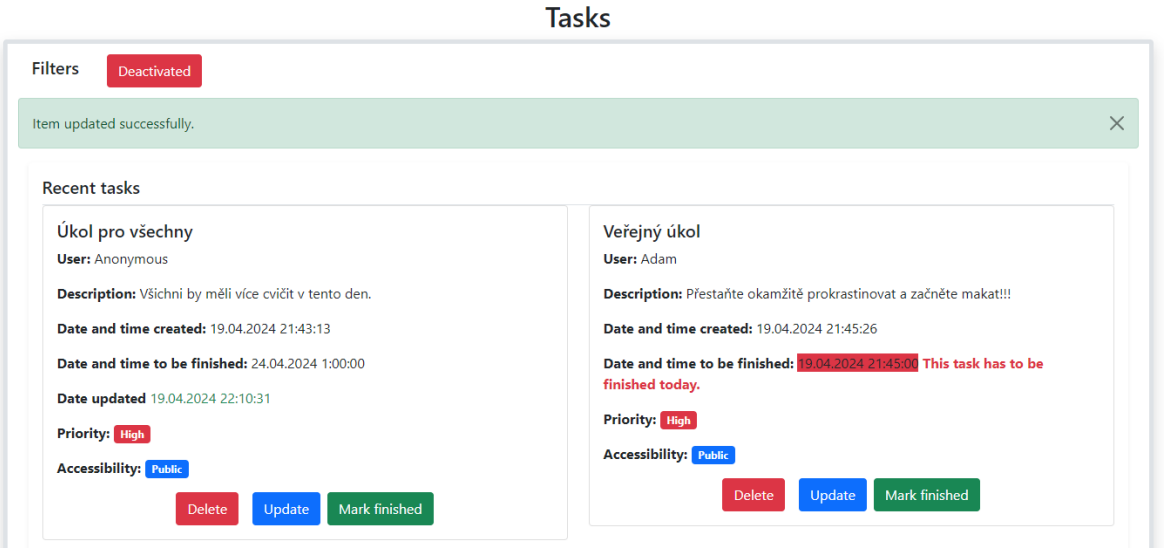
Obrázek 75 Ukázka filtrování úkolů podle času vytvoření od nejstaršího

Úkoly je možné dále mazat (Obrázek 76) nebo upravovat. Při kliknutí na tlačítko se otevře modalové okno, na kterém je uživatel dotázán, zda skutečně chce smazat nebo upravit úkol. Protože uživatel není přihlášen, musí také zadat klíč. Tento klíč je zadáván při vytváření nesoukromých úkolů pro zabezpečení proti smazání jiným uživatelem.



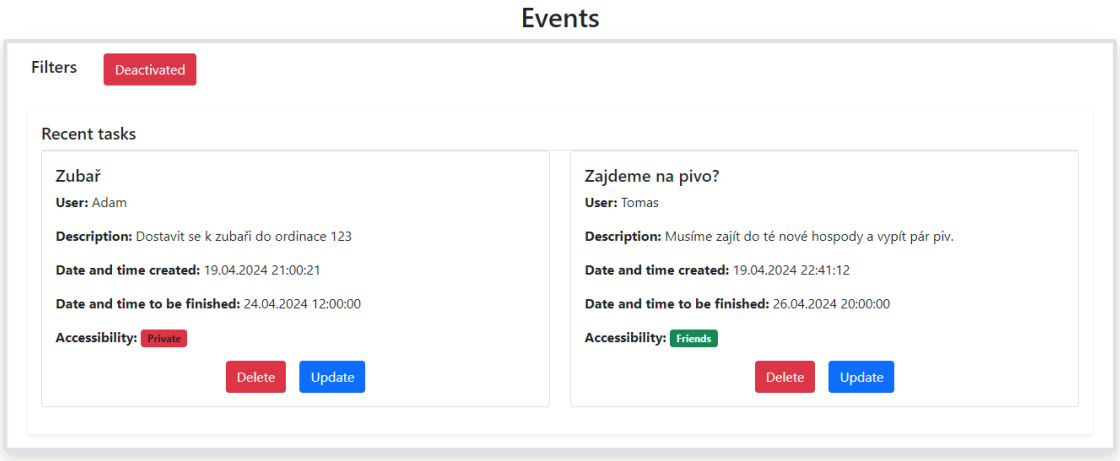
Obrázek 76 Modalové okno pro smazání úkolu

Uživatel, který je přihlášen může navíc úkoly dokončit. Po správném zpracování operací aktualizace, smazání nebo dokončení úkolů je uživateli zobrazena hláška s úspěchem.



Obrázek 77 Zobrazení hlášky s úspěchem aktualizace úkolu

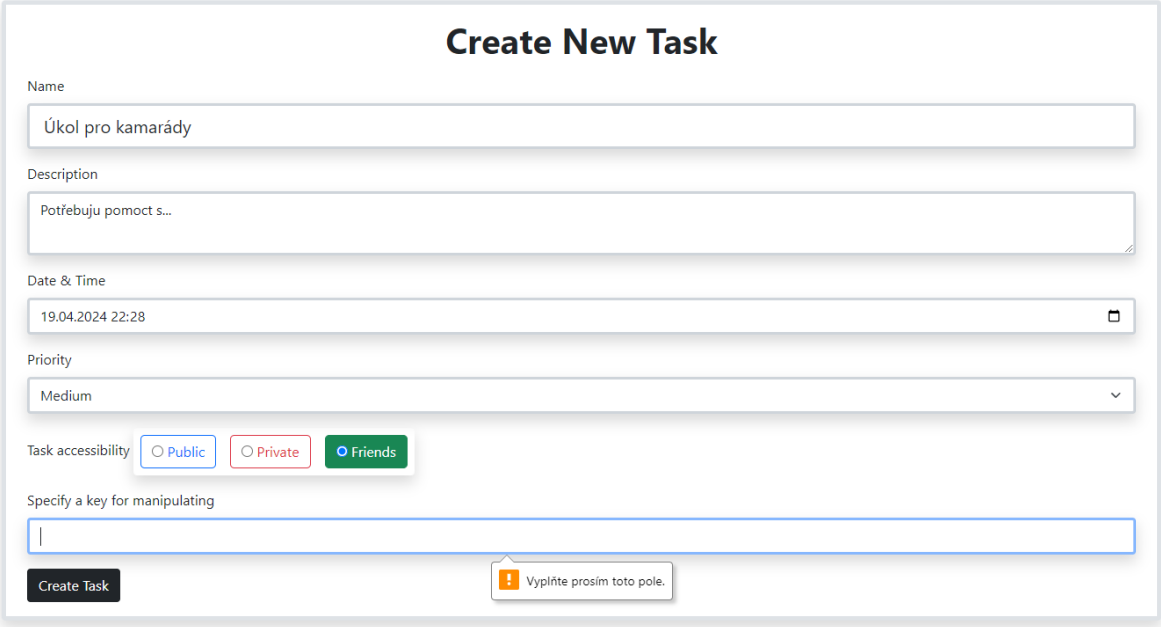
Podobným způsobem jako byla vytvořena stránka zobrazující úkoly, byla vytvořena i stránka s událostmi. Jejich funkcionalita se tak příliš neliší.



Obrázek 78 Zobrazení uživatelských událostí

6.2 STRÁNKA VYTVOŘENÍ NOVÉHO ÚKOLU

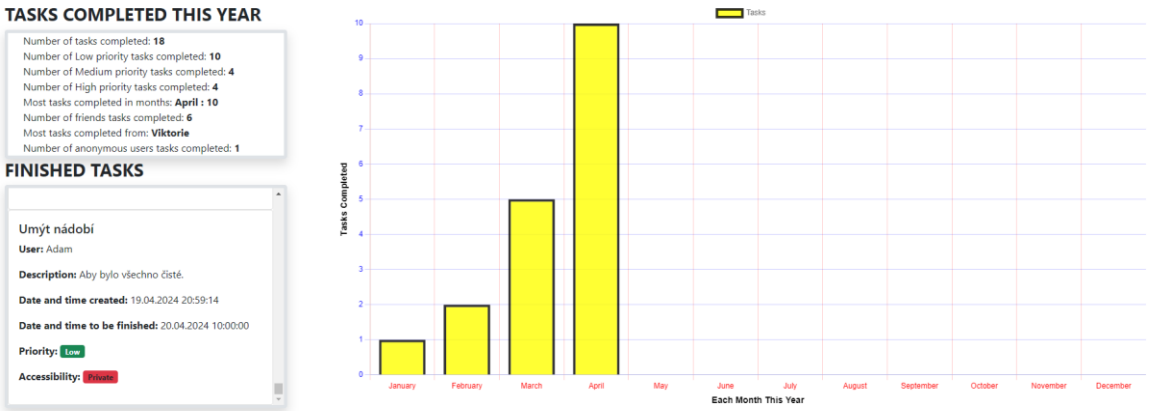
Na této stránce je možné vytvořit všechny nové úkoly. Aby bylo možné nový úkol vytvořit je nutné vyplnit všechny pole. Vytváření nových úkolů je tak validováno a ošetřeno proti prázdným hodnotám.



Obrázek 79 Stránka s vytvořením nového úkolu z pohledu přihlášeného uživatele

6.3 STRÁNKA S DOKONČENÝMI ÚKOLY

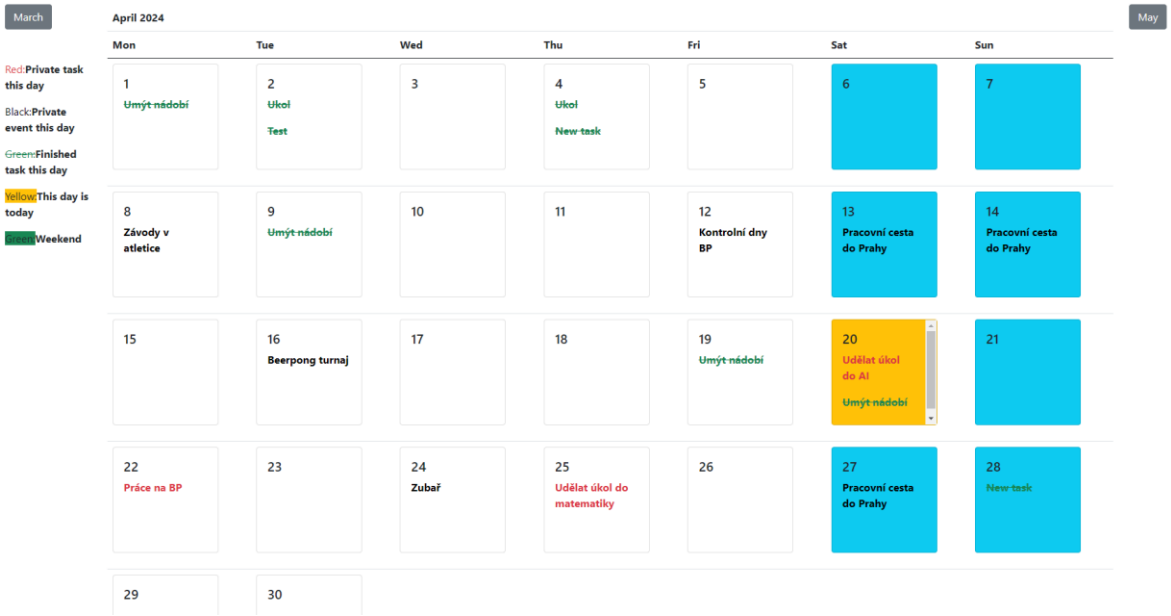
Na této stránce uživatel vidí v grafickém zobrazení všechny své dokončené úkoly v tomto roce. Jsou zde taky statistiky, které charakterizují například celkový počet dokončených úkolů nebo rozdělení podle priority. Uživatel je taky může jednotlivě procházet.



Obrázek 80 Stránka zobrazující dokončené úkoly

6.4 STRÁNKA KALENDÁŘE

Stránka kalendáře přehledně zobrazuje všechny soukromé úkoly a události, které má uživatel naplánované na jednotlivé dny. Je zde rozlišeno, zda byl nebo nebyl úkol dokončen. V kalendáři je taky barevně oddělen aktuální den a víkendy. Měsíce v kalendáři je možné přepínat.



Obrázek 81 Uživatelský kalendář jednotlivých událostí a úkolů

6.5 STRÁNKA PŘÁTEL, ŽÁDOSTÍ A UŽIVATELŮ

Na stránce přátel je možné vidět všechny své přátele, žádosti o přátelství anebo návrhy na přátelství existujících uživatelů. Uživatel tak může posílat, přijímat žádosti anebo odstranit přátele ze seznamu přátel. Na jednotlivé profily uživatelů je taky možné se prokliknout přes odkaz.

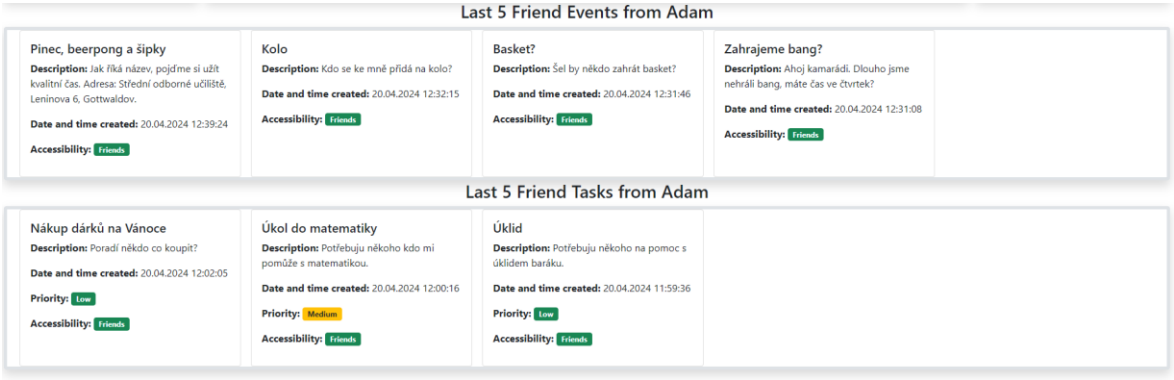
Obrázek 82 Stránka s přáteli, žádostmi a uživateli

6.6 STRÁNKA UŽIVATELSKÉHO PROFILU

Na stránce svého osobního profilu může uživatel vidět své informace, statistiky a přátele. Mezi profily svých přátel může libovolně procházet.

Obrázek 83 Profil uživatele, zobrazený jiným uživatelem.

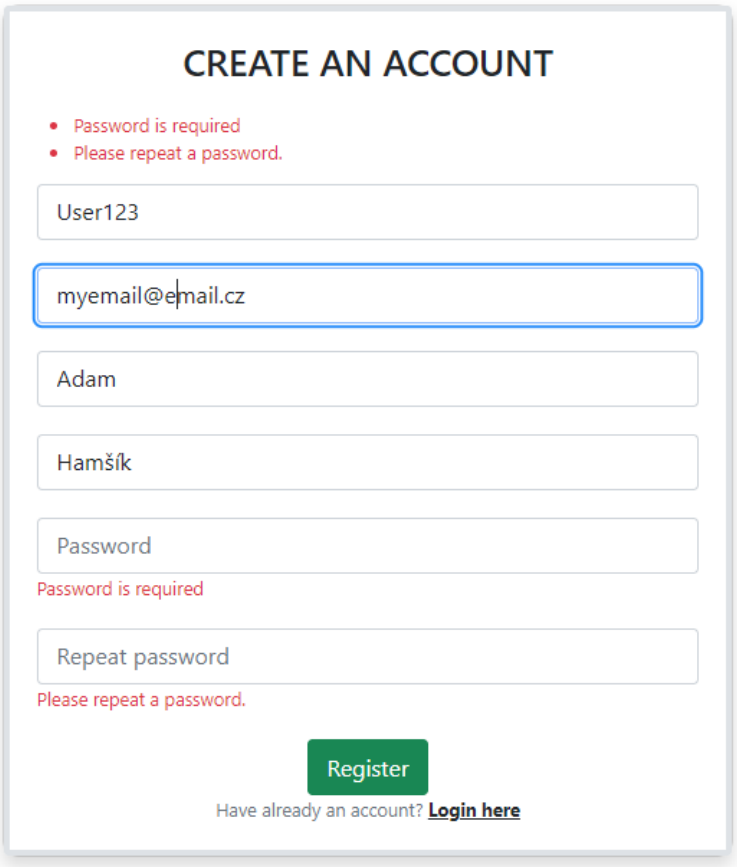
V případě zobrazení profilu přítele může uživatel vidět taky několik poslední úkolů a událostí, které přítel vytvořil a jsou dostupné jen pro přátele.



Obrázek 84 Zobrazení posledních pěti úkolů a událostí od přítele

6.7 STRÁNKY REGISTRACE A PŘIHLÁŠENÍ

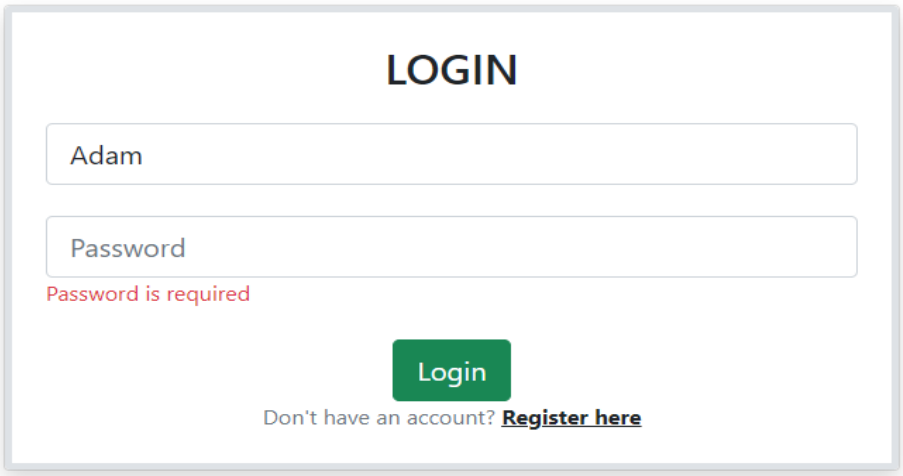
Stránky registrace a přihlášení se zobrazují uživatelům, kteří již nejsou přihlášení. V případě registrace zde potencionální nový uživatel zadává své údaje jako jsou – uživatelský username, email, jméno, příjmení a heslo.



Obrázek 85 Ukázka registrace nového uživatele

Aplikace kontroluje všechny pole, které uživatel zadává, nemůže se tak stát, že by existovali dva uživatelé se stejným username nebo emailem. Zadaná hesla se musí shodovat a mají nastavená jistá bezpečností pravidla.

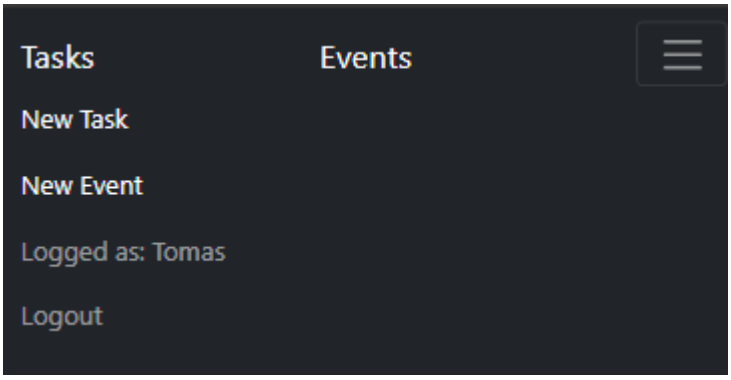
Uživatel je po úspěšné registraci přesměrován na stránku přihlášení a má možnost se přihlásit s nově vytvořeným účtem pomocí zadaného username.



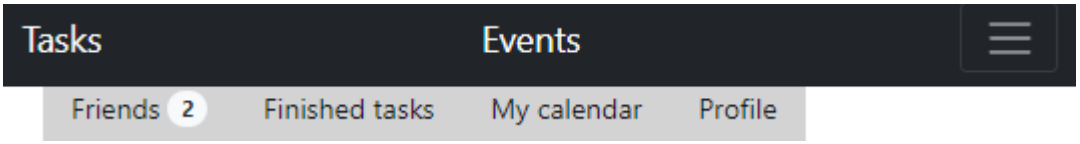
Obrázek 86 Okno pro přihlášení uživatele

6.8 NAVIGAČNÍ LIŠTY

Mezi všemi výše popsanými stránkami je možné procházet pomocí navigačních lišt, které jsou zobrazeny v horní části aplikace. Přihlášený uživatel vidí obě lišty a všechny stránky.



Obrázek 87 První navigační lišta, zobrazená na menším zařízení



Obrázek 88 Druhá navigační lišta

ZÁVĚR

V této práci bylo seznámeno s moderními technologiemi pro vývoj webových aplikací a jejich příslušenstvím. Byla také vytvořena jednoduchá aplikace pro správu úkolů, která má sloužit jako podklad pro pochopení a dále ukázkový příklad tvorby webové stránky.

V teoretické části byly popsány klíčové technologie ASP.NET Core a Cosmos DB a jejich vlastnosti a principy, na kterých fungují. Také byl znázorněn a popsán způsob integrace mezi těmito technologiemi. Pro účely práce bylo využito bezplatného emulátoru, který byl vhodný pro jednoduché nasazení na lokálním zařízení. Byly popsány moderní HTML frameworky, které jsou nezbytnou součástí webových aplikací, díky nim jsou webové aplikace uživatelsky přívětivé a podnikově úspěšné. Byl vybrán jeden z frameworků, který byl dále využit při tvorbě aplikace v praktické části.

V praktické části byl stanoven návrh aplikace s definovanými požadavky a vlastnostmi, které by aplikace měla splňovat. Byla popsána struktura aplikace a praktické způsoby integrace s databází a dat, které se v ní nacházejí. Popsány byly klíčové funkce pro základní a správné fungování aplikace. Následně byly vytvořeny pedagogické materiály ve snaze zopakování a možnosti vlastní tvorby. Úkoly mají charakterizovat snahu naučit implementovat způsoby vlastního provedení a seznámení s problematikou. Byl vytvořen kurz v systému Moodle, kde se tyto materiály nacházejí i s testovacími otázkami pro zopakování tématu.

SEZNAM POUŽITÉ LITERATURY

[1] FOWLER, Adam. NoSQL for dummies. John Wiley, 2015. ISBN 9788126554904.

[2] Introduction to Graph Database on NoSQL. GeeksforGeeks [online]. 2023 [cit. 2024-02-19]. Dostupné z: https://www.geeksforgeeks.org/introduction-to-graph-database-on-nosql/

[3] NoSQL Databases: Advantages and Disadvantages. Dataversity [online]. 2022 [cit. 2024-02-25]. Dostupné z: https://www.dataversity.net/nosql-databases-advantages-and-disadvantages/

[4] Database Comparison - SQL vs. NoSQL. Profilsoftware [online]. 2019 [cit. 2024-02-25]. Dostupné z: https://profil-software.com/blog/development/database-comparison-sql-vs-nosql-mysql-vs-postgresql-vs-redis-vs-mongodb/

[5] NoSQL Beginner Guide: Pros, Cons, Types, and Philosophy. Altexsoft [online]. 2021 [cit. 2024-02-25]. Dostupné z: https://www.altexsoft.com/blog/nosql-pros-cons/

[6] Advantages and disadvantages of NoSQL. Macrometa [online]. 2024 [cit. 2024-02-26]. Dostupné z: https://www.macrometa.com/distributed-data/advantages-and-disadvantages-of-nosql

[7] 7 Key Features of Azure Cosmos DB. Aarete [online]. 2020 [cit. 2024-02-26]. Dostupné z: https://www.aarete.com/insights/7-key-features-of-azure-cosmos-db/

[8] How do you speed up your development process with HTML and CSS frameworks? LinkedIn [online]. 2023 [cit. 2024-02-26]. Dostupné z: https://www.linkedin.com/advice/0/how-do-you-speed-up-your-development-process

[9] 8 Best HTML5 Frameworks for Front-End Development 2024. Colorlib [online]. 2024 [cit. 2024-02-26]. Dostupné z: https://colorlib.com/wp/html5-frameworks/

[10] YÖNDEM, Daron a Gaston HILLAR. Guide to NoSQL with Azure Cosmos DB: Work with the massively scalable Azure database service with JSON, C#, LINQ, and. NET Core 2. Packt, 2018. ISBN 1789612896.

[11] Foundation. Foundation Framework [online]. 2024 [cit. 2024-02-28]. Dostupné z: https://get.foundation/

[12] Přehled ASP.NET. Learn.microsoft [online]. 2023 [cit. 2024-02-28]. Dostupné z: https://learn.microsoft.com/cs-cz/aspnet/overview

[13] Foundation (framework). Wikipedia [online]. 2024 [cit. 2024-02-28]. Dostupné z: https://en.wikipedia.org/wiki/Foundation_(framework)

[14] Benefits of Bootstrap for Web Design. Clarityventures [online]. 2024 [cit. 2024-02-28]. Dostupné z: https://www.clarity-ventures.com/blog/benefits-of-using-bootstrap-for-web-design

[15] What is Bootstrap & Advantages of Bootstrap in Web Development. Cybersuccess [online]. 2021 [cit. 2024-02-28]. Dostupné z: https://www.cybersuccess.biz/advantages-of-bootstrap/

[16] Build fast, responsive sites with Bootstrap. Getbootstrap [online]. 2024 [cit. 2024-02-28]. Dostupné z: https://getbootstrap.com/

[17] Foundation: The Best Framework for Building Responsive Sites. Piecesfordevelopers [online]. 2023 [cit. 2024-02-28]. Dostupné z: https://code.pieces.app/blog/foundation-the-best-framework-for-building-responsive-sites

[18] Create your site quickly and effectively with Metro 4. MetroUI [online]. 2024 [cit. 2024-03-05]. Dostupné z: https://www.metroui.org.ua/~index.html

[19] ASP.NET Core. Dotnetmicrosoft [online]. 2024 [cit. 2024-03-06]. Dostupné z: https://dotnet.microsoft.com/en-us/apps/aspnet

[20] Web Framework Benchmarks. Techempower [online]. 2022 [cit. 2024-03-06]. Dostupné z: https://www.techempower.com/benchmarks/#section=data-r21&hw=ph&test=plaintext&l=fsu2yk-cfx&c=a

[21] Differences Between SQL vs NoSQL. Scylla [online]. 2024 [cit. 2024-03-06]. Dostupné z: https://www.scylladb.com/learn/nosql/nosql-vs-sql/

[22] What is Azure? Azuremicrosoft [online]. 2024 [cit. 2024-03-06]. Dostupné z: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-azure

[23] What is the cloud? | Cloud definition. Cloudflare [online]. 2024 [cit. 2024-03-06]. Dostupné z: https://www.cloudflare.com/learning/cloud/what-is-the-cloud/

[24] What is the Azure Cosmos DB emulator? Learn.microsoft [online]. 2023 [cit. 2024-03-09]. Dostupné z: https://learn.microsoft.com/en-us/azure/cosmos-db/emulator

[25] Partitioning and horizontal scaling in Azure Cosmos DB. Learn.microsoft [online]. 2023 [cit. 2024-03-11]. Dostupné z: https://learn.microsoft.com/en-us/azure/cosmos-db/partitioning-overview

[26] ASP.NET Core Basics: ASP.NET Core Overview. Telerik [online]. 2023 [cit. 2024-04-06]. Dostupné z: https://www.telerik.com/blogs/aspnet-core-basics-aspnet-core-overview

[27] Build your first web app with ASP.NET Core using Blazor. Dotnet.microsoft [online]. 2024 [cit. 2024-04-06]. Dostupné z: https://dotnet.microsoft.com/en-us/learn/aspnet/blazor-tutorial/try

[28] MVC Design Pattern. Geeksforgeeks [online]. 2024 [cit. 2024-04-06]. Dostupné z: https://www.geeksforgeeks.org/mvc-design-pattern/

[29] The basics of NoSQL databases—and why we need them. Freecodecamp [online]. 2019 [cit. 2024-04-06]. Dostupné z: https://www.freecodecamp.org/news/nosql-databases-5f6639ed9574/

[30] What is a Key-Value Database? Redis [online]. 2024 [cit. 2024-04-06]. Dostupné z: https://redis.com/nosql/key-value-databases/

[31] Graph Databases: The Future of Scalable and Flexible Data Management for Complex Relationship-driven Applications. Linkedin [online]. 2023 [cit. 2024-04-06]. Dostupné z: https://www.linkedin.com/pulse/graph-databases-future-scalable-flexible-data-complex-

[32] Horizontal Scaling Vs. Vertical Scaling: An In-Depth Look. Cloudzero [online]. 2023 [cit. 2024-04-06]. Dostupné z: https://www.cloudzero.com/blog/horizontal-vs-vertical-scaling/

[33] Metro UI your way to build a web application. Korzh [online]. 2023 [cit. 2024-04-06]. Dostupné z: https://korzh.com/metroui

[34] Develop locally using the Azure Cosmos DB emulator. Learn.microsoft.com [online]. 2023 [cit. 2024-04-06]. Dostupné z: https://learn.microsoft.com/en-us/azure/cosmos-db/how-to-develop-emulator?tabs=windows%2Ccsharp&pivots=api-nosql

[35] Cosmos DB Provider for ASP.NET Core Identity. Github [online]. 2024 [cit. 2024-04-08]. Dostupné z: https://github.com/MoonriseSoftwareCalifornia/AspNetCore.Identity.CosmosDb

[36] Tag Helpers in ASP.NET Core. Learn.microsoft [online]. 2023 [cit. 2024-04-18]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers/intro?view=aspnetcore-8.0¨

[37] Are HTML, CSS, And JavaScript Enough? (Front-End Framework). Bigscal [online]. 2022 [cit. 2024-04-18]. Dostupné z: https://www.bigscal.com/blogs/frontend/are-html-css-and-javascript-enough-for-front-end-framework/

[38] Semantic UI User Interface is the language of the web. Semantic-ui [online]. 2024 [cit. 2024-04-22]. Dostupné z: https://semantic-ui.com/

[39] Semantic UI. Github [online]. 2024 [cit. 2024-04-22]. Dostupné z: https://github.com/Semantic-Org/Semantic-UI

[40] Metro UI logo. Metroui [online]. 2024 [cit. 2024-04-22]. Dostupné z: https://metroui.org.ua/icons.html

[41] Semantic UI logo. Iconduck [online]. 2020 [cit. 2024-04-22]. Dostupné z: https://iconduck.com/icons/94872/semantic-ui

[42] Materialize A modern responsive front-end framework based on Material Design. Materializecss [online]. 2024 [cit. 2024-04-22]. Dostupné z: https://materializecss.com/

[43] Materialize CSS. Geeksforgeeks [online]. 2023 [cit. 2024-04-22]. Dostupné z: https://www.geeksforgeeks.org/materialize-css/

[44] Materialize Logo PNG Vector. Seeklogo [online]. 2024 [cit. 2024-04-22]. Dostupné z: https://seeklogo.com/vector-logo/296808/materialize

[45] Bootstrap: Should I use it as a Developer? Medium [online]. 2022 [cit. 2024-04-22]. Dostupné z: https://medium.com/@simply_stef/bootstrap-should-i-use-it-as-a-developer-e7c7d0d26ff0

[46] Semantic UI Guide. Freecodecamp [online]. 2020 [cit. 2024-04-22]. Dostupné z: https://www.freecodecamp.org/news/semantic-ui-guide/

[47] How to prototype websites with Justinmind's Foundation UI kit. Medium [online]. 2017 [cit. 2024-04-22]. Dostupné z: https://medium.com/justinmind/how-to-prototype-websites-with-justinminds-foundation-ui-kit-5192d3e12ecd

[48] Unique key constraints in Azure Cosmos DB. Learn.microsoft [online]. 2022 [cit. 2024-04-25]. Dostupné z: https://learn.microsoft.com/en-us/azure/cosmos-db/unique-keys

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
MVC	Model View Controller
API	Application Programming Interface
NoSQL	Not only SQL
SQL	Structured Query Language
JSON	JavaScript Object Notation
XML	Extensible Markup Language
RDBMS	Relational Database Management System
ACID	Atomicity Consistency Isolation Durability
SLA	Service Level Agreement
RU	Request Unit
kB	kilobyte
UI	User Interface
CRUD	Create Read Update Delete
CSFR	Cross Site Request Forgery

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: CD DISK

Přiložené CD obsahuje složku s prezentacemi a složku s vytvářenou aplikací a diagramy vytvořenými v programu EA