Vyšší odborná škola a Střední průmyslová škola, Jičín, Pod Koželuhy 100
18-20-M/01 Informační technologie

Aplikace pro MS Teams

Individuální maturitní práce

Jičín 2023 Jiří Navrátil

Obor vzdělávání: Informační technologie **Číslo oboru: 18-20-M/01**

Třída: I4B

Školní rok: 2022-2023

Téma maturitní práce: Aplikace pro MS Teams pro výuku
Vedoucí maturitní práce: Mgr. Ilona Hamanová
Oponent maturitní práce: Václav Doškář
Téma zpracuje žák: Jiří Navrátil

Zadání maturitní práce:

Cílem práce je naprogramovat aplikaci pro MS Teams podporující výuku. Aplikace učiteli umožní zadat kapitoly, jejich obsah, podmínky pro úspěšné splnění a vztahy (prerekvizity) mezi kapitolami. Žákům by pak aplikace měla umožňovat co nejsamostatnější procházení kapitolami na základě plnění úkolů požadovaných pro splnění jednotlivých kapitol a tím zpřístupnění kapitol dalších. Učitel bude mít také k dispozici přehledy o plnění kapitol a úkolů jednotlivými žáky.

Práce bude obsahovat:

V teoretické části:

- popis řešeného úkolu, jeho plánované vlastnosti
- výběr vývojových prostředků pro řešení
- stručný popis jednotlivých funkcí z hlediska uživatele, včetně jejich významu pro celou aplikaci i zamýšlenou podporu výuky

V praktické části:

- řešení GUI a jeho zdůvodnění
- popis celkové koncepce programu
- vysvětlení řešení podstatných částí programu

V přílohách:

- funkční program v podobě zdrojového kódu i v podobě k distribuci
- stručný návod k používání aplikace učitelem i žákem

Způsob zpracování a kritéria hodnocení jsou uvedeny v dokumentu Metodický návod na tvorbu individuální maturitní práce 2022-23.pdf, který je umístěn v týmu individuálních maturit prostřednictvím MS Teams.

Řešitel práce předloží vedoucímu práce svoje řešení k posouzení (elektronicky prostřednictvím Microsoft Teams):

1. konzultace (úvodní rešerše, návrh koncepčního řešení): 30. 10. 2022
2. konzultace (první průběžné řešení): 27. 11. 2022
3. konzultace (druhé průběžné řešení): 20. 12. 2022
4. konzultace (třetí průběžné řešení): 15. 1. 2023
5. konzultace (návrh konečného řešení s návrhem prezentace): 26. 2. 2032

Odevzdání práce: 17. března 2023

Vypracovanou práci odevzdá žák elektronicky prostřednictvím aplikace Microsoft Teams, dále pak zástupcům ředitelky jednu vytištěnou písemnou práci nejdéle do **17.března 2023**.
Vytištěné práce slouží jako doklad pro případnou kontrolu dosaženého stupně vzdělání a zůstávají v majetku školy. Jsou uloženy v archivu knihovně školy, přístupné žákům školy ke studiu.

…… ……

Podpis studenta **Podpis vedoucího práce**

Prohlašuji, že jsem práci na téma aplikace pro MS Teams zpracoval samostatně a uvedl v ní všechny prameny, literaturu a ostatní zdroje, které jsem použil.

V Jičíně 17. 3. 2023

Anotace

Práce se zabývá vytvořením aplikace pro MS Teams, která poskytuje studentům efektivní a individualizovanou výuku, která vypadá jako seznam témat. Aplikace je rozdělena do třech hlavních oprávnění, a to na učitele, studenta a správce sítě.

Učitel může vytvořit v aplikaci pod MS týmem strukturu (předmět), kde bude mít oprávnění přidávat, upravovat či mazat jednotlivé učební kapitoly. Kapitoly jsou mezi sebou navázané pomocí prerekvizit. Po korekci studentova zadání může danou kapitolu studentovi označit jako úspěšně x neúspěšně splněnou. Může si zobrazit souhrn studentů, kde vidí individuální postup každého studenta.

Student může individuálně procházet danou látkou ve formě kapitol, kde na závěr každé kapitoly vyplní zadání (např: v MS Forms). Po úspěšném splnění zadání může student získat přístup k další kapitole. Odemykání kapitol závisí na prerekvizitách, které jsou potřeba splnit.

Klíčová slova

Microsoft Teams, Blazor, Figma, Bootstrap, C#, HTML, CSS, SQL, Entity Framework

Annotation

The thesis deals with the creation of an application for MS Teams that provides students with an effective and personalized learning experience that looks like a list of topics. The application is divided into two main permissions, namely teacher and student.

The teacher can create a structure (subject) in the application under MS group where he/she will have the permission to add, edit or delete individual learning chapters. The chapters are linked to each other using pre-requisites. After correcting the student's assignment, he/she can mark the chapter as pass x fail. He can view a summary of the students to see each student's individual progress.

The student can individually go through the material in the form of chapters, where at the end of each chapter he/she completes the assignment (e.g.: in MS Forms). After successfully completing the assignment, the student can access the next chapter. Unlocking chapters depends on the prerequisites that need to be fulfilled.

Keywords

Microsoft Teams, Blazor, Figma, Bootstrap, C#, HTML, CSS, SQL, Entity Framework

Obsah

SLOVNÍK POUŽÍVANÝCH POJMŮ A ZKRATEK

MS Teams – Microsoft Teams

MS Forms – Microsoft Forms

MS Graph – Microsoft Graph

EF Core – Entity Framework Core

HTML – Hypertext Markup Language

CSS – Cascading Style Sheets

SQL – Structured Query Language

LINQ – Language Integrated Query

JSON – JavaScript Object Notation

ASP.NET – Application Service Providing

API – Application Programming Interface

SDK – Software development kit

ID – identifikátor

DB – Databáze

UX – User experience

GUI – Grafické uživatelské rozhraní

MS tým – Vytvořený Tým v Microsoft Teams

Skupina – Určitá vytvořená skupina v aplikaci v MS týmu uložena v DB. Jednotlivá skupina má svoji strukturu. Se skupinou jsou spjaty i informace o studentech a záznamy jejich pokroku v plnění kapitol v DB.

Struktura – Jiným názvem předmět, který obsahuje jednotlivé kapitoly z DB.

Prerekvizita – Kapitola potřebná pro odemčení jiné kapitoly.

1 ÚVOD

V této práci se vytvoří aplikace pro MS Teams, která poskytne studentům efektivní a individualizovanou výuku. To v praxi znamená, že student, který ovládá lépe daný předmět se může více rozvíjet ve svých znalostech. Výuková aplikace bude vypadat jako seznam témat, kde každý student bude procházet jednotlivé kapitoly a plnit úkoly v nich zadané.

Nejdříve je potřeba návrh aplikace, aby bylo možné určit požadavky frontend. V rámci projektu se bude používat framework Blazor. Dále bude nutné ukládat studenta včetně jeho uživatelských dat, jako jsou například zaznamenané splněné kapitoly. Taktéž bude potřeba ukládat kapitoly, které budou obsahovat textový obsah, odkazy a úkoly v podobě MS Forms. Po splnění úkolu a následné kontrole a zaznamenání učitelem bude student mít přístup k dalším kapitolám. Odemčení kapitol bude záviset na splnění prerekvizit jednotlivých kapitol. Pokud student splní všechny požadované prerekvizity, daná kapitola se odemkne a student bude moci k ní přistoupit. Pokud student nesplní požadované prerekvizity pro odemknutí určité kapitoly, bude mít přístup pouze k náhledu kapitoly. Bude také nutné zajistit správu oprávnění pro učitele a studenty. Učitel bude moci vidět progres všech studentů a zároveň přidávat či upravovat kapitoly, na což student nebude mít oprávnění.

Aplikace bude fungovat v rámci různých MS týmech v MS Teams, což znamená, že bude potřeba rozpoznat v jaké MS týmu se nachází, jaká strukturu je pro ni určena a které kapitoly patří do této struktury. Pro studenty bude důležité zjistit, jací studenti patří pod skupinu a zároveň ukládat celkový progres ve splnění jednotlivých kapitol.

2 ROZBOR APLIKACE

Aplikace bude fungovat v prostředí MS Teams a bude k dispozici ve vybraných MS týmech podle potřeb učitele. Následně v dané skupině bude zvolena struktura, která bude obsahovat jednotlivé kapitoly. Studenti, kteří jsou navázaní pod danou skupinou, budou mít evidováno splnění jednotlivých kapitol a jejich celkový progres v rámci této skupiny.

Aplikace bude rozdělena podle oprávnění na tři role: učitel, student a správce sítě. Pouze uživatelé s těmito oprávněními budou mít přístup k aplikaci.

2.1 Učitel

Po prvotním nahrání aplikace v MS Teams umožní aplikace vytvořit novou strukturu nebo vybrat ze seznamu již existujících veřejných struktur, které budou navázané na daný MS tým. Pokud učitel zvolí existující strukturu s kapitolami, snadno se překopíruje. Tímto způsobem se učitelům ulehčí práce s opakovaným vytváření stejných předmětu s kapitolami. Jednotlivé struktury jsou znovupoužitelné a zároveň si každý učitel po výběru existující struktury může libovolně upravit její obsah.

Pokud je v MS týmu v aplikaci vytvořena skupina, učitel, který ji vytvořil může:

- Vytvářet jednotlivé kapitoly obsahující nadpis, perex, obsah, internetové odkazy a odkazy na zadání například v podobě MS Forms či jiných aplikací třetích stran a prerekvizity, které student musí splnit pro odemčení kapitoly.
- Upravovat a mazat jednotlivé kapitoly. Mazání kapitol je možné, pokud daná kapitola není potřeba k odemčení jiné kapitoly (tj. není její prerekvizita).
- Procházet všechny vytvořené kapitoly.
- Zobrazit souhrn studentů a jejich plnění kapitol. V úvodním přehledu každého studenta bude zobrazen jeho počet splněných kapitol a celkový počet kapitol. Po rozkliknutí bude možné zobrazit přesný seznam splněných kapitol.
- Kontrolovat jednotlivé úkoly a následně označit u studenta splnění kapitoly (úspěšné x neúspěšné).
- Smazat daného studenta ze skupiny (například když daný student, odešel ze školy…).
- Měnit název pouze své struktury nebo vymazat či resetovat skupinu.
- Měnit viditelnost předmětu pro ostatní učitele. (Privátní x Veřejný).

Obrázek 1 Vytvoření struktury (předmětu) z pohledu učitele – návrh

Pokud nastane situace, že by v MS týmu působilo více učitelů, aplikaci mohou používat všichni s rozdílem, že učitel, který nevytvořil skupinu v aplikaci v určitém MS týmu, bude moci pouze číst všechny kapitoly. Nemůže nastavit skupinu ani zobrazit souhrn studentů či vytvořit novou kapitolu. Dá se říct, že je v read-only režimu.

2.2 Student

Pokud v MS týmu není vytvořena skupina včetně struktury (tedy není vytvořena v DB), student bude informován o této skutečnosti a bude muset počkat na vytvoření.

Pokud je skupina v MS týmu vytvořena student pak může:

- Procházet jednotlivé kapitoly, které jsou buď odemčené či splněné. U uzamčených kapitol bude moci pouze zobrazit nadpis a perex.
- Plnit jednotlivé kapitoly tak, že bude odkázán na úkol v podobě MS Forms nebo pomocí jiných aplikací třetích stran. Po odevzdání a kontrole učitelem bude

následující kapitola splněna v případě úspěšného výsledku a tím dojde k odemčení nových kapitol podle daných prerekvizit.

Díky těmto plnění kapitol a postupnému odemykání, aplikace provede studenty postupně danou látkou individuálně. Studenti, kteří danou látku lépe ovládají se mohou dostat k složitějším tématům a tím se lépe rozvíjet v daném předmětu či problematice.

Obrázek 2 Informace pro studenta, pokud skupina není vytvořena

2.3 Správce sítě

Má ve všech vytvořených skupinách stejná oprávnění jako ten, kdo je vytvořil (učitel), což znamená, že ke všem skupinám má plný přístup. Může vytvořit i svoji skupinu včetně struktury a zacházet s ní jako učitel.

3 PROSTŘEDKY PRO VÝVOJ

Tato část bude představovat MS Teams a s ní spojené služby, které se budou používat ve vývoji. V další části se do detailu rozebere, v čem a co bude použito při vývoji aplikace pro MS Teams. Jedná se pak o návrh, frontend a backend aplikace.

3.1 Microsoft Teams

Jedná se o nástroj pro komunikaci vyvíjený společností Microsoft, který patří do rodiny produktů Microsoft 365 dříve Office 365. Spadá pod proprietární software, což znamená, že se jedná o software s uzavřeným kódem. Microsoft Teams je k dispozici pro použití na různých platformách, včetně Windows, macOS, iOS a Android. [1]

Microsoft Teams umožňuje nejen chat mezi uživateli, ale i možnosti pořádat virtuální schůzky a videokonference. Dále obsahuje možnost vytvářet jednotlivé skupiny a s ní samotnou správu uživatelů s oprávněními. V jednotlivých skupinách jsou nadále i konverzační kanály, které slouží k rozdělení komunikace napříč skupinou. Umožnuje sdílení nebo ukládání dokumentů pod jednotlivý MS tým či mezi uživateli. Microsoft Teams spolupracuje i s dalšími nástroji od společnost Microsoft např: OneNote pro poznámky, Word, PowerPoint, Forms, Excel atd., které tuto aplikaci rozšiřují o spoustu funkcionalit.

Další výhodou jsou vznikající aplikace v samotných MS Teams, které dále posunují veškeré funkcionality v MS Teams a to bezplatně, což se bude v tomhle směru opírat tato práce.

Během covidové pandemie se tato aplikace stala velmi populární jak u firem, tak i ve školství po celém světě. Ve školství se tato aplikace prosadila nejen díky své jednoduchosti, ale i vysoké efektivnosti co se týká nejen správy jednotlivých týmů, tak i rozvržení oprávnění uživatelů (student x učitel) i komunikace s jinými aplikacemi od Microsoft, jako je zadávání úkolů nebo testů pomocí Microsoft Forms nebo ukládání dat pomocí Onedrive. MS Teams umožnil učitelům a studentům pokračovat vzdělávání, aniž by museli opustit své domovy. Díky tomu se podařilo zajistit plynulost vzdělávání a udržet studenty aktivně zapojené do výuky i přes distanční výuku.

3.2 Blazor

Veškeré informace v této kapitole pochází od zdroje [2].

Framework, který umožnuje vytvářet webové aplikace pomocí, HTML, CSS a C# místo Javascriptu za pomocí .NET. Díky tomu můžeme dále přistupovat ke knihovnám .NET, které budu v tomto projektu potřebovat.

Blazor je založen na komponentách, což je vlastní prvek uživatelského rozhraní, který obsahuje jak HTML a CSS tak i c# kód. Můžou přijímat parametry, vnořovat se a opakovaně používat. Komponenty pak mají zkratku razor.

Blazor funguje na architektuře MVC (Model, View, Controller). Model poskytuje metody pro přístup k datům a jejich aktualizaci. Neví o existenci dalších komponent (View, Controller), což znamená, že výsledná data nikam neposílá. View (pohled) zobrazuje data zpracovaná modelem a uživatelským rozhraním. Určuje, jak mají data být prezentovaná v prohlížeči (to, co vidí uživatel). Controller (řadič) řídí celou architekturu. Přijímá vstupní data, které pak přiřadí modelu, od modelu následně získá data, která předá View. [3]

Blazor se dělí podle způsobu vývoje, a to na Serverový a WebAssembly.

Serverový Blazor, hostuje aplikace pomocí serveru ASP.NET Core. Aktualizace uživatelského rozhraní pak probíhají přes SignalR. Což znamená, že kdykoliv klient spustí událost, zavolá server pomocí SignalR. V praxi to znamená, že prvotní načtení aplikace je rychlejší než u WebAssembly, protože sestavení a zkompilování probíhá na serveru, ale uživatelské požadavky mívají delší odezvu oproti WebAssembly, protože veškerá komunikace se odesílá na server a pak zpátky do prohlížeče.

Blazor WebAssembly hostuje aplikace přímo u klienta (SPA), což znamená, že veškerá práce se přesune ze serveru na klienta, respektive na prohlížeč. V praxi to pak znamená, že prvotní načítání aplikace je mnohem pomalejší než u serveru, protože veškeré sestavení a zkompilování se stahují v prohlížeči. Oproti serveru, pak uživatelské požadavky probíhají rychleji díky tomu, že všechna komunikace probíhá mezi prohlížečem a sestavení .NET.

Javascript knihovny či rozhraní API můžu volat díky JavaScript interop, který umožnuje volat Javascript kód pomocí c# i zpětně. Díky tomu nejsme omezeni pouze na knihovny .NET.

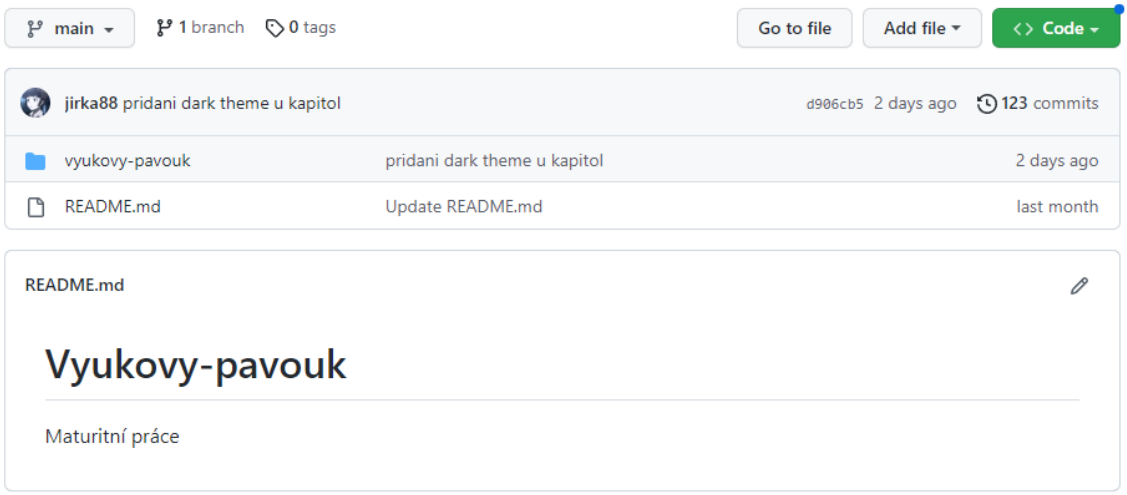
3.2.1 Důvod zvolení pro vývoj

Vývoj MS Teams aplikace v Blazoru byl zvolen především, kvůli větší znalosti c# a ASP.NET či práci dále v Entity Framework. Další důvod byl, že vývoj MS Teams aplikace v prostředí Blazoru je oproti jiným prostředím nejvíce propracované a obsahuje mnoho dokumentací či tutoriálů, což by samotný vývoj mělo ulehčit.

3.3 Github/Git

Github je webová cloudová služba, která slouží k ukládání a spravování kódu. Zároveň funguje i jako sociální síť pro programátory. Jednotlivé projekty v githubu jsou uloženy do takzvaných repositářů, který spravuje buďto vývojář či vývojářský tým. Další alternativou služby je například Gitlab či BitBucker. [4]

Git je open source systém pro správu verzí (sledování a řízení změn v kódu). Pracuje se s ním pomocí příkazové řádky či prostřednictvím IDE. Pomocí Gitu se může projekt zálohovat či vrátit do určitého stavu (commitu). [4]



Obrázek 3 Repositář k projektu [1]

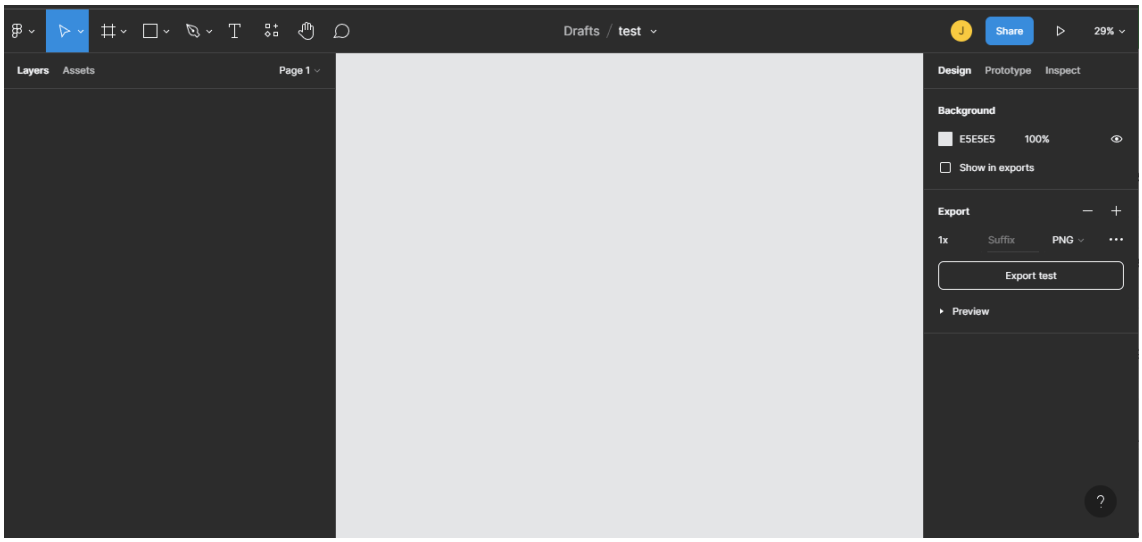
3.4 Zpracování frontendu

V této části se rozebere, co bylo použito pro návrh a vývoj frontendu v aplikaci pro MS Teams.

Frontend označuje část webové stránky nebo aplikace, kterou uživatel vidí a s čím interaguje. Je tedy zodpovědný za vykreslování uživatelského rozhraní a zpracování interakcí uživatele. Obvykle je tvořen pomocí HTML, CSS či JavaScriptu. Může se dále skrývat i pod názvem GUI.

3.4.1 Figma

Jedná se o cloud-based kolaborační nástroj pro design. Slouží především k vytváření designu pro weby či webové aplikace, a to jak na desktop, tablet či mobil, takže díky tomu můžeme navrhnout plně responzivní web či aplikaci. Využití má jak pro UX designera, copywriter, ale i pro projekt managera nebo kodéra, který si např: může exportovat jednotlivé styly. Její výhoda spočívá i v tom, že je v real-time, což znamená v praxi, že na daném návrhu může pracovat více lidí a všichni uvidí aktuální stav dané práce. [5]



Obrázek 4 Náhled na prostředí ve Figmě

3.4.2 Pluginy ve Figmě

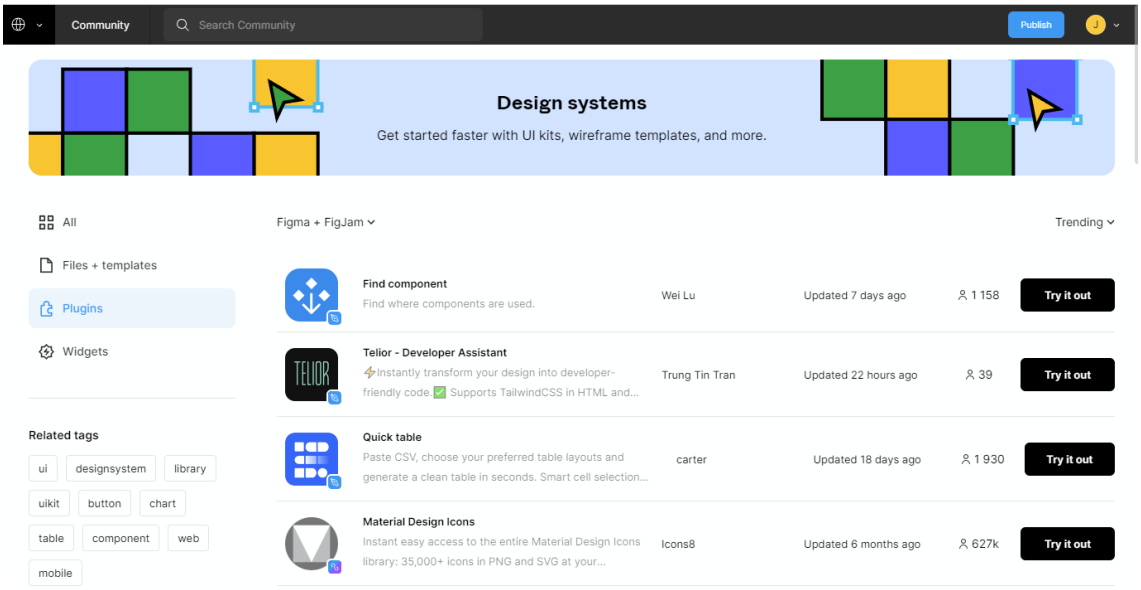
Veškeré informace v této kapitole pochází od zdroje [6].

Jedná o komunitní pluginy, které obsahují mnoho užitečných nástrojů jak pro jednotlivé designy různých komponent či nástroje pro usnadnění práce. Pluginy celkově rozšiřují funkcionality ve Figmě. Můžou se rozdělit do třech hlavních skupin, a to na kontent, design a Workflow pluginy.

Kontent pluginy pomáhají především v práci s obrázky, ikonami či grafy nebo automatickým doplňování textů (Lorem Ipsum).

Design pluginy podle samotného názvu rozšiřuje práci s designem, a to jak u jednotlivých komponent, tak i u celé šablony. Dále se může jednat i o práci s fontem až po 3D render.

Workflow pluginy slouží pro usnadnění práci v týmu, může se jednat o jednotlivé chaty mezi uživateli či přidání statusu progresu v daném projektu a mnoho dalších služeb.



Obrázek 5 Zobrazení komunitních pluginů [2]

3.4.3 Fluent UI

Jedná se o UX Framework obsahující webové komponenty v HTML a CSS od společnosti Microsoft. Díky těmto webovým komponentám se docílí sladění uživatelského rozhraní, které využívají Microsoft služby, hlavně pak Microsoft Teams (např: podobná tlačítka, výběry či dosáhnutí barevného sladění) a tím pádem ulehčuje samotnou práci s Frontendem. Další výhodou je, že obsahuje plugin ve figmě (Microsoft Teams UI kit), kde se můžou jednotlivé komponenty, které obsahuje použít k jednotlivým návrhům. Podporuje i mnoho Frameworků jako Blazor, React, Vue či .NET. [7]

Fluent UI React využívají produkty od společnosti Microsoft např: Office 365, Outlook, Onedrive, SharePoint, Azure DevOps atd. [8]

V MS Teams app v Blazoru je při vytvoření přímo integrovaná.

3.4.4 Bootstrap

Jedná se o bezplatný frontend framework, který napomáhá při vytvoření responzivního webu či webové aplikace. Jedná se o soubor nástrojů HTML, CSS a JavaScript, které usnadňují vytváření responzivních a uživatelsky přívětivých webových stránek. Javascript pak využívá u interaktivních prvků. Díky bootstrapu se nemusí vytvářet nové CSS styly či JavaScript kód, což umožnuje rychlý vývoj responzivního webu. [9]

3.5 Zpracování backendu

V této části se rozeberou prostředky, které byly použity při vývoji backendu. Jedná se především o použité API či prostředek pro zpracování dat z databáze.

Backend se jinak nazývaná strana serveru, která označuje část webové stránky či aplikace, která je zodpovědná za zpracování a ukládání dat, celkově se stará o logiku aplikace. Backend komunikuje s frontendem a tím podporuje funkčnost uživatelského rozhraní.

3.5.1 MS Graph API

Jedná se o RESTful API od společnosti Microsoft, které umožňuje přístup k datům a přehledům v Microsoft 365 v reálném čase. Umožnuje nezávisle zjišťovat různé informace, jako například o Onedrive, OneNote, ShareList, Outlook, Excelu nebo uživatelích v organizaci. Dále umožnuje získání informací o MS Teams, kde může zjistit např: základní informace o uživateli (jméno, příjmení, funkce...) nebo informace o MS týmu. Dále dokáže i vytvořit nový kanál nebo tým v Microsoft Teams. MS Graph podporuje nejen c#, ale i JavaScript, Powershell či Javu. [10]

V tomto případě se bude používat k autentizování uživatelů, a informace získané pomocí MS Graph API se použijí k tomu, aby aplikace mohla pracovat s uživateli a udělovat jim oprávnění nebo ukládat do DB.

3.5.2 Entity Framework Core

Veškeré informace v této kapitole pochází ze zdroje [11].

Jedná se o open source multiplatformní technologii, která umožnuje přístup k datům z databáze. Jedná se o operace CRUD (Create, Read, Update, Delete). Umožnuje vytvářet pomocí tříd jednotlivé entity (modely) a definovat mezi nimi vztahy. Ty se pak pomocí DbContext a migrací převedou do databáze (Code first). Druhý způsob se nazývá Db-first, který z existující databáze převede jednotlivé tabulky a jejich konfigurace a relace na modely a současně vytvoří i DbContext. Dále umožnuje dotazovat se do databáze pomocí LINQ dotazů, které se pak při použití převedou na SQL dotazy, které putují přímo do zvolené databáze.

Migrace EF Core je způsob, kde jednotlivé modely dat a jejich nadefinované vztahy jsou převedeny do migrace, které pak pomocí sady příkazů v Manager konzoli či v rozhraní příkazového řádku .NET, vytvoří či aktualizuje dané entity a vztahy v námi určené databázi. Migrace v EF Core jsou velmi užitečné pro údržbu databáze, při vývoji aplikace a umožňují snadno aktualizovat databázi při změnách v modelu dat.

Instance DbContext se stará o připojení a interakci s databázi, dotazování, sledování změn, mapování dat a ukládání entit do databáze. Umožnuje vkládat jednotlivé modely a jejich konfigurace či vztahy a následně je propojit s databázovou tabulkou. Pokud by jakýkoliv model nebyl v rámci DbContext zahrnut, nebude obsažen ani v migraci, a tedy se nevytvoří v databázi. Dokáže se dále dotazovat do databáze či ukládat už počáteční data. [12]

3.5.3 SessionStorage

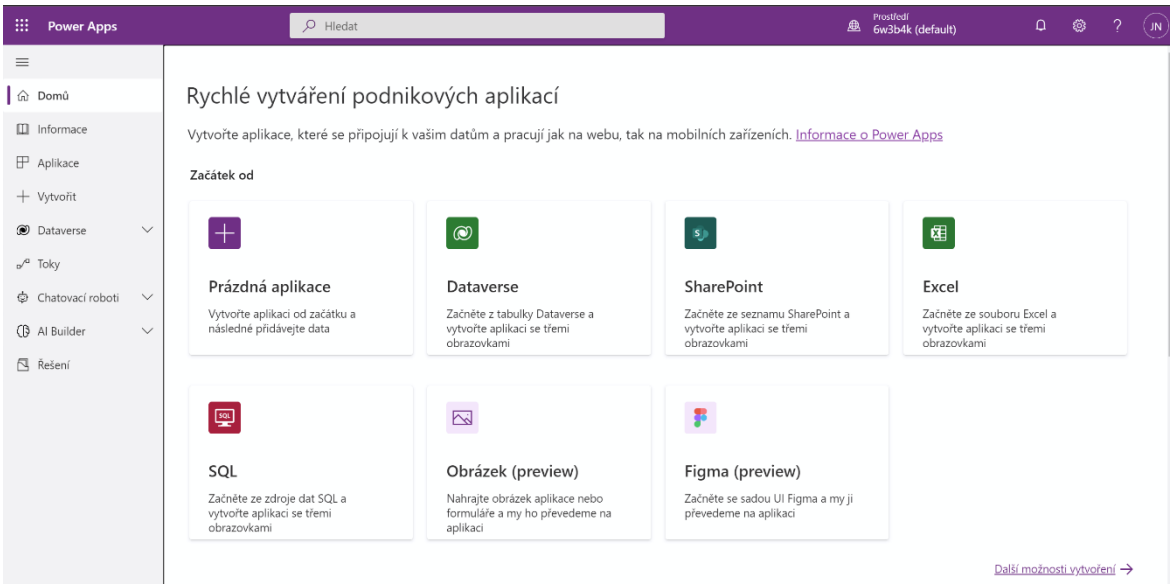
SessionStorage či česky úložiště relací, slouží pro ukládání dat na straně klienta, která jsou uložena v paměti prohlížeče a jsou přístupná pouze webové stránce či aplikaci, která je vytvořila. Tyto informace jsou dostupné pouze pomocí Javascript kódu, což umožnuje uživatelům si zobrazit obsah úložiště relací. Po uzavření webové stránky, aplikace nebo relace, jsou data uložená v relaci vymazána. Je svázaná s relací serveru, což znamená, že jsou k dispozici pouze pokud je stránka vyžádána na serveru, pokud běží lokálně není k dispozici. Obvykle se používá k ukládání dočasných dat, která jsou specifická pro jednoho uživatele a jednu relaci. [13]

4 DALŠÍ MOŽNOSTI VÝVOJE APLIKACE PRO MS TEAMS

Tato část se bude zabývat dalšími způsoby vývoje aplikací pro MS Teams.

4.1 Power Apps

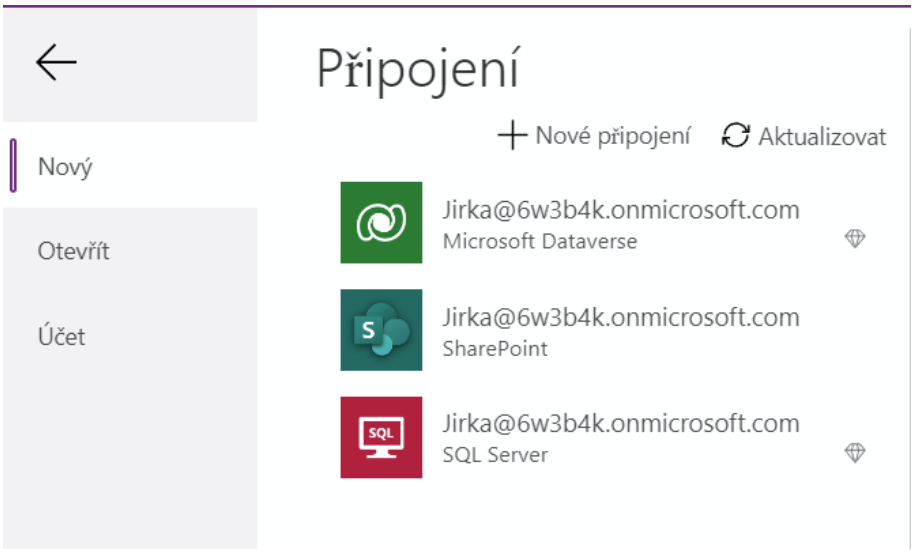
Jeden ze způsobů vývoje aplikací pro MS Teams se jedná o Power Apps, která patří do platformy Power Platform. Jedná se o sadu aplikací, služeb, konektorů a datových platform, které poskytují prostředí pro vývoj aplikace. Umožnuje vytvářet aplikace bez větších znalostí programování. Pomocí nich se může vyvíjet nejen aplikace, ale i chatovací roboty nebo AI buildery, které slouží pro automatizované procesy ve firmách. V neposlední řadě se může vyvíjet pomocí webové aplikace či desktopové aplikace, ale může se vyvíjet přímo v aplikaci pro MS Teams. [14]



Obrázek 6 Náhled na úvodní obrazovku v Power Apps ve webové aplikaci [3]

4.1.1 Přístup k datům

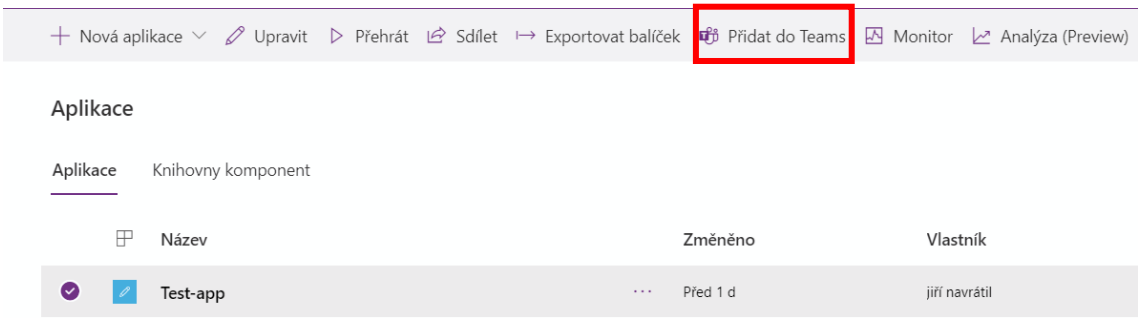
Nabízí snadné připojení k datům, uložených buďto v datové platformě dataverse nebo i k online nebo místním datům (Sharepoint, SQL, MySQL, Excel, Google Drive...). Pokud se nastaví tyto připojení, může se přistupovat k jednotlivým datům ve všech vytvořených aplikací. [14]



Obrázek 7 Ukázka připojení některých dat k Power Apps [3]

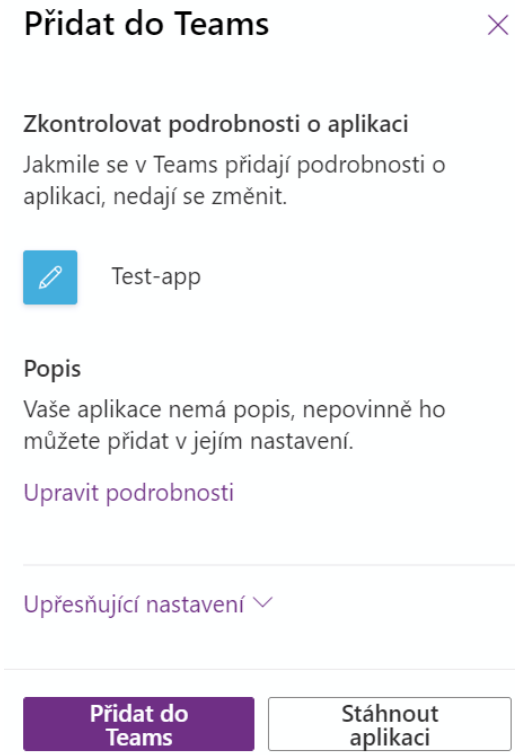
4.1.2 Nahrání do Teamsu

Aplikace se dají přímo sdílet či exportovat a poté nahrát do MS Teams, kde daná aplikace vytvoří kartu v MS týmu. Umožnuje i importovat cizí aplikace, které se můžou používat v našem prostředí. Toto jde pouze pokud se vyvíjí aplikace ve webové či desktopovém prostředím. Aplikace vyvíjené přímo v MS Teams lze přidat pouze v MS týmu, který se zvolil na začátku vývoje aplikace.

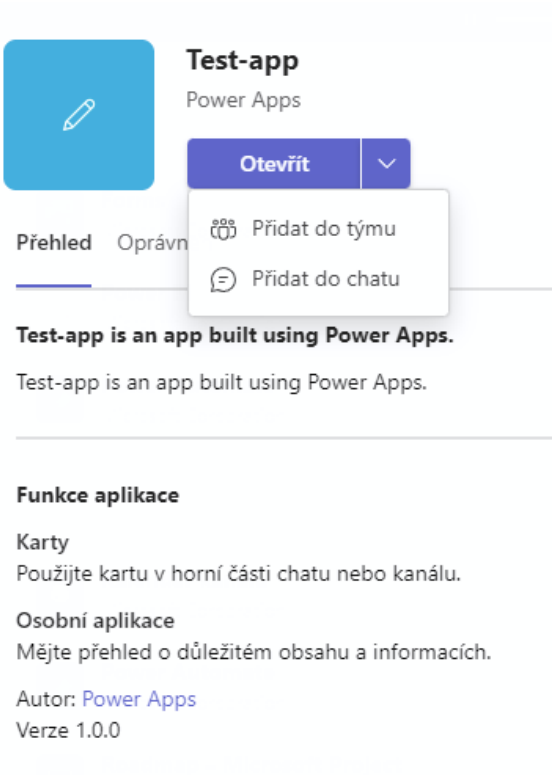


Obrázek 8 Vybraní vytvořené aplikace a ukázka přidání do Teams [3]

Po rozkliknutí se vybere, jestli se chce daná aplikace nahrát do MS Teams nebo stáhnout aplikaci v podobě zip. Pokud se vybere „nahrát do MS Teams“, přímo otevře MS Teams, kde se spustí dialogové okno. Aplikace se může přidat jako osobní aplikaci pouze pro uživatele nebo přidat do zvoleného MS týmu či chatu. Pokud se vybere možnost stáhnout aplikaci, stáhne se nám zip soubor s ikonkami a manifestem. Tento zip soubor se pak může nahrát do MS Teams pouze v naší organizaci.



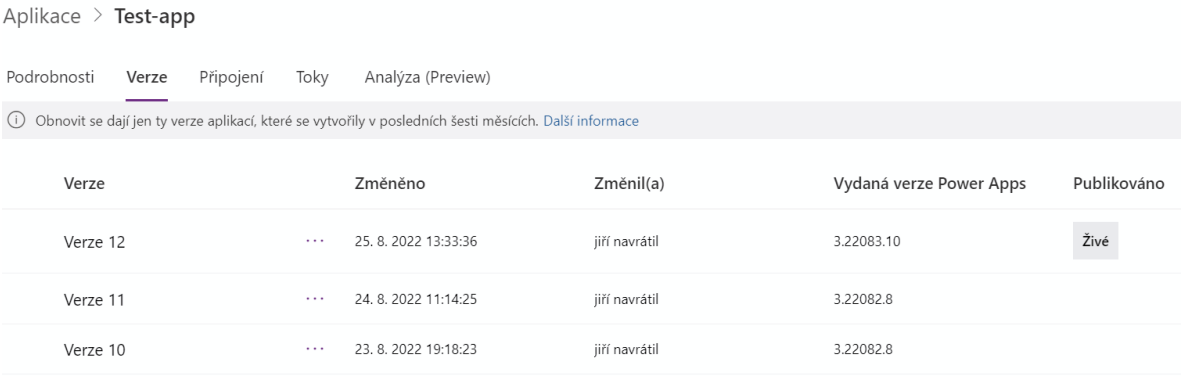
Obrázek 9 Způsoby nahrání aplikace[3]



Obrázek 10 Dialogové okno při nahrání aplikace v MS Teams

4.1.3 Verze aplikace

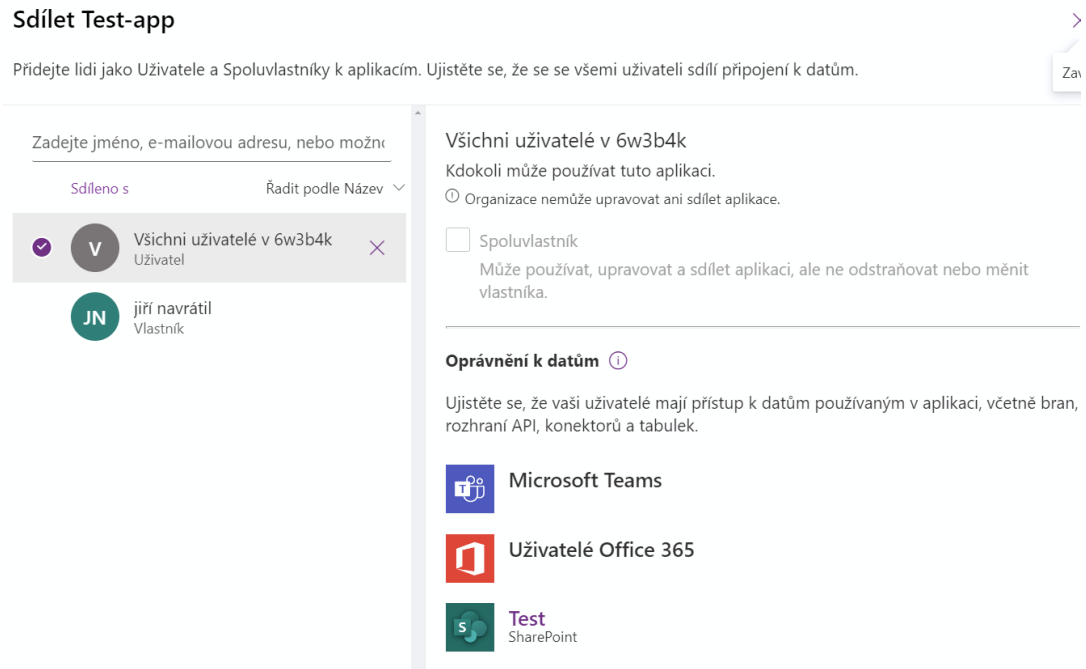
Pro každou aplikaci se podle ukládání aplikace, vytváří jednotlivé verze. Tyto verze jsou následně viditelné a lze je spravovat, což umožňuje snadné publikování konkrétní verze aplikace. Pokud je starší verze aplikace nahraná v MS Teams a publikuje se nová verze aplikace, automaticky aktualizuje aplikace na novější verzi i v MS Teams, kde aplikace běží. Pak přímo závisí na vývojáři, jakou verzi chce publikovat.



Obrázek 11 Ukázka publikace aplikace [3]

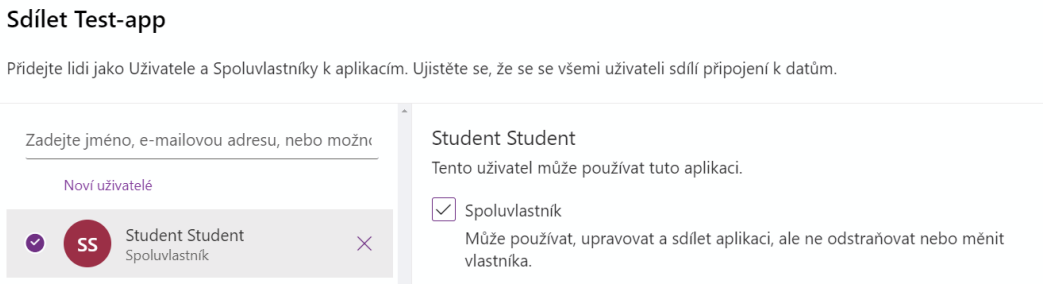
4.1.4 Přístup k aplikaci

Nesmí se zapomenout, že pro přístup je nutné k aplikaci povolit přístup daným uživatelům v organizaci. Pokud by uživatel nedostal přímo přístup do aplikace, nebudou mu umožněny žádné akce s aplikací, i když se tato aplikace nachází v MS týmu, do které uživatel patří.



Obrázek 12 Přidání oprávnění pro všechny uživatelé v organizaci [3]

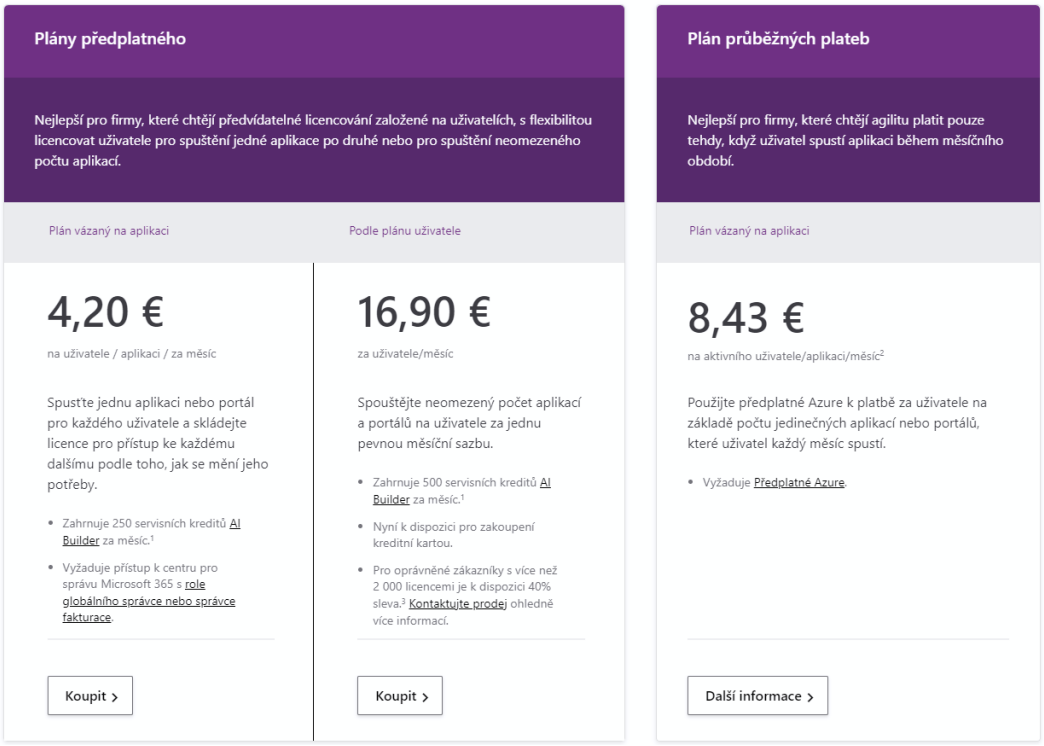
Pomocí těchto oprávnění můžeme nastavit, že daný uživatel má právo spoluvlastnictví aplikace. Díky těmto oprávněním může uživatel aplikaci jakkoliv upravovat a dále sdílet. Avšak aplikaci může najednou upravovat pouze jeden uživatel.



Obrázek 13 Přidání spoluvlastníka uživateli [3]

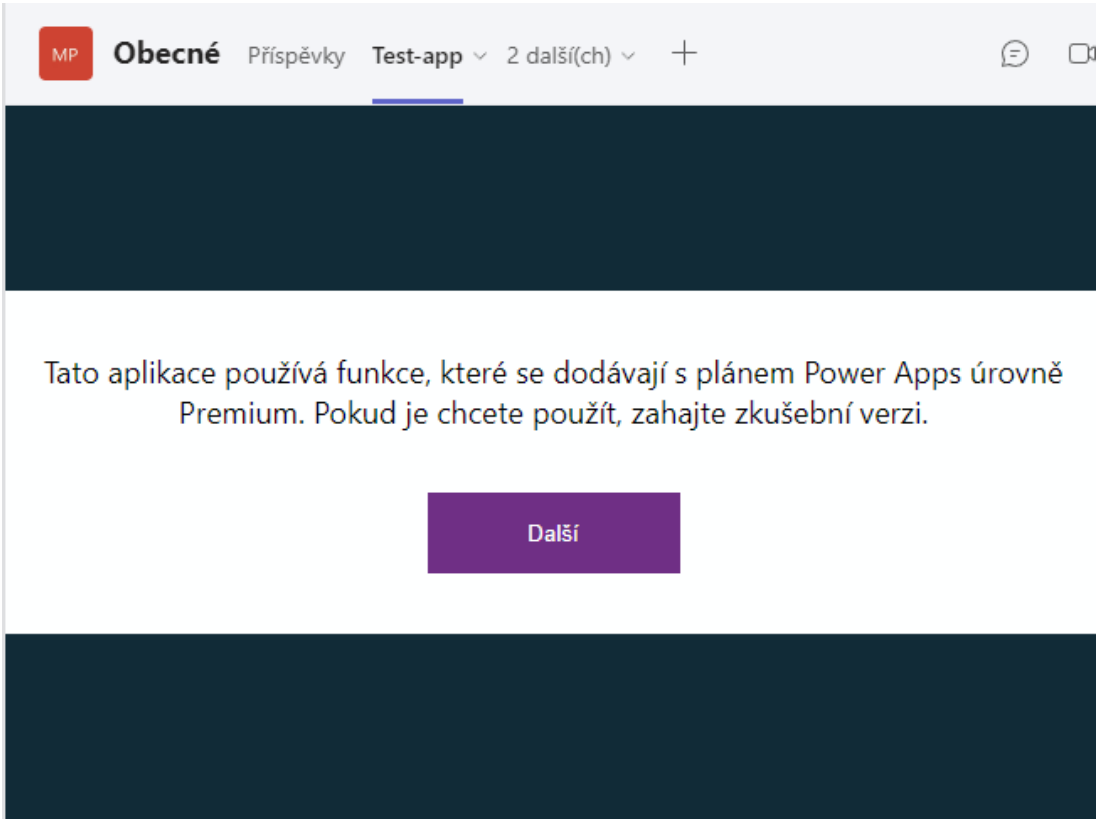
4.1.5 Důvod nevybrání pro vývoj

I přes to, že tato platforma umožnuje spoustu funkcionalit a ulehčení práce jako např: práci s databázemi či získání dat z MS Teams, kde je přímo určen speciální konektor, nebyl tento způsob vývoje vybrán, kvůli licencování této platformy. Ve standartní licenci Power Apps for office 365, která má i naše škola, je umožněno používat tento vývoj, ale pouze ze standartními konektory, což razantně omezuje možnost použít plno funkcionalit, které jsou potřeba v tomto projektu. Jedná se především o veškeré konektory k databázím, které by se nemohly použít. Jediná možnost by byla uložení dat pomocí Sharepoint listu nebo jiných podobných platforem, ale tyto systémy nejsou databázové systémy, což znamená, že nelze s nimi provádět komplexní práci s daty.



Obrázek 14 Licence k používání powers apps [4]

Problém v licencováním nastává i pro uživatelé, kteří nevyvíjí aplikace v Power Apps, ale pouze je spouštějí. Pokud aplikace využívá prémiové funkce, jako jsou například zmiňované konektory k databázi, uživatel by nemohl aplikaci spustit bez příslušné licence. Power apps nabízí spuštění trial verze k používání, ale tato licence trvá pouze 90 dní a po vypršení se přístup k aplikacím, kde jsou použity prémiové funkce, opět zablokuje.



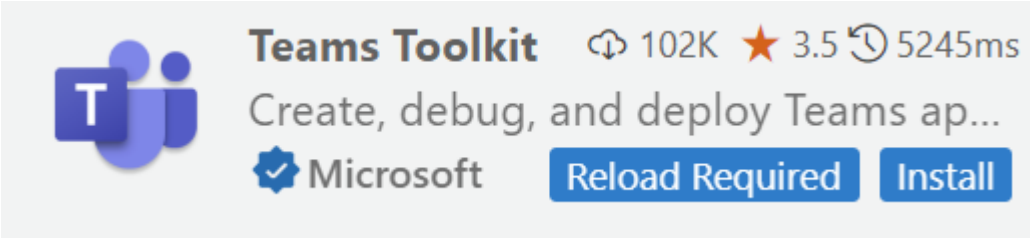
Obrázek 15 Spuštění aplikace s prémiovými funkcemi pouze s licencí Power Apps for Office 365

4.2 React

Je populární knihovna jazyka JavaScript pro vytváření uživatelských rozhraní, které využívají webové aplikace s menším a efektivnějším kódem, než kdyby se použil čistý JavaScript. Využívá opakovatelné komponenty, což usnadňuje vytváření rozsáhlých uživatelských rozhraní. Využívá architektury MVC (Model-View-Controller). Je snadno rozšiřitelný a podporuje různé nástroje a knihovny. Díky Reactu se může snadno měnit a aktualizovat data, aniž by se musela znova načítat stránka. [15]

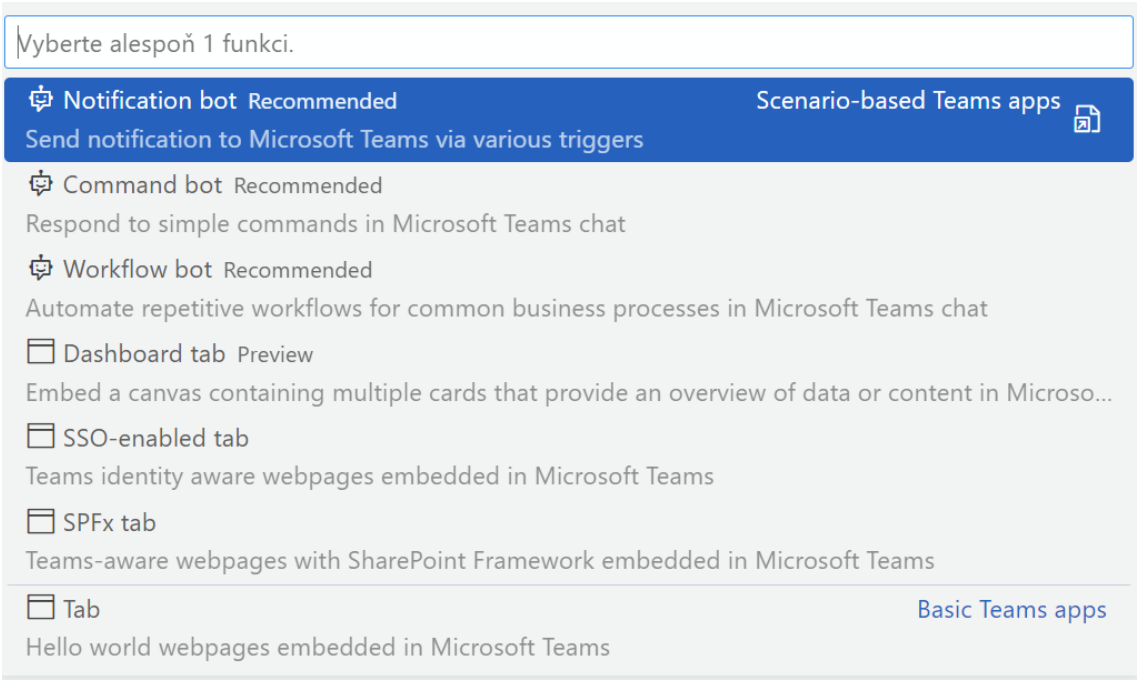
4.2.1 Vytvoření aplikace (tab)

Pokud se vybere vývoj pomocí Reactu, bude nejdříve potřeba nainstalovat službu node.js pro backendové prostředí JavasScriptu. Po instalaci node.js se otevře Visual Studio Code, kde je potřeba nainstalovat rozšíření Teams Toolkit.

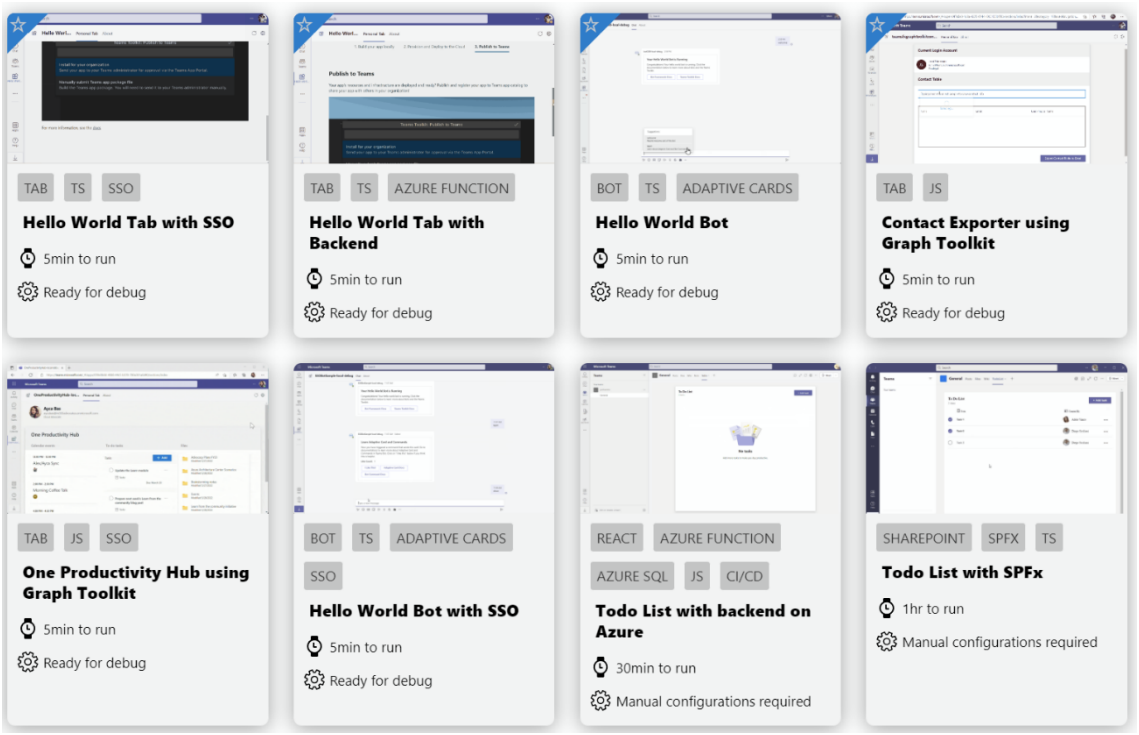


Obrázek 16 Teams Toolkit

Po instalaci rozšíření bude uživatel schopen vytvořit novou aplikaci pro MS Teams. Nejdříve se zobrazí výběr možností pro vývoj MS Teams aplikace, včetně typu aplikace (Bot, Tab…) a platformy pro vývoj. Uživatel si může také vybrat z předdefinovaných ukázek, kde je již k dispozici funkcionalita, což pomůže snadněji pochopit fungování dané věci.

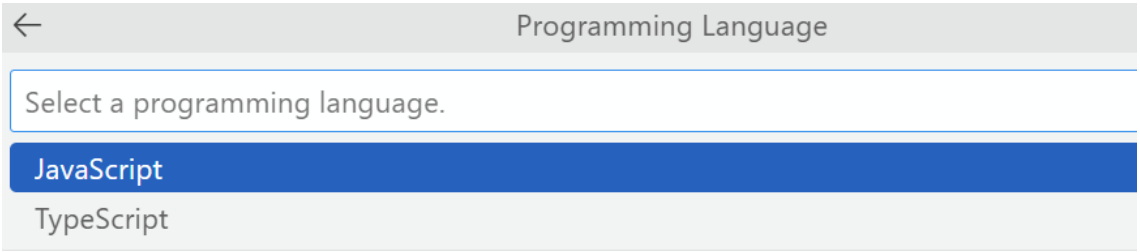


Obrázek 17 Výběr rozvrženích pro vývoj MS Teams app



Obrázek 18 Zobrazení ukázek

Po výběru tab, bude následovat možnost výběru způsobu vývoje, konkrétně v jakém programovacím jazyce bude aplikace vyvíjena.



Obrázek 19 Výběr programovacího jazyku

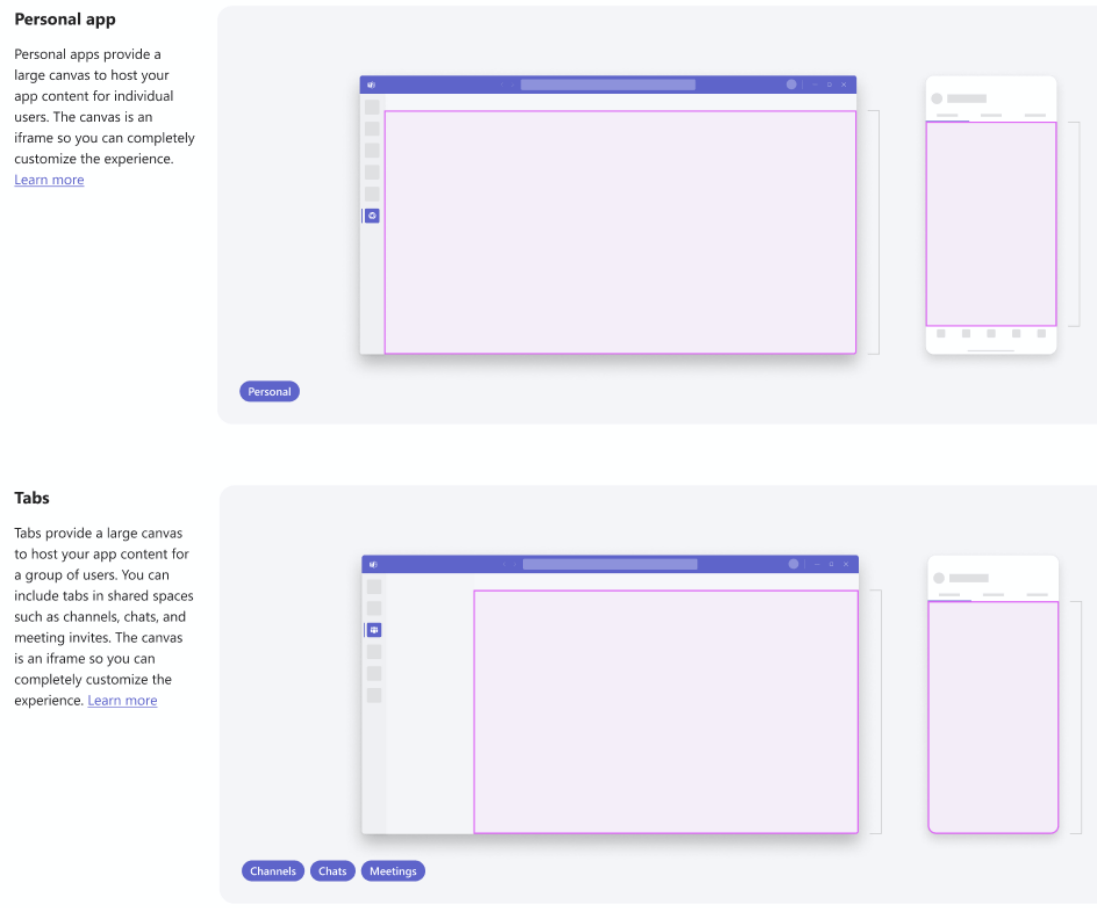
Po výběru se vytvoří aplikace. Pokud se chce aplikace spustit v MS Teams, musí se vybrat v jakém prohlížeči a zároveň ho propojit s tenant účtem. Poté se pod danou organizaci v MS Teams daná aplikace sestaví a spustí.

5 NÁVRH APLIKACE

V této části se rozebere, co bylo potřeba a zároveň jak byl zpracován návrh frontendu a databáze.

5.1 Frontend

Pro návrh frontendu celé aplikace byla zvolena webová aplikace Figma. Tato služba byla vybrána nejen kvůli své přehlednosti, ale také proto, že všechny své funkce nabízí zcela bezplatně. Dále obsahuje přímo plugin Microsoft Teams UI kit pro design uživatelského rozhraní, zejména pak komponent a šablon přesně pro návrh aplikací pro MS Teams. Tento plugin přesně slouží pro daný návrh při použití Fluent UI, což je knihovna přímo integrovaná v Microsoft Teams Apps v Blazeru.

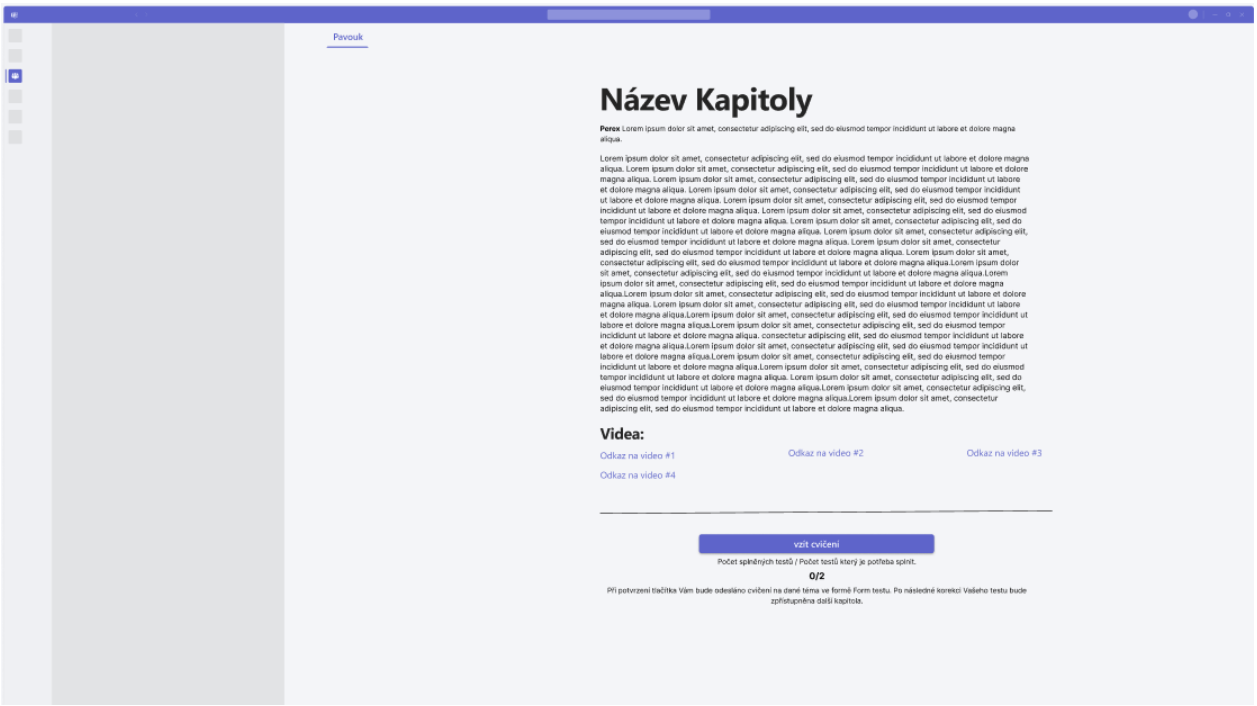


Obrázek 20 Dostupné šablony pro návrh aplikace v MS teams ve Figmě

5.1.1 Rozvržení

Jednotlivé návrhy bylo potřeba rozdělit do dvou skupin, a to pro zobrazení pro učitele a pro studenty, které se bude lišit podle přístupu k daným webům či komponent, např. student bude moci pouze zobrazit dané kapitoly, zatímco učitel bude moci kapitoly i přidávat či upravovat. Dále bylo potřeba navrhnout hlavní stránku se všemi dostupnými kapitolami či prvotním vytvoření struktury.

Obrázek 21 Návrhy zobrazení kapitol pro učitele

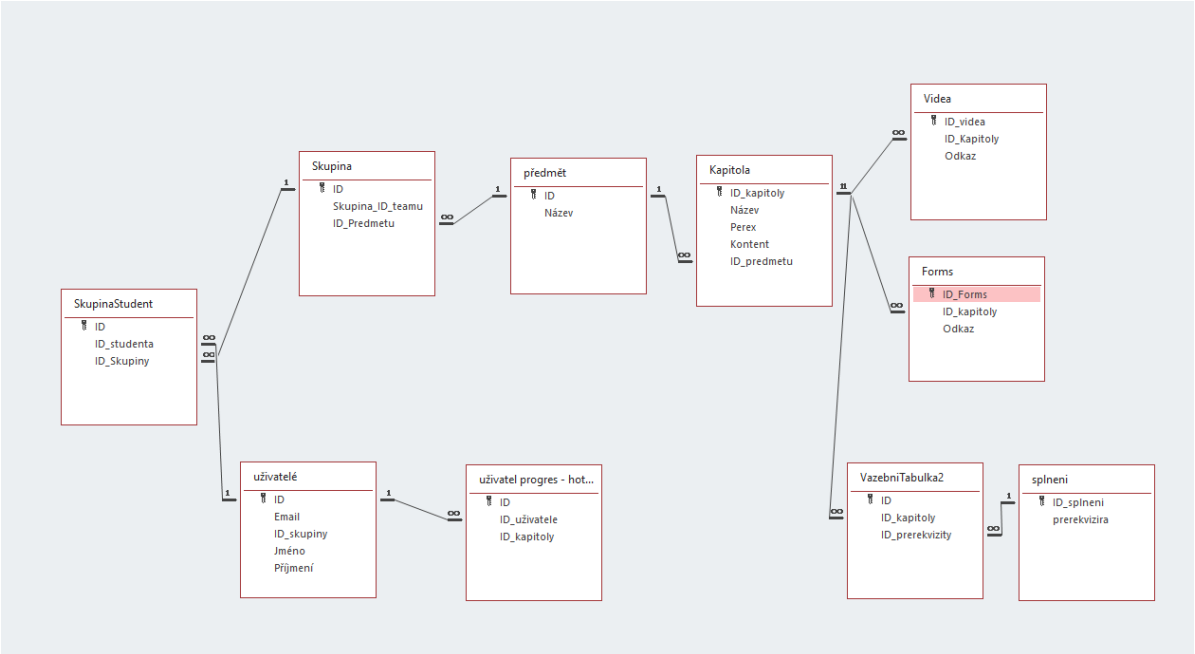


Obrázek 22 Návrh zobrazení kapitoly pro studenta

5.2 Databáze

Pro vytvoření prvotního návrhu databáze byla zvolena aplikace Microsoft Access, kvůli velice snadnému vytvoření jak samotných tabulek či jednotlivých vztahů tak i samotnému vložení testovacích dat, které se použily pro vytváření jednoduchých dotazů, které nasimulovali chování databáze.

Databáze začíná skupinou, pomocí které rozeznáme, pod jaký team v MS Teams patří daný předmět. Toto rozeznání bude fungovat díky jedinečnému ID Teamu v MS Teams, který se získá pomocí API a pak následně uloží do databáze při prvotním vytvoření skupiny. Dále každá struktura obsahuje jednotlivé kapitoly, které pak dále obsahují nejen samotný kontent, ale i odkazy na videa či prerekvizity, které jsou potřeba pro odemčení kapitoly. Nakonec máme jednotlivé uživatelé, kteří jsou v jednotlivých skupinách v MS Teams, které pak dále mají tabulku progres, která zaznamenává všechny jejich splněné kapitoly.

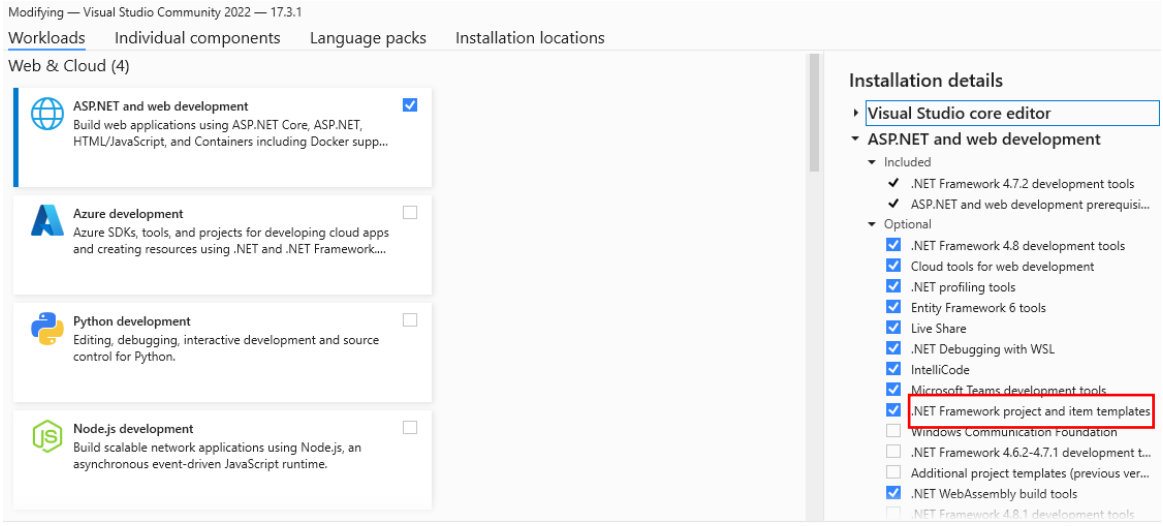


Obrázek 23 Prvotní návrh databáze v Microsoft Access

5.3 Prvotní instalace prostředí

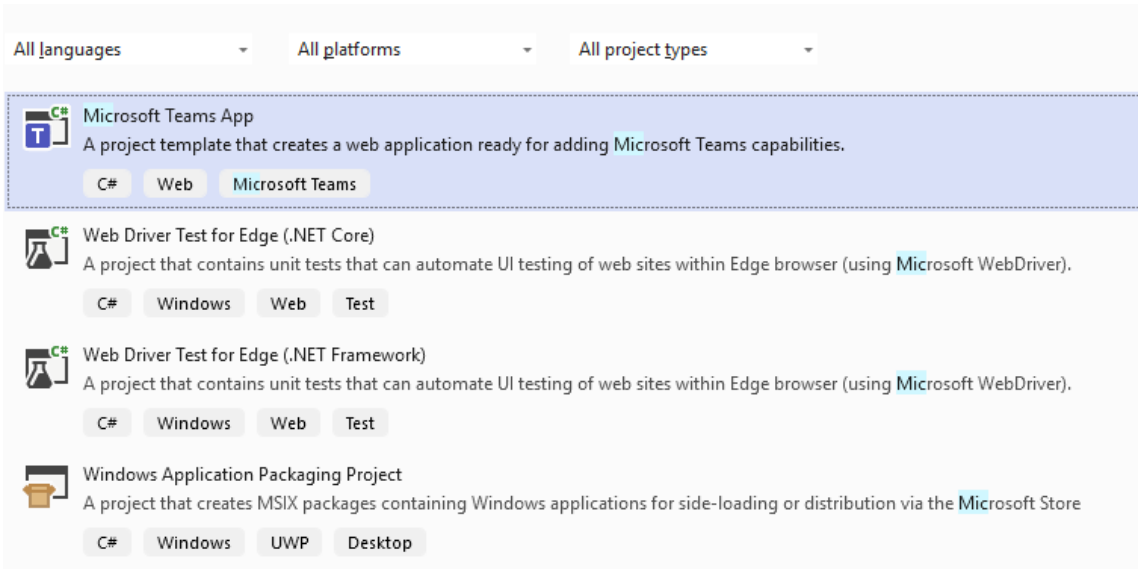
Na vývoj aplikace se bude používat Visual Studio, kde se musí doinstalovat balíčky Microsoft Teams development tools, které slouží pro vytváření a samotný vývoj aplikací pro MS Teams v Blazoru. Tento balíček vytvoří vše potřebné pro sestavení aplikace.

Nejdříve se spustí instalační soubor Visual Studio a po načtení se vybere verze Visual Studio a klikne se na „modifikování“. Zde se vybere vývoj v ASP.NET a v tom se vybere Microsoft Teams development tools.



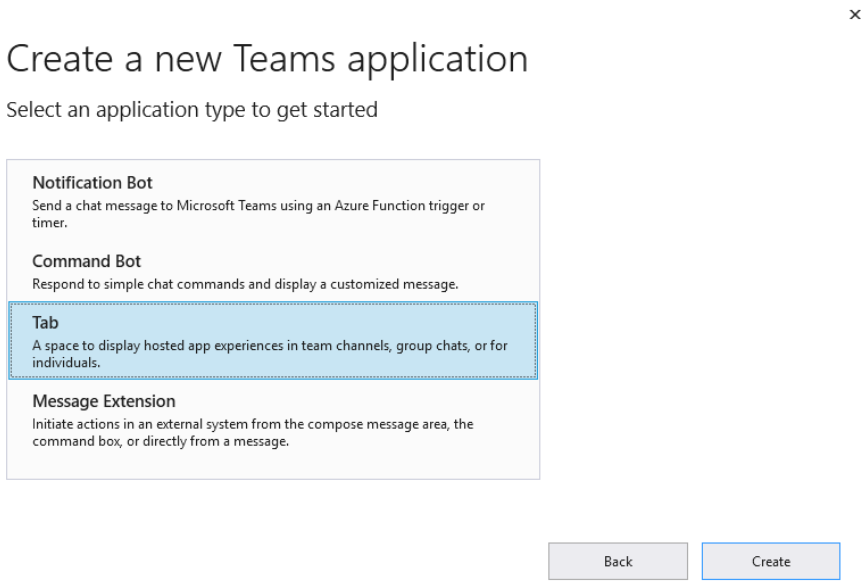
Obrázek 24 Instalování balíčku MS Teams development tools

Po instalaci se přejde k vytvoření projektu, kde se vybere šablona Microsoft Teams app.



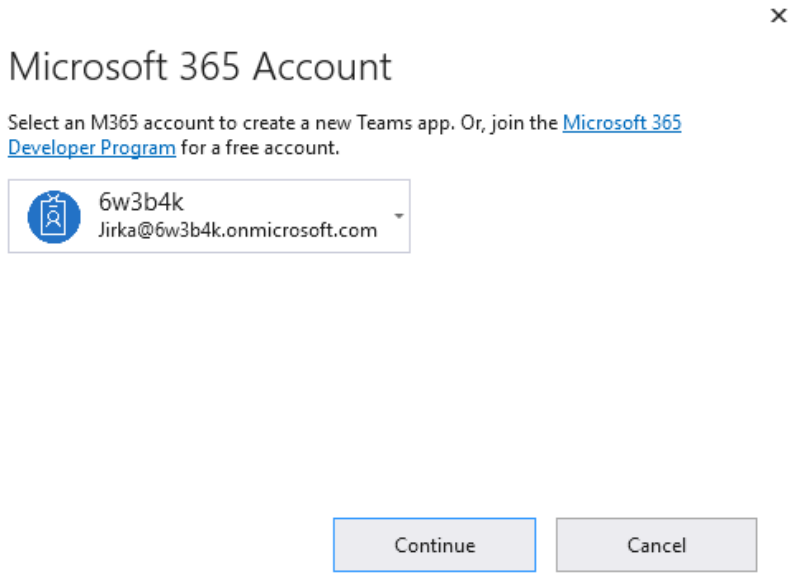
Obrázek 25 Vybrání šablony pro Microsoft Teams app vývoj

Po vybrání místa uložení projektu se přejde k výběru typu aplikace, který se bude vytvářet. V tomto případě se vybral typ Tab, který přímo vyhovuje požadavkům přesněji pro hostování aplikace sloužící jako kanál v MS týmu v MS Teams.



Obrázek 26 Výběr typ aplikace

Poslední věc po vytvoření aplikace je propojení s tenant účtem, pomocí kterého se bude aplikace spouštět v MS Teams. Musí se přejít do nastavení projektu, tam vybrat Teams Tool kit, kde se vybere Prepare Teams App Dependencies. Následně se přidá Tenant účet. Poté půjde spustit aplikaci v prostředí MS Teams v rámci organizace pod daným tenant účtem v prohlížeči dle výběru.



Obrázek 27 Vybrání Tenant účtu

6 KONCEPT APLIKACE

Aplikace je rozdělena podle toho, kdo k ní přistupuje (učitel nebo student). Pomocí tohoto rozdělení se musí zajistit, aby podle role měl logicky přístup k obsahu, ke kterému má mít.

6.1 Zobrazení učitel

Učitel v aplikaci má rozdělení na hlavní čtyři záložky, pomocí nich se odkazuje na různé stránky, kde může podnikat určité akce.

První záložka je seznam, kde se učiteli zobrazují kapitoly, které v dané struktuře vytvořil. Jednotlivé kapitoly si může otevřít a popřípadě upravit či vymazat.

Druhá záložka je přidání, která slouží k vytvoření kapitoly.

Třetí záložka je souhrn, která slouží pro zobrazení všech studentů ve skupině. Zobrazuje u studentů počet splněných kapitol a maximální počet kapitol. Tímto způsobem se dále vypočítá procentuální plnění studenta. Dále odkazuje na stránku progres, kde u každého studenta se můžou zobrazit jednotlivé splněné kapitoly. Dále se tu nachází i povolení, které slouží u studenta k zaznamenání, že odemčenou kapitolu splnil úspěšně či neúspěšně. Pokud ji splnil úspěšně může odemknout další kapitoly. Pokud neuspěl, musí zadání projít znovu. V neposlední řadě tu je i možnost smazání studenta ze skupiny, například v případě, když opustil školu.

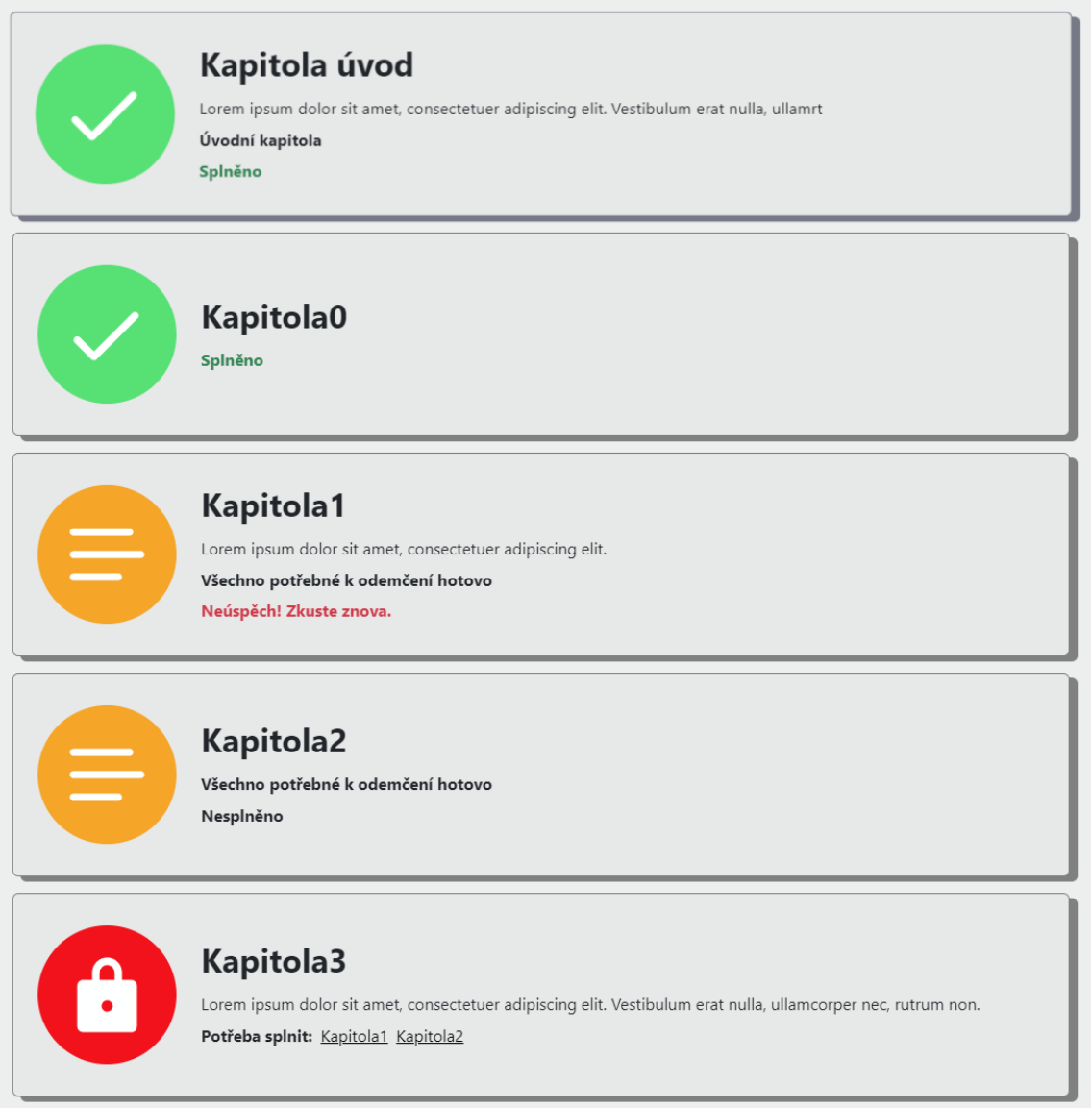
Obrázek 28 Souhrn u studenta

V čtvrté záložce je nastavení, kde je možné přejmenovat strukturu a resetovat skupinu, což vymaže všechna splnění studentů a jejich navázání na skupinu. Po resetování skupiny je možné v aplikaci pod MS týmem znovu vytvořit nebo použít existující strukturu (viz kapitola 8.4). Dále obsahuje i možnost smazání struktury. Poslední funkcionalitou je přepnutí viditelnosti struktury mezi privátní a veřejnou.

6.2 Zobrazení student

Student má přístup pouze k záložce seznam, kde se mu zobrazují všechny kapitoly, ale zobrazit obsah a plnit zadání, může u kapitol, ke kterým má přístup. Kapitoly se mu tedy dělí na tři skupiny podle splnění na splněné kapitoly, odemčené kapitoly, uzamčené

kapitoly. U splněné a odemčené kapitoly si může zobrazit obsah kapitoly. U odemčené má navíc přístup k zadání, které je potřeba vyplnit pro splnění kapitoly. U uzamknuté kapitoly si může zobrazit pouze nadpis a perex ke kapitole a informace, co potřebuje splnit (názvy kapitol) pro její odemčení.



Obrázek 29 Kapitoly z pohledu studenta

7 API VYTVOŘENÉ PRO APLIKACI

V této části se popíše, jak funguje vytvořené API, které komunikuje s DB.

Nejdříve je potřeba pomocí http požadavků (GET, POST, PUT, DELETE) vyvolat určitý kontrolér s určitou metodou. Co má kontrolér vyvolat pak pozná podle definované cesty s případným parametrem, který se společně posílá s požadavkem.

V tomto příkladu se zjišťuje aktuální skupina se strukturou, v které se nachází uživatel. V Obrázek 30 Vyvolání požadavku se odešle požadavek http GET, který slouží

```
group = await Http.GetFromJsonAsync<Skupina>
($"{MyNavigationManager.BaseUri}api/Skupina/"
+ (GroupToken.teamsContext.TeamId).ToString());
```

Obrázek 30 Vyvolání požadavku

k přijímání dat.

Http GET vyvolá v kontrolérů SkupinaController.cs metodu, která je pro GET požadavek a zároveň přijímá parametr. To vyvolá metodu „GetGroup“ třídy GroupManager, jehož instance se skrývá pod _IGroup.

GroupManager implementuje nadefinované metody z rozhraní IGroup. Dále obsahuje instanci DBContext, pomocí, které se v jednotlivých metodách přijímají, případně odesílají do DB pomocí LINQ dotazů.

```
public class SkupinaController : ControllerBase
{
    private readonly IGroup _IGroup;
    0 references
    public SkupinaController(IGroup iGroup)
    {
        _IGroup = iGroup;
    }
```

Obrázek 31 Deklarování _IGroup

```
[HttpGet("{IDTeam}")]
0 references
public async Task<Skupina> Get(string IDTeam)
{
    Skupina group = await _IGroup.GetGroup(IDTeam);
    return group;
}
```

Obrázek 32 Volání metody v kontroléru

```
4 references
public interface IGroup

    //vrátí skupinu s předmětem
    2 references
    public Task<Skupina> GetGroup (string IDTeamu);
    //uloží skupinu s předmětem do databáze
    2 references
    public Task AddGroup(Skupina skupina);
    //vytvoří uvodní prerekvizitu --> u splnění o hodnotě ID --> 0
    2 references
    public Task CreateIntroductionPrerequisite(Splneni splneni);
    //vymaže danou skupinu se strukturou
    2 references
    public Task DeleteGroup(int Id);
    2 references
    public Task ResetGroup(int Id);
    2 references
    public Task ConnectGroup(Skupina group);
}
```

Obrázek 34 IGroup rozhraní

Po vyvolání metody „GetGroup“ se získá určitá skupina se strukturou pomocí parametru, který se posílá. Získaná data z DB vrátí a uloží do modelu proměnné group, kde se s daty může následně pracovat.

```
public class GroupManager : IGroup
{
    readonly DBContext _dbContext;
    0 references
    public GroupManager(DBContext dbContext)
    {
        _dbContext = dbContext;
    }
    //připojí resetovaný předmět k nové skupině
    2 references
    public async Task ConnectGroup(Group group)...
    //vytvoří Teams skupinu pod existující předmět v databázi
    2 references
    public async Task AddGroup(Group group)...

    2 references
    public async Task CreateIntroductionPrerequisite(Completion splneni)...
    2 references
    public async Task<Group> GetGroup(string IDTeam)
    {
        Group group = await _dbContext.Group
            .Where(s => s.TmGroup == IDTeam)
            .Include(p => p.Subject)
            .FirstOrDefaultAsync();
        if(group == null)
        {
            group = new Group();
        }
        return group;
    }
```

Obrázek 33 GroupManager

8 Vývoj

V této části je rozebráno, co všechno bylo potřeba vyřešit při vývoji a jak byla zpracována aplikace a její jednotlivé části.

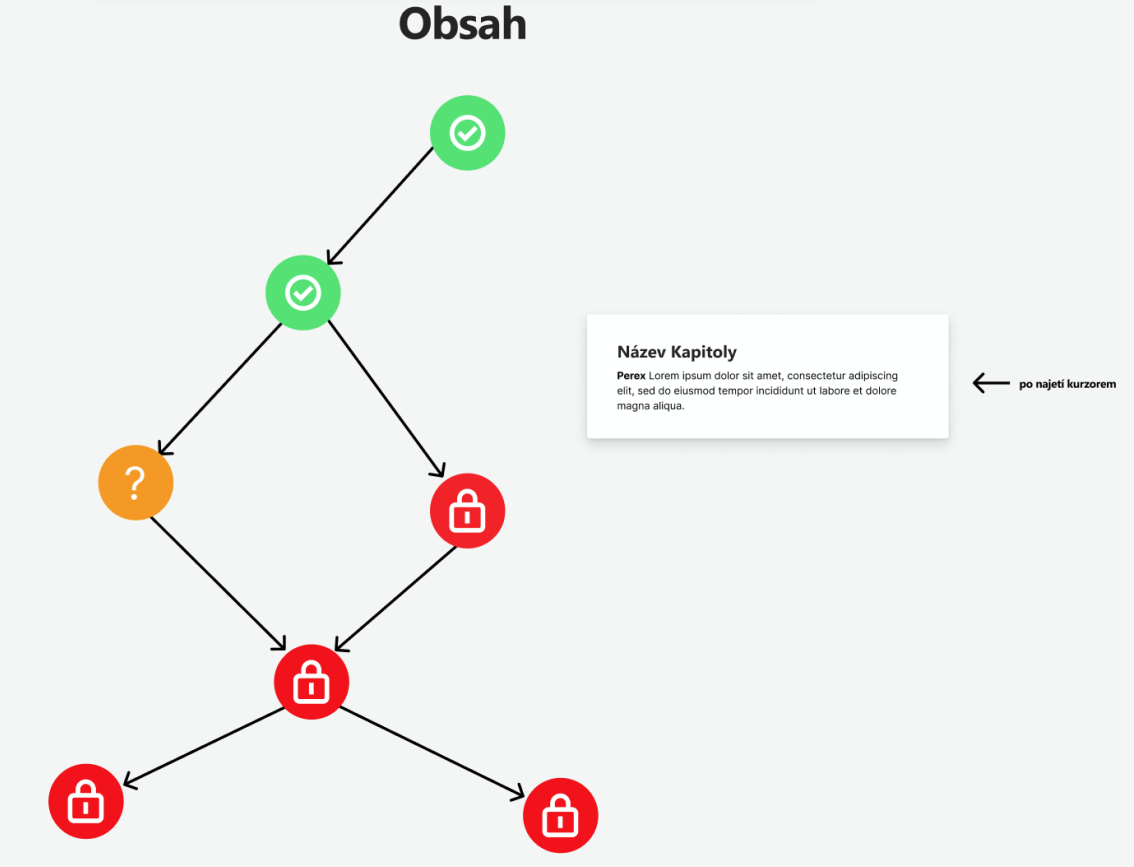
8.1 Frontend

K zpracování frontendu byly použity již zmiňované frameworky, a to bootstrap a Fluent UI viz kapitoly 3.4.3 a 3.4.4. Bootstrap byl použit k rozvržení a responzivitě webové aplikace a Fluent UI byl použit při tvorbě uživatelských prvků (tlačítka, checkbox atd.). Pomocí MS Graph API se zjistí, jaký režim (bílý, tmavý, kontrastní) používá uživatel v MS Teams. Díky tomu se frontend umí přizpůsobit i dalším režimům a nezůstává stále v defaultním bílém režimu.

```
if (GroupToken.teamsContext.Theme == "dark" || GroupToken.teamsContext.Theme == "contrast")
{
    dark = true;
```

Obrázek 35 Zjištění režimu

Největší problém u frontendu nastal při tvorbě hlavní stránky s kapitolami. Kapitoly v prvním plánu měly vypadat jako pavouk. Jednotlivé kapitoly by pak byly podle tohoto návrhu spolu vzájemně propojeny podle prerekvizit.



Obrázek 36 Prvotní návrh pavouka

Od tohoto nápadu se nakonec odešlo kvůli složitosti realizace. Složitost spočívala například ve vymyšlení logiky v propojení mezi jednotlivými kapitolami, ale zároveň i to, aby se jednotlivé kapitoly generovaly v úrovních podle prerekvizit. Nebylo vymyšleno ani responzivní chování, které by bylo potřeba zejména pak u přenosných zařízeních (tablet, telefon…).

Proto se přešlo k druhému nápadu, který byl pro zpracování mnohem snazší, a zároveň i uživatelsky přijatelnější co se týká poskytování údajů o kapitole. Zobrazení kapitol je v podobě seznamu, kde jsou kapitoly logicky uspořádané pod sebou.

Obrázek 37 Návrh v podobě listu

8.2 Připojení k databázi

Protože pro připojení k databázi používá aplikace Entity Framework Core, tuto knihovnu bylo potřeba nejprve do projektu nainstalovat.

Knihovny, které byly použity:

- Microsoft.EntityFrameworkCore [1]
- Microsoft.EntityFrameworkCore.SqlServer → Slouží přímo pro náš databázový systém. [2]
- Microsoft.EntityFrameworkCore.Tools → Obsahuje příkazy, které se zapisují do konzole Package Manageru. [3]

Po stažení patřičných knihoven se přešlo do souboru appsettings.json, kde se vytvořil připojovací řetězec k databázi.

```
"ConnectionStrings": {
  "DBConnection": "server=(localdb)\\MSSQLLocalDB;database=MSTeams;Trusted_Connection=true"
}
```

Obrázek 38 Připojovací řetězec do DB

Řetězec odkazuje v tomto případě na lokální databázový server, přímo do vytvořené databáze MSTeams. Trusted_Connection=true znamená, že uživatel je ověřován ze strany Windows.

Následně byl vytvořen soubor DbContext, kde se budou definovat jednotlivé modely a jejich konfigurace. Konfigurace bude načítaná následně pomocí metody OnModelCreating().

```
15 references
public class DBContext : DbContext
{
    0 references
    public DBContext(DbContextOptions<DBContext> options) : base(options)
    {

    }

    0 references
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {

    }
}
```

Obrázek 39 Vytvoření DbContext

Poté se tato služba zaregistrovala v souboru program.cs, která předá připojovací řetězec třídě DbContext. Metoda UseSqlServer konfiguruje kontext pro použití SQL databáze a jako parametr přebírá připojovací řetězec. [16]

```
builder.Services.AddDbContext<DBContext>(options =>
{
    options.UseSqlServer(builder.Configuration.GetConnectionString("DBConnection"));
});
```

Obrázek 40 Navázání připojovacího řetězce s DbContext v program.cs

8.2.1 Vytvoření modelů a vztahů (code-first)

Po připojení k databázi bylo potřeba vytvořit jednotlivé modely, z kterých pomocí migrace se převedou na tabulky v DB. To znamená, že bylo potřeba nadefinovat nejen jejich strukturu, primární klíče, cizí klíče, omezení atd., ale i jejich vztahy, které mají mezi sebou (1:1,1:M, M:N).

Nejdříve bylo potřeba vytvořit danou třídu, kde se nadefinuje daný model s jeho strukturou, ale i vlastnosti navigace.

Vlastnosti navigace jednoduše představují vztah k dané entitě. Existují dva typy vlastností navigací, a to referenční navigační vlastnost a vlastnost navigace kolekce. Referenční navigační vlastnost představuje vztah 1. Vlastnost navigace kolekce představuje vztah M. [17]

```
public class Chapter
{
    50 references
    public int Id { get; set; }
    34 references
    public string Name { get; set; }
    19 references
    public string Perex { get; set; }
    11 references
    public string Content { get; set; }
    9 references
    public int SubjectID { get; set; }
    1 reference
    public Subject Subject { get; set; }
    20 references
    public List<Assignment> Assignments { get; set; }
    22 references
    public List<Link> Links { get; set; }
    67 references
    public List<ChapterPrerequisite> ChapterPrerequisites { get; set; }
```

Obrázek 41 Model Kapitola

```
public Subject Subject { get; set; }
20 references
public List<Assignment> Assignments { get; set; }
22 references
public List<Link> Links { get; set; }
67 references
public List<ChapterPrerequisite> ChapterPrerequisites { get; set; }
```

Obrázek 42 Navigační vlastnosti v modelu kapitola

Po nadefinování struktury modelu je potřeba nakonfigurovaní vlastností např: entitní omezení a entitních konfigurací např: primární klíč, cizí klíč, vztahy atd. To vytvoříme za pomoci fluent API. Nejdříve se přejde k vytvoření třídy, která implementuje IEntityTypeConfiguration<Entita>. [18]

```
public class ChapterConfiguration : IEntityTypeConfiguration<Chapter>
{
    0 references
    public void Configure(EntityTypeBuilder<Chapter> builder)
    {
        builder.HasKey(i => i.Id);
        builder.Property(x => x.Id).ValueGeneratedOnAdd();
        builder.Property(n => n.Name).HasMaxLength(30).IsRequired();
        builder.Property(p => p.Perex).HasMaxLength(255);
        builder.Property(c => c.Content).IsRequired();

        // 1:M --> Predmet : kapitoly
        builder.HasOne(p => p.Subject)
            .WithMany(k => k.Chapters)
            .HasForeignKey(p => p.SubjectID)
            .OnDelete(DeleteBehavior.Cascade);
    }
}
```

Obrázek 43 Konfigurace kapitoly

Z Obrázek 43 je vidět, že pomocí metody Property nastavujeme chování vybraným vlastnostem. Pomocí této metody nastavujeme např: HasMaxLength, která definuje maximální délku textového řetězce. IsRequired nastavuje, aby daná vlastnost nesměla být prázdná. HasOne a WithMany nakonfiguruje vztah, který v tomto případě je, že jedna struktura má více kapitol. Poté pomocí HasForeignkey je nastaven cizí klíč na IdPredmetu, který má kapitola. OnDelete(DeleteBehavior.Cascade) nastavuje , že pokud by byla daná kapitola (hlavní entita) vymazána, budou vymazány i závislé entity (assignments, links, chapterPrerequisites). [19]

Po vytvoření a konfigurace modelu je nutné ho přidat v DbContext, aby se následně při migraci namapoval do databáze společně s jeho konfigurací a abychom s entitou mohli provádět operace CRUD (Create, Read, Update, Delete). Pro umožnění těchto operací a namapování na databázové tabulky je potřeba použít DbSet. [20][20]

```
public class DBContext : DbContext
{
    0 references
    public DBContext(DbContextOptions<DBContext> options) : base(options)
    {

    }
    8 references
    public DbSet<Chapter> Chapters { get; set; }
    5 references
    public DbSet<Subject> Subject { get; set; }
    1 reference
    public DbSet<Link> Link { get; set; }
    1 reference
    public DbSet<Assignment> Assignment { get; set; }
    5 references
    public DbSet<Group> Group { get; set; }
    5 references
    public DbSet<GroupStudent> GroupStudent { get; set; }
    3 references
    public DbSet<Student> Student { get; set; }
    7 references
    public DbSet<StudentCompletion> StudentCompletion { get; set; }
    12 references
    public DbSet<Completion> Completion { get; set; }
    4 references
    public DbSet<ChapterPrerequisite> ChapterPrerequisite { get; set; }
    8 references
    public DbSet<Prerequisite> Prerequisite { get; set; }
```

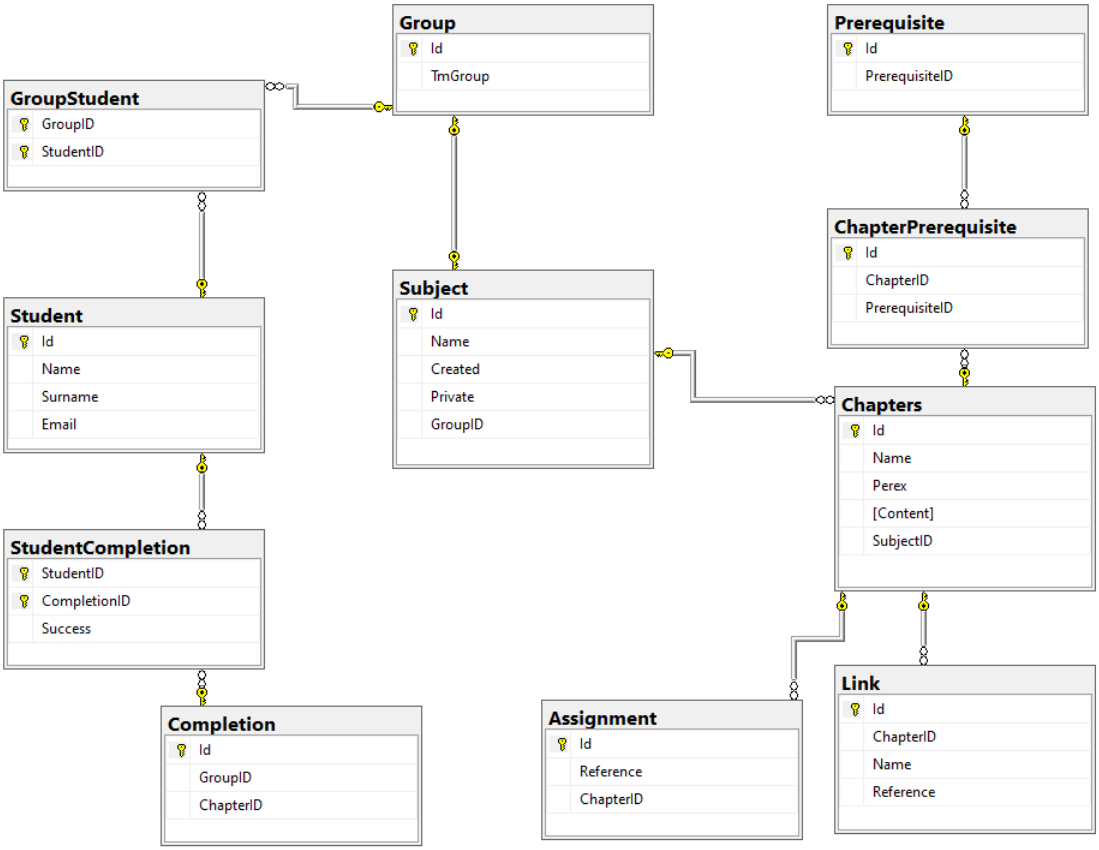
Obrázek 44 Přidání jednotlivých modelů do DBContext

Jednotlivé konfigurace entit se načítají v metodě OnModelCreating v DbContext, kde díky tomu, že konfigurace jsou v jednotlivých třídách, se načítají následně pomocí ApplyConfigurationsFromAssembly(Assembly.GetExecutingAssembly()). Konfigurace tato metoda rozpozná díky dědění z IEntityTypeConfiguration<>.

```
0 references
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.ApplyConfigurationsFromAssembly(Assembly.GetExecutingAssembly());
}
```

Obrázek 45 Načtení všech konfigurací

Po vytvoření všech příslušných modelů a jejich jednotlivých konfigurací nastala migrace, která převedla následně jednotlivé modely na DB tabulky na SQL serveru. Oproti prvotnímu návrhu (viz kapitola 5.2) se liší zásadně ve vztazích a přidaných atributech.



Obrázek 46 Výsledná databáze v SQL

Group uloží unikátní ID MS týmu při vytváření skupiny v aplikaci. Díky tomu v DB rozpoznáme, jací studenti a jaká struktura patří pod daný MS tým. Při vytváření skupiny se vytvoří či překopíruje struktura (subject), která obsahuje kapitoly. Přes skupinu v DB jsou propojeni studenti, kteří mají tabulku completion, které zaznamenává plnění kapitol v určitých MS týmech.

8.3 Získání a uložení identity uživatele, týmu

Nejdříve bylo potřeba zjistit identitu daného uživatele, který si spustí aplikaci v dané MS týmu. Proto, aby uživatel mohl procházet aplikaci bylo potřeba zjistit jméno, příjmení, email a jobtitle (např: žák, učitel) z MS Teams. Zjištění údajů probíhá na úvodní stránce, která po následném zjištění informací a uložení do sessionstorage a v DB (pokud je v aplikaci poprvé) mu následně zpřístupní podle informací aplikaci.

```
if (await HasPermission(_scope))
{
    await GroupToken.GetGroupAsync(MicrosoftTeams);
    //pokud o uživateli nic nevíme získáme o něm informace
    if ( UserToken.Profile == null || UserToken.Profile.DisplayName == null)
    {
        await ShowProfile();
        await UserToken.SetLocalUser(sessionStorage);
    }
```

Obrázek 47 Volání metody pro zjištění informací o uživateli

Nejdříve se zeptá pomocí metody „HasPermission“, kde se posílá parametr oprávnění v tomto případě user.read, zdali MS Graph API má oprávnění číst údaje o uživateli a skupině. User.read je nejnižší možné oprávnění.

```
private async Task<bool> HasPermission(string scope)
{
    try
    {
        await teamsUserCredential.
        GetTokenAsync(new TokenRequestContext(new string[] { _scope }),
        new System.Threading.CancellationToken());
        NeedConsent = false;
        return true;
    }
    catch (ExceptionWithCode e)
    {
        if (e.Code == ExceptionCode.UiRequiredError)
        {
            NeedConsent = true;
        }
        else
        {
            ErrorMessage = e.Message;
        }

    }
    IsLoaded = true;
    return false;
}
```

Obrázek 48 Zjištění, zdali MS Graph API má oprávnění

Na Obrázek 48 pomocí třídy TeamsUserCredential, která představuje aktuální identitu uživatele Teams, zjišťujeme token uživatele s oprávněním v _scope. Pokud se token získá, může projít. Pokud se nezíská token bude vráceno „false“ a tím pádem neprojde. Díky tomu se vykreslí stránka, kde hlavní administrátor musí odsouhlasit přístup MS Graph API k informacím o uživatelích v MS Teamu (stačí potvrdit pouze jednou, poté už není potřeba).

Pokud je token úspěšně získán je potřeba zjistit ID MS týmu, ve které se nachází. To se bude používat v další části kódu. Pomocí třídy GroupToken, se zavolá TeamSDK pro zjištění ID MS týmu. K volané metodě „GetGroupAsync“ se posílá parametr MicrosoftTeams, což je SDK, které se používá pro zjištění obecných informací o MS týmu či uživatelských nastavení atd…

```
public interface IGroupToken
{
    14 references
    TeamsContext teamsContext { get; set; }
    10 references
    Task GetGroupAsync(MicrosoftTeams microsoftTeams);

}
1 reference
public class GroupToken : IGroupToken
{
    14 references
    public TeamsContext teamsContext { get; set; }
    10 references
    public async Task GetGroupAsync(MicrosoftTeams microsoftTeams)
    {
        await microsoftTeams.InitializeAsync();
        teamsContext = await microsoftTeams.GetTeamsContextAsync();
    }
}
```

Obrázek 49 Zjištění ID MS týmu v třídě GroupToken

Při volání metody „GetGroupAsync“ se nejdříve musí inicializovat samotný TeamSDK. Po inicializaci se uloží do třídy TeamsContext informace o ID teamu. Třída TeamsContext slouží pro ukládání informací získaných z TeamSDK.

Po načtení ID MS týmu v Obrázek 49 se ptá, zdali daný uživatel nemá informace o sobě ze sessionstorage. Kontroluje se to protože, se může stát, že uživatel může načíst aplikaci znova a tím se opět dostat do úvodního ověřování nebo spustit aplikaci v jiném MS týmu. Protože uživatel bude mít stále uložené informace o sobě v sessionstorage, nemusí se znovu volat API pro zjištění informací o uživateli. Pokud je nemá, projde metodou ShowProfile().

```
private async Task ShowProfile()
{
    var msGraphAuthProvider =
    new MsGraphAuthProvider(teamsUserCredential);
    var graph =
    new GraphServiceClient(msGraphAuthProvider);
    UserToken.Profile =
    await graph.Me.Request()
    .Select("GivenName,displayName,surname,jobTitle,mail").GetAsync();
}
```

Obrázek 50 Zjištění informací o uživateli

Pomocí metody „ShowProfile“ lze získat informace o uživateli pomocí MS Graph API a získaného tokenu uloženého v teamsUserCredential. Přesněji se zjišťuje jméno, příjmení, JobTitle, mail. Po získání informací se uloží pomocí třídy UserToken do třídy User (třída User slouží, pro ukládání dat o uživateli získaného z MS Graph API). Volání tohoto API pro zjištění informací o uživatelích je velice náročný proces, proto se tento údaj uloží do zašifrované sessionstorage, aby se tento proces nemusel volat pokaždé co se bude procházet v aplikaci.

```
public interface IUserToken
{
    36 references
    User Profile { get; set; }

    10 references
    Task GetUserAsync(IIWSessionStorageService sessionStorage);

    2 references
    Task SetLocalUser(IIWSessionStorageService sessionStorage);
}
1 reference
public class UserToken : IUserToken
{
    private const string UserTokenAuth = "####";

    36 references
    public User Profile { get; set; } = new User();

    10 references
    public async Task GetUserAsync(IIWSessionStorageService sessionStorage)
    {
        string json = await sessionStorage.GetItemAsync<string>(UserTokenAuth);
        if (json != null)
        {
            Profile = JsonConvert.DeserializeObject<User>(json);
        }
        else {
            Profile = null;
        }

    }
    2 references
    public async Task SetLocalUser(IIWSessionStorageService sessionStorage)
    {
        string json = JsonConvert.SerializeObject(Profile);
        await sessionStorage.SetItemAsync<string>(UserTokenAuth, json);
    }
}
```

Obrázek 51 Třída UserToken

Protože už informace o uživateli jsou uložený v Profile, stačí teď uložit informace do sessionstorage. Protože sessionstorage neumí zobrazit objekt, je potřeba serializovat objekt user a uložit jej do stringu, který se poté uloží do sessionstorage pomocí metody „SetItemAsync“. V metodě se určuje název sessionstorage, který slouží k rozeznání jednotlivých sessionstorage. Pokud proběhne výše uvedený postup, jsou nyní dostupné

informace o uživateli (v zašifrované sessionstorage) a ID MS týmu, v němž se uživatel nachází.

Knihovny, které byly použity:

Newtonsoft.Json – Pro umožnění práci s JSON formátem např: serializace, deserializace objektu. [4]

Solutaris.InfoWARE.ProtectedBrowserStorage – Pro možnost použití zašifrované sessionstorage. [5]

8.4 Vytvoření struktury

Po získání a uložení informací do sessionstorage je potřeba vytvořit strukturu, pod kterou budou následně umístěny kapitoly. Nejdříve se zjistí, zdali v aplikaci v MS týmu se nenachází už struktura. To zjistíme podle API požadavku, kterým následně uloží získaná data do určeného modelu.

```
group = await Http.GetFromJsonAsync<Data.Group>
($"{MyNavigationManager.BaseUri}api/Skupina/" +
(GroupToken.teamsContext.TeamId).ToString());
if (group.TmGroup == null)
{
    subjects = await Http.GetFromJsonAsync
    <List<Subject>>($"{MyNavigationManager.BaseUri}api/predmet");
    IsLoaded = true;
}
```

Obrázek 52 Zjištění, zdali předmět ve skupině už existuje

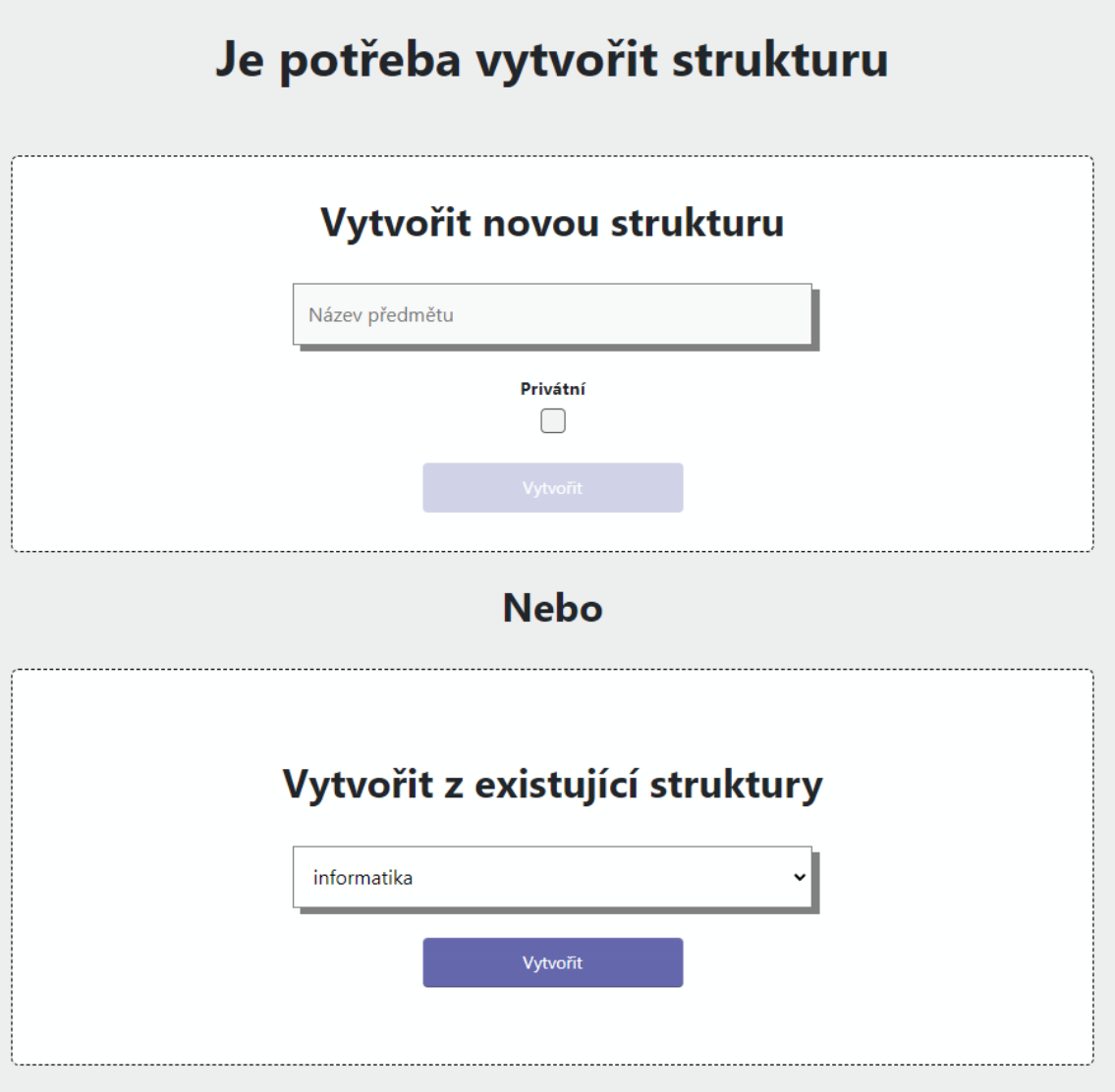
Po získání dat a uložení do modelu skupina v Obrázek 52 se kontroluje, zdali vlastnost TmSkupina není prázdná. Pokud ano, znamená to, že struktura v MS týmu nebyla vytvořena. To následně spustí další API, které získá všechny struktury, které byly vytvořeny a vykreslí stránku.

Poté se vykreslí určité komponenty v závislosti na roli uživatele (učitel nebo žák). U učitele se zobrazí komponenta, která umožní vytvořit nebo překopírovat strukturu. Tato komponenta přijímá seznam všech vytvořených předmětů, což učiteli umožní překopírovat existující strukturu z výběru.

Studentovi na rozdíl od učitele se vykreslí stránka, kde mu bude popsáno, že se momentálně čeká na vytvoření skupiny.

```
else if(IsLoaded)
{
    //pokud je učitel
    @if (UserToken.Profile.JobTitle == "Učitel" || UserToken.Profile.JobTitle == "Administrátor")
    {
        <Vytvoreni_Teamu subjects="@predmety" />
    }
    //pokud je student
    else
    {
        <Vytvoreni_teamu_student />
    }
}
```

Obrázek 54 Vykreslení stránky v úvodu



Obrázek 53 Učitel – Vytvoření struktury

Na Obrázek 53 je vidět, že učitel může vytvořit novou strukturu. Pokud chce, aby předmět nebyl vidět ostatními učiteli a mohl ho překopírovat či použít pouze on, zaškrtne volbu „privátní“. Po vytvoření nové struktury se naváže na daný MS tým.

Druhá možnost je vytvoření z existující struktury, kterou učitel či jiný učitelé v minulosti vytvořily. Učitel může vybrat z veřejných struktur nebo z vlastních privátních struktur. Po výběru struktury se vytvoří nová skupina a celá struktura s kapitolami, kterou obsahuje, se překopíruje do této skupiny. Tím pádem vznikne kopie už existující struktury, kterou učitel může upravit podle svých potřeb nezávisle na originální struktuře.

8.4.1 Obsah komponenty pro vytvoření struktury

Komponenta, kterou načte učitel, obsahuje dva editformy, které jsou propojené s určitým datovým modelem. Editform je komponenta, která je součástí blazor, která umožnuje vytvoření a zpracování formulářů a zajištuje také ověřování uživatelského vstupu. Blazor také obsahuje jednotlivé prvky formuláře, které umožňují interakci s daty v editformu.

```
<EditForm Model="@subject" OnValidSubmit="@Create" class="editForm d-flex justify-content-center">
     <FluentValidationValidator />
    <div class="card @(theme ? "darkTheme" : "")">
        <div class="card-body gap-4 @(theme ? "text-white darkInput" : "")">
            <h2 class="card-title text-center">Vytvořit novou strukturu</h2>
            <InputText id="nazev" type="text" placeholder="Název předmětu"
            @bind-Value="@subject.Name"></InputText>
            <ValidationMessage For="@(()=>subject.Name)" />
                <div class="d-flex flex-column align-items-center">
                <label for="private" class="fw-bold">Privátní</label>
                <FluentCheckbox id="private" @bind-Value="@subject.Private"></FluentCheckbox>
                </div>
            <fluent-button appearance="accent" type="submit"
            disabled="@(!context.IsModified() || !context.Validate())">Vytvořit</fluent-button>
        </div>
    </div>
</EditForm>
```

Obrázek 55 Formulář pro vytvoření nového předmětu

Na Obrázek 55 je formulář pro vytvoření nového předmětu. Formulář je navázán na model subject. Po odeslání formuláře vyvolá metodu, kde vyvolá POST požadavek a vytvoří novou strukturu. Nová struktura je zároveň navázaná na ID MS týmu.

FluentValidationValidator zajišťuje ověřování správnosti vstupních dat. V tomto konkrétním případě se kontroluje, aby InputText, do kterého uživatel zadává název předmětu nebyl prázdný nebo příliš dlouhý. Validace pak funguje díky FluentValidator, která musí být definovaný v daném modelu.

```
public class PredmetValidator : AbstractValidator<Predmet>
{
    1 reference
    public PredmetValidator()
    {
        RuleFor(n => n.Nazev).NotEmpty().WithMessage("Zadejte název předmětu!");
        RuleFor(n => n.Nazev).MaximumLength(30).WithMessage("Předmět má příliš velký název!");

    }
}
```

Obrázek 56 FluentValidator v modelu předmět

Funkce RuleFor nastavuje pravidla pro vlastnosti v datovém modelu. Funkce NotEmpty kontroluje, zda vstup není prázdný, a pokud ano, funkce WithMessage vypíše odpovídající chybovou zprávu. Funkce MaxLength nastavuje maximální délku vstupu a pokud je vstup delší, funkce WithMessage opět vypíše odpovídající chybovou zprávu. [21]

Pokud u názvu předmětu bude uživatel chtít odeslat prázdný nebo příliš dlouhý vstup, <ValidationMessage> zobrazí odpovídající chybovou zprávu definovanou funkcí WithMessage v rámci příslušné validace.

Po správném vyplnění formuláře na Obrázek 55 se spustí metoda „Create“, která model připraví a odešle ho do DB.

Knihovna, která byla použita:

- Blazored.FluentValidation [6]

```
public async Task Create()
{
    searchSubject = subjects.Find(x => x.Name.ToLower() == subject.Name.ToLower());
    if(searchSubject == null) {
        subject.Created = UserToken.Profile.Mail;
        subject.Group.TmGroup = (GroupToken.teamsContext.TeamId).ToString();
        //vytvoření nového předmětu se skupinou
        await Http.PostAsJsonAsync($"{MyNavigationManager.BaseUri}api/predmet", subject);
        //získání ID nově vytvořené skupiny
        Group Getskupina = await Http.GetFromJsonAsync<Group>
        ($"{MyNavigationManager.BaseUri}api/Skupina/" + subject.Group.TmGroup);
        //vytvoření úvodní prerekvizity navazané na skupinu --> 0
        await VytvorUvodniSplneni(Getskupina);
        MyNavigationManager.NavigateTo("/");
    }
    else {
        modelHidden = false;
    }
```

Obrázek 57 Vytvoření nového předmětu s navázáním na skupinu

V metodě „Create“ se nejprve kontroluje, zda se zadaný název struktury neshoduje už s již vytvořenou strukturou. Pokud ano, zobrazí se dialogové okno, které tuto skutečnost vypisuje. Pokud ne, nejprve se modelu nastaví, kdo vytvořil danou strukturu. Poté se naváže na strukturu skupina, které se předá ID MS týmu, ve které se struktura nachází.

Po vytvoření předmětu se pomocí požadavku GET získá z DB nově vytvořená skupina. Tento krok je nutný, protože je potřeba získat ID MS týmu, pod kterou se struktura nachází v DB. Pomocí získané informace umožní vytvoření úvodního splnění, které se naváže pod danou skupinu (viz kapitola 8.4.2).

Druhý formulář slouží k vytvoření skupiny s existující strukturou. To znamená, že se vytvoří skupina a zároveň se překopíruje existující struktura s jejími kapitolami. Zároveň

je potřeba zjistit všechny struktury, které jsou v DB uložené, aby se postupně načetly do výběru, kterým bude učitel schopen vybrat konkrétní strukturu, kterou chce použít.

```
@if (subjects.Count() != 0)
{
    if(!IfAllPrivate()) {
        <h2 class="py-4">Nebo</h2>
        <EditForm Model="@group" OnValidSubmit="@CreateIfExist" class="editForm d-flex justify-content-center">
            <div class="card">
                <div class="card-body gap-4">
                    <h2 class="card-title pt-2 text-center">Vytvořit z existující struktury</h2>
                    <select @bind-Value="@subjectSelect" @bind-Value:event="onchange">
                        @foreach (var item in subjects)
                        {
                            if (item.Privatni == false)
                            {
                                <option>@item.Nazev</option>
                            }
                            else
                            {
                                if (item.Vytvoril == UserToken.Profile.Mail)
                                {
                                    <option>@item.Nazev</option>
                                }

                            }
                        }
                    </select>
                    <fluent-button appearance="accent" Disabled="@offBtn" type="submit">Vytvořit</fluent-button>
                </div>
            </div>
        </EditForm>
    }
```

Obrázek 58 Formulář pro použití existujícího předmětu

Nejdříve se musí provést kontrola, zda existují struktury. Pokud ano, formulář se může vykreslit. Pokud ne, formulář se nezobrazí.

```
private bool IfAllPrivate() {
    if(subjects.Where(x => x.Private == true
        && x.Created != UserToken.Profile.Mail).Count() != subjects.Count()) {
        return false;
    }
    return true;
}
```

Obrázek 59 Podmínka pro vykreslení druhého formuláře

Pokud projde první podmínkou, je nutné ověřit i další podmínku. Ta se ptá, zda jsou všechny předměty v aplikaci privátní a zároveň, zda daný učitel nevytvořil žádný předmět. Pokud platí, že všechny předměty jsou privátní a zároveň daný učitel nevytvořil žádný předmět, formulář se nevykreslí. Pokud ale existují veřejné předměty nebo pokud danému učiteli patří alespoň jeden privátní předmět, formulář se vykreslí.

Poté se zobrazí formulář s možnostmi výběru, které se přidávají v závislosti na tom, zda je předmět veřejný nebo privátní. V případě, že struktura je veřejná, bude přidána možnost, ale pokud je privátní, bude nutné se dále zeptat na to, zda předmět vytvořil načítající učitel nebo jiný učitel. Pokud předmět vytvořil načítající učitel, bude mu přidána možnost i na jeho privátní strukturu. Kontrola, zda předmět patří načítajícímu učiteli, se provádí pomocí emailu, který se musí shodovat s emailem uloženým pod danou

strukturou v DB. Po výběru předmětu a stisknutí tlačítka „vytvořit“ se spustí metoda „CreateIfExist“.

Tato metoda a následně vyvolané API se postarají o vytvoření kopie dané struktury, kde se pouze změní ID kapitol a prerekvizit. Pokud byla struktura resetovaná, tak se pouze naváže na nový MS tým.

```
private async Task CreateIfExist() {
    //přidání ID skupiny
    group.TmGroup = (GroupToken.teamsContext.TeamId).ToString();
    //získání ID předmětu, který si učitel vybral
    searchSubject = subjects.Find(x => x.Name == subjectSelect);
    //vlastnictví skupiny
    group.Subject.Created = UserToken.Profile.Mail;
    //viditelnost předmětu --> vždy defaultně nastaveno na privátní
    group.Subject.Private = true;
    group.Subject.Name = searchSubject.Name.ToString();
    //pokud předmět byl pouze resetovaný zpátky ho naváže
    await Http.PostAsJsonAsync
    ($"{MyNavigationManager.BaseUri}api/skupina/pripoj", group);
    //získání ID nově navázané skupiny --> pokud neexistuje musí se struktura (předmět) překopírovat
    Group Getskupina = await Http.GetFromJsonAsync<Group>
    ($"{MyNavigationManager.BaseUri}api/skupina/" + GroupToken.teamsContext.TeamId);

    if(Getskupina.TmGroup == null) {
    //přidání příjmení učitela za název předmětu
    group.Subject.Name =
    $"{searchSubject.Name.ToString()} - {UserToken.Profile.Surname.ToString()}";
    modelHidden2 = false;
    }
    else {
    //vytvoří úvodní splnění pro navázanou skupinu s předmětem
    await VytvorUvodniSplneni(Getskupina);
    MyNavigationManager.NavigateTo("/");
    }
```

Obrázek 60 Metoda pro propojení či překopírování existující struktury se skupinou

8.4.2 Vytvoření úvodního splnění

Úvodního splnění se provádí, aby student po připojení k dané skupině měl přístup k úplně první kapitole. Princip spočívá v tom, že první kapitola má prerekvizitu úvod, která se při propojení studenta automaticky splní. Tím pádem student získá přístup k úvodní kapitole, která by jinak zůstala zamčená.

Při vytvoření struktury a skupiny se vytvoří i úvodní splnění, které je jako všechny ostatní splnění spojeno s danou skupinou. Úvodní splnění je vždy označeno 0.

```
private async Task CreateIntroductionCompletion(Group group) {
    Completion completion = new Completion();
    completion.GroupID = group.Id;
    completion.ChapterID = 0;
    await Http.PostAsJsonAsync($"{MyNavigationManager.BaseUri}api/skupina/uvod", completion);
}
```

Obrázek 61 Vytvoření úvodního splnění

Na Obrázek 61 je vidět, že nejdříve se vytvoří model completion. Získaní ID MS týmu se zaznamená v tomto modelu, který se následně použije pro vytvoření úvodního splnění. V neposlední řadě se nastaví ID splnění na 0, což značí, že se jedná o úvodní prerekvizitu, Po naplnění modelu daty je zaslán pomocí API požadavku POST k vytvoření záznamu v DB. Díky tomu je vytvořeno úvodní splnění, které je navázané pod danou skupinou v aplikaci v příslušném MS týmu.

8.5 Vytvoření a navázání studenta do daného Teamu

Pokud je skupina se strukturou vytvořena, může se student k ní už připojit. Nicméně, než se připojí, je ho potřeba vytvořit v DB. To se děje proto, aby byl student spojen pod danou skupinou i s jeho dosavadním progresem v plnění kapitol. Vytvoření studenta probíhá při úvodním načítání aplikace v MS týmu.

```
else if (UserToken.Profile.JobTitle.ToLower() == "žák školy")
{
    student = await Http.GetFromJsonAsync<Student>
    ($"{MyNavigationManager.BaseUri}api/studenti/{group.Id}/{UserToken.Profile.Mail}");
    //pokud student neexistuje v databázi vytvoří se a propojí se skupinou
    if (student.Email == null)
    {
        studentInfo.Name = UserToken.Profile.GivenName;
        studentInfo.Surname = UserToken.Profile.Surname;
        studentInfo.Email = UserToken.Profile.Mail;
        studentInfo.GroupStudents.Add(new GroupStudent());
        studentInfo.GroupStudents[0].GroupID = group.Id;
        await Http.PostAsJsonAsync
        ($"{MyNavigationManager.BaseUri}api/studenti", studentInfo);
    }
    //pokud student už existuje v databázi, ale není navázán v naší skupině v DB
    else if (student.GroupStudents.Find(x => x.GroupID == group.Id) == null)
    {
        skupinaStudentConnect.GroupID = group.Id;
        skupinaStudentConnect.StudentID = student.Id;
        await Http.PostAsJsonAsync
        ($"{MyNavigationManager.BaseUri}api/studenti/connect", skupinaStudentConnect);
    }
    MyNavigationManager.NavigateTo("/");
```

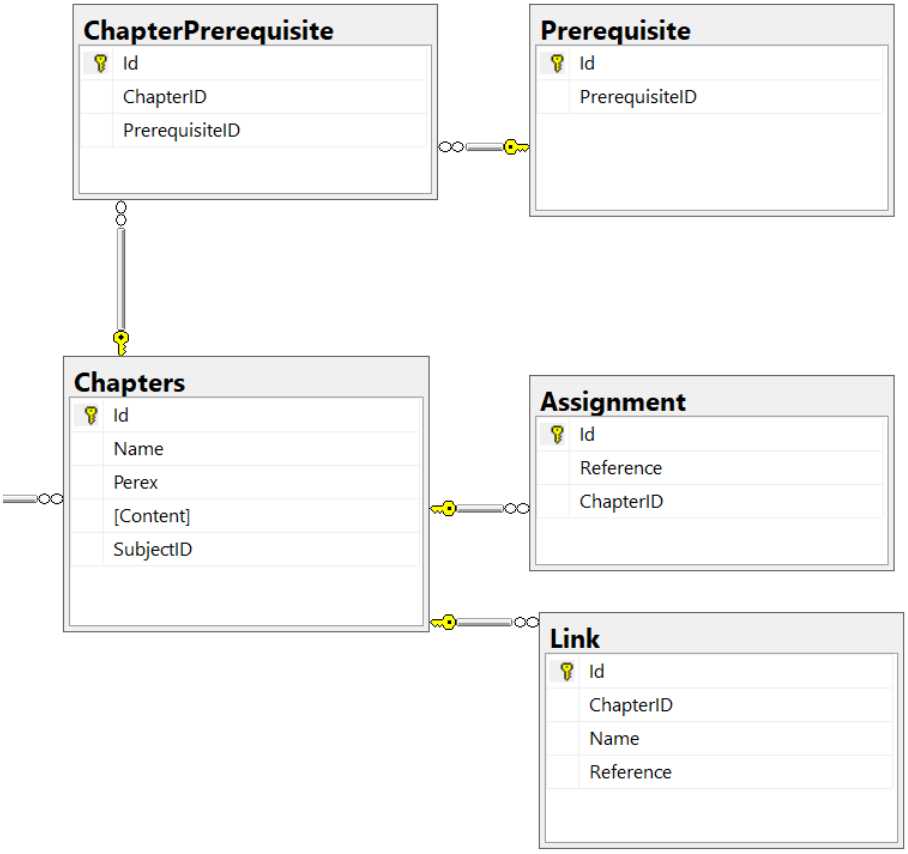
Obrázek 62 Vytvoření studenta do DB

Nejprve pomocí API se zjistí, zda je student již v DB vytvořený. Pokud není, vytvoří studenta s jeho informacemi, které jsou uložené v sessionstorage. Poté se student naváže s daným MS týmem pomocí modelu GroupStudent (tabulka mezi studentem a skupinou), kterému se předá ID MS týmu. Po POST požadavku se vytvoří student, který se automaticky spojí s danou skupinou v aplikaci a získá přístup k úvodní kapitole.

Pokud je student již v DB, pouze se propojí s daným MS týmem pomocí modelu GroupStudent, kde se dosadí ID studenta s ID MS týmu.

8.6 Vytvoření kapitol

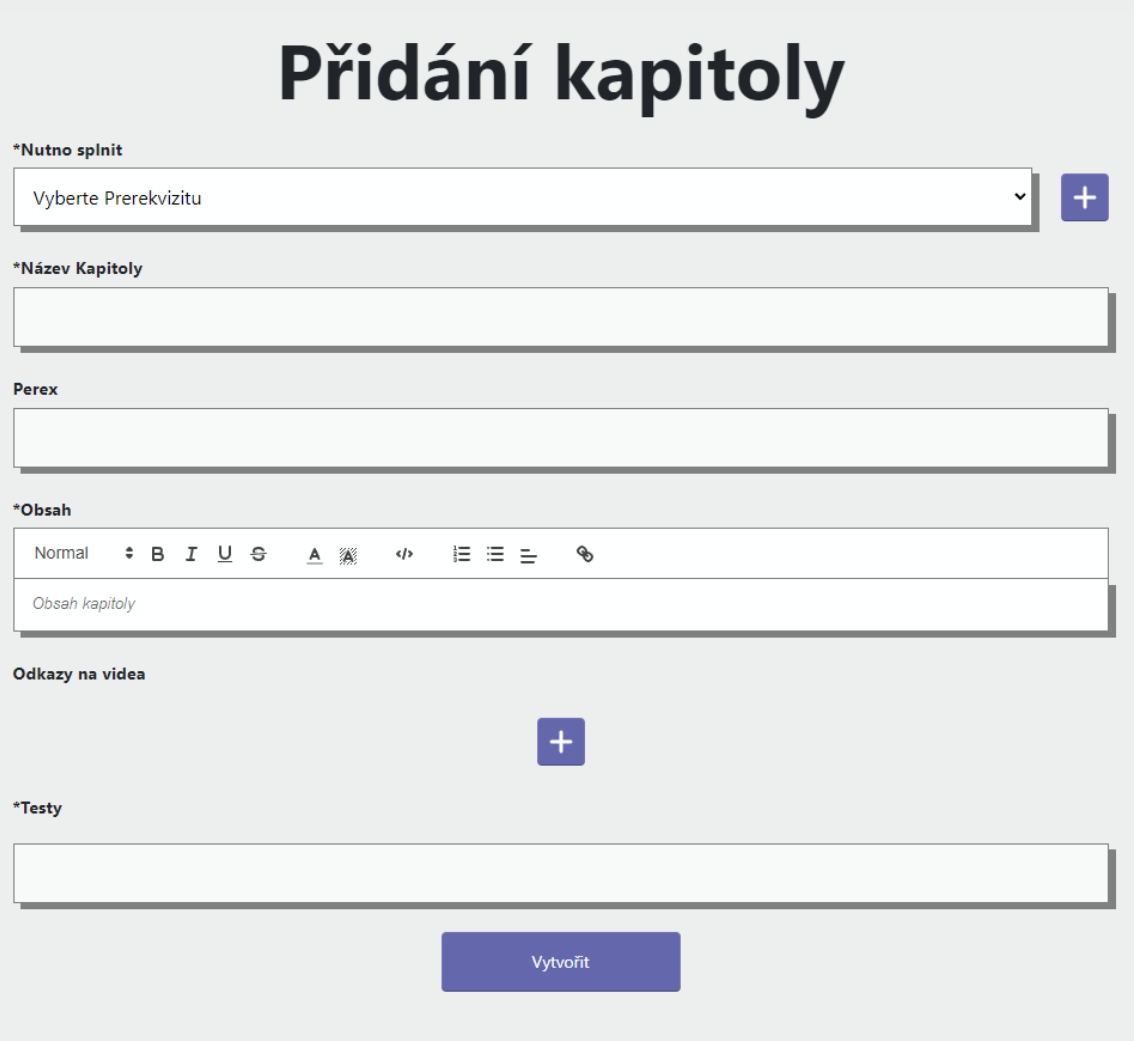
Učitel může vytvářet/upravovat/mazat jednotlivé kapitoly v rámci struktury, které studenti mohou přistupovat na základě splněných prerekvizit. Kapitola zahrnuje nadpis, perex a hlavní obsah, kde je popsána určitá problematika a odkaz na zadání v podobě MS Forms nebo jiných webových aplikacích třetích stran. Poté může kapitola obsahovat volitelné odkazy například na webové stránky. V neposlední řadě obsahuje podmínku, kde maximálně čtyři prerekvizity musí být splněny pro odemčení dané kapitoly. V první řadě bylo potřeba vymyslet databázovou strukturu, která se týká kapitoly.



Obrázek 63 Kapitola a její struktura v DB

Na Obrázek 63 je vidět, že pro odkazy a zadání je použit vztah 1:M, protože každá kapitola má jiné odkazy. Prerekvizity jsou ve vztahu M:N, protože určitá prerekvizita, která slouží jako podmínka pro odemčení kapitoly může být zároveň ve více kapitolách.

Po vytvoření struktury v DB je potřeba vytvořit formulář, která bude umožnovat přijímání dat a jejich následné zapsání do DB.



Obrázek 64 Formulář pro vytvoření kapitoly

Formulář je tvořen pomocí editform a odkazuje na model kapitola, který obsahuje modely pro odkazy, zadaní a prerekvizity. Formulář obsahuje základní formulářové prvky, jako vstup pro text, výběr, ale také wysiwyq editor. Pokud projde validací pomocí FluentValidationValidator dojde k vytvoření kapitoly pod příslušnou strukturou. Data jsou následně uložena do DB v odpovídajících tabulek podle modelů.

8.6.1 Wysiwyq editor

Jedná se o editor, který zobrazuje text či grafiku tak, jak bude vypadat po publikaci. Zkratka WYSIWYG znamená „What You See Is What You Get“. [22]

V tomto případě byl použit pro úpravu hlavního obsahu kapitoly, který umožňuje různé formátování jako je například zvýraznění, podtržení, zvětšení či zmenšení písma. Dále umožnuje vytváření odrážek, odkazů nebo speciálních tabulek pro výstřižek kódu.

Knihova, která byla použita:

- Blazored.TextEditor [7]

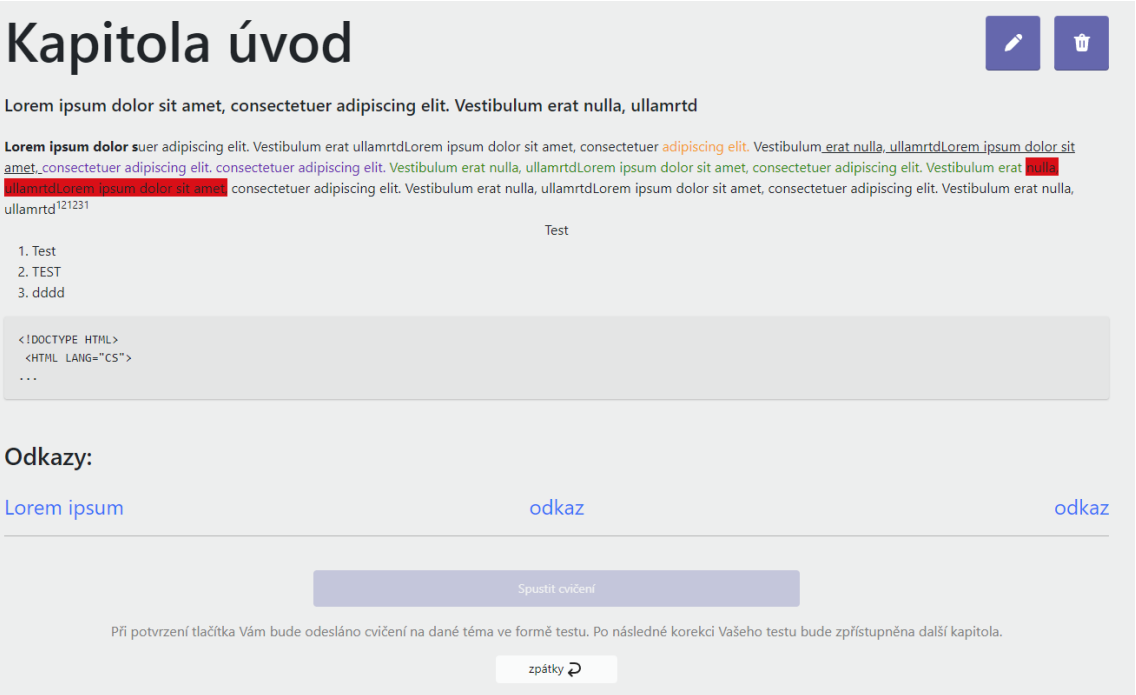
```
<BlazoredTextEditor @ref="@QuillHtml" Placeholder="Obsah kapitoly">
    <ToolbarContent>
        <select class="ql-header">
            <option selected=""></option>
            <option value="1"></option>
            <option value="2"></option>
            <option value="3"></option>
            <option value="4"></option>
            <option value="5"></option>
        </select>
        <span class="ql-formats">
            <button class="ql-bold"></button>
            <button class="ql-italic"></button>
            <button class="ql-underline"></button>
            <button class="ql-strike"></button>
        </span>
            <span class="ql-formats">
                <button class="ql-script" value="sub"></button>
                <button class="ql-script" value="super"></button>
            </span>
        <span class="ql-formats">
            <select class="ql-color"></select>
            <select class="ql-background"></select>
        </span>
        <span class="ql-formats">
            <button class="ql-code-block" value="code-block"></button>
        </span>
        <span class="ql-formats">
            <button class="ql-list" value="ordered"></button>
            <button class="ql-list" value="bullet"></button>

            <select class="ql-align">
                <option selected=""></option>
                <option value="center"></option>
                <option value="right"></option>
                <option value="justify"></option>
            </select>
        </span>
        <span class="ql-formats">
            <button class="ql-link"></button>
            <button class="ql-video"></button>
        </span>
    </ToolbarContent>
```

Obrázek 65 Wysiwyq editor

Na Obrázek 65 je patrné, že wysiwyq editor se skládá ze dvou hlavních – ToolbarContent a vstup. ToolbarContent představuje menu nástrojů, které se používají pro úpravu textu, zatímco vstup obsahuje text, který je pomocí třídy QuillEditor ukládán do proměnné QuillHtml. Následně se může s QuillHtml pracovat pomocí předdefinovaných metod „GetText“ a „GetHtml“.

Veškeré formátování, které bylo provedeno pomocí wysiwyg editoru, je uloženo pomocí HTML zápisu. To znamená, že do DB se ukládat text s HTML formátováním. Při načítání kapitoly se poté používá metoda „GetText“, který tento text s HTML formátováním převede na běžný text s formátováním.



Obrázek 66 Náhled na vytvořenou kapitolu s formátovaným textem

8.7 Načítání kapitol

Záleží, jestli uživatel, které kapitoly načítá je učitel či student. Zobrazení kapitol podle role je popsané v kapitole 6.1 a 6.2.

Role se zjistí ze sessionstorage, kde jsou uložené všechny informace o aktuálním uživateli získané z úvodní stránky.

Při načtení stránky je nejprve nutné zjistit informace o skupině z DB. Pokud by DB nevrátila žádná data, znamená to, že došlo k resetování skupiny nebo smazání její struktury. V takovém případě bude student přesměrován na úvodní stránku.

Po načtení skupiny se následně načtou všechny kapitoly, které patří pod danou strukturu. K tomu se používá parametr ID struktury, který se získal pomocí API výše.

```
group = await Http.GetFromJsonAsync<Skupina>
($"{MyNavigationManager.BaseUri}api/Skupina/" + (GroupToken.teamsContext.TeamId).ToString());
//pokud by skupina byla resetovaná či smazaná --> vrátí studenta k úvodní stránce
if(group.TmSkupina == null) {
    MyNavigationManager.NavigateTo("/tab");
}

chapters = await Http.GetFromJsonAsync<List<vyukovy_pavouk.Data.Kapitola>>
($"{MyNavigationManager.BaseUri}api/Kapitoly/" + (Convert.ToInt32(group.PredmetID)));

//pokud je studenta, zjistí informace o progresu atd..
if(UserToken.Profile.JobTitle == "Žák školy") {
    student = await Http.GetFromJsonAsync<Student>
    ($"{MyNavigationManager.BaseUri}api/studenti/" + Convert.ToInt32(group.Id)
    + "/" + UserToken.Profile.Mail);
}
```

Obrázek 67 Načtení stránky s kapitolami

Pokud se jedná o studenta, načte se s ním i informace o jeho plnění kapitol. V API máme dva parametry, kde první přijímá ID skupiny, která se zjistila pomocí API, a druhým parametrem je email studenta, který se získá ze sessionstorage.

Po zjištění informací o skupině a načtení kapitol dochází k vykreslení stránky s ohledem na roli uživatele (učitel x student). Pokud se jedná o učitele, jsou mu zobrazeny všechny kapitoly, ke kterým má volný přístup. Tyto kapitoly jsou jednotlivě vykresleny pomocí komponent, které přijímají tři parametry, které tvoří obsah zobrazený v kapitole.

```
@if(chapters.Count() > 0) {
    @if(UserToken.Profile.JobTitle == "Učitel" || UserToken.Profile.JobTitle == "Administrátor") {
        //zobrazení kapitol pro učitele
        @foreach (var chapter in chapters)
        {
            <Kapitola_ucitel ID="@chapter.Id" Nazev="@chapter.Název" Perex="@chapter.Perex" />
        }
    }
```

Obrázek 68 Určení načítání kapitol pro učitele

Obrázek 69 Vykreslení kapitol pro učitele

Při načítání kapitol studentem se musí provést kontrola každé načítané kapitoly, aby se zjistilo, zda ji student splnil úspěšně, neúspěšně anebo pokud je odemčená nebo zamčená. Kontrola se provádí pomocí metody „Check“, která vrátí hodnotu, podle které se zjistí stav kapitoly (viz kapitola 8.8).

```
@foreach (var chapter in chapters)
{
    int status = Check(chapter);
    @if (status == 1)
    {
        <Kapitola_splneno ID="@chapter.Id"
        Nazev="@chapter.Název"
        Perex="@chapter.Perex"
        Prerekvizita="@chapter.KapitolaPrerekvizita[0]"/>
    }
    else if(status == 2) {
        <Kapitola_odemceno ID="@chapter.Id"
        Nazev="@chapter.Název"
        Perex="@chapter.Perex"
        Prerekvizita="@chapter.KapitolaPrerekvizita[0]"
        Uspech="false"  />
    }
    else if (status == 3)
    {
        <Kapitola_odemceno ID="@chapter.Id"
        Nazev="@chapter.Název"
        Perex="@chapter.Perex"
        Prerekvizita="@chapter.KapitolaPrerekvizita[0]"
        Uspech="true"  />
    }
    else
    {
        PrerekvizityShow(chapter);
        <Kapitola_uzavreno ID="@chapter.Id"
        Nazev="@chapter.Název"
        Perex="@chapter.Perex"
        Prerekvizity="@searchPrerekvizity" />

    }
```

Obrázek 70 Určení vykreslení kapitol pro studenta podle statusu

8.8 Určení splnění/odemčení/uzamknutí kapitoly (status)

Jak bylo zmíněno, určení statusu se provádí pomocí metody „Check“. Ta může vrátit 4 možné hodnoty, podle kterých se určuje stav splnění kapitoly.

Možné stavy:

Tabulka 1 Statusy kapitol

Návratová hodnota	Status
1	Kapitola byla studentem úspěšně splněna.
2	Kapitola je neúspěšně splněna.
3	Kapitola je odemčena.
4	Kapitola je uzamčena.

Nejdříve dojde k první fázi, kde metoda zjistí, zdali je splněna úspěšně či neúspěšně.

```
ChapterPrerequisite save = new ChapterPrerequisite();
foreach (var completion in student.StudentCompletion
    .Select(x => x.Completion))
{
    //pokud je splněno
    if(completion.ChapterID == chapter.Id && student.StudentCompletion
        .Find(x => x.Completion.Id == completion.Id).Success == true) {
        return 1;
    }
    // neúspěšně splněno
    else if (completion.ChapterID == chapter.Id && student.StudentCompletion
        .Find(x => x.CompletionID == completion.Id).Success == false)
    {
        return 2;
    }
    //ukládání pokud je odemčena
    save = chapter.ChapterPrerequisites.Find(x => x.Prerequisite.PrerequisiteID == completion.ChapterID);
    if (save != null) {
        prerequisites.Add(save);
    }
}
```

Obrázek 71 Zjištění splnění (úspěšné x neúspěšné)

Zjištění probíhá tak, že se postupně projíždí jednotlivé splnění studenta. Každé splnění obsahují ID splněné kapitoly, které se porovnává s ID načítané kapitoly. Pokud se tato ID shodují, víme, že student danou kapitolu splnil, ale nevíme, zda úspěšně či neúspěšně. Úspěšnost splnění se zjistí pomocí modelu studentCompletion, který navazuje na model completion u studenta. V modelu studentCompletion se nachází vlastnost success, která vrací hodnotu „true“, pokud je kapitola úspěšně splněna nebo „false“, pokud je kapitola neúspěšně splněná.

Následně pokud není kapitola splněna se jednotlivé prerekvizity načítané kapitoly porovnávají se studentovými splněními. Pokud se najde shoda uloží ji do listu prerequisites.

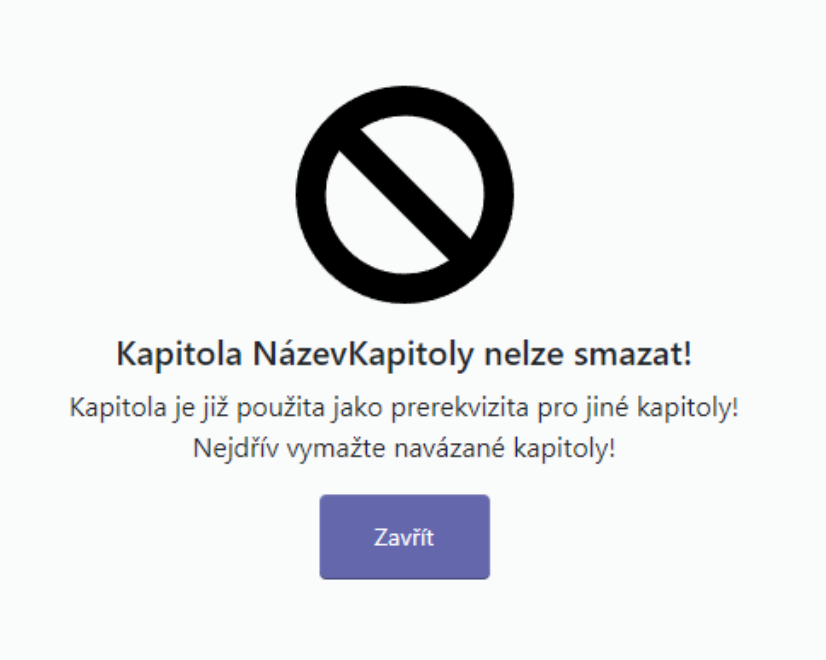
Po projetí studentových splnění se porovná počet prerekvizit načítané kapitoly s počtem splněných prerekvizit uložených v listu prerequisites. Pokud se počet neshoduje znamená to, že je kapitola uzamčena. Pokud se počet shoduje, provede se další kontrola, zda student splnil všechny požadované prerekvizity úspěšně. To se děje proto, aby nedošlo k zobrazení určité kapitoly, kde student nesplnil všechny požadované prerekvizity úspěšně. Proto se postupně porovnávají jednotlivé prerekvizity načítané kapitoly se studentovými splněními. Pomocí vlastnosti success v modelu studentCompletion, se zjistí, zda byla nějaká prerekvizita (kapitola) neúspěšně splněna. Pokud se najde neúspěšně splněná prerekvizita, kapitola zůstává uzamčena. Jinak, pokud jsou všechny prerekvizity (kapitoly) úspěšně splněny, kapitola se odemkne (viz Obrázek 72).

```
if(prerequisites.Count() == chapter.ChapterPrerequisites.Count()) {
    //kontrola zdali není
    foreach (ChapterPrerequisite chapterPrerequisite in chapter.ChapterPrerequisites)
    {
        int IDChapterPrerequisite = chapterPrerequisite.Prerequisite.PrerequisiteID;
        StudentCompletion studentTestCompletion = new StudentCompletion();
        studentTestCompletion = student.StudentCompletion
        .Where(x => x.Success == false && x.Completion.ChapterID == IDChapterPrerequisite)
        .SingleOrDefault();
        if(studentTestCompletion != null) {
            return 4;
        }
    }
    return 3;
}
//uzamčená kapitola
else {
    return 4;
}
```

Obrázek 72 Zjištění, zdali je kapitola odemčená či uzamčena

8.9 Mazání kapitol

Pokud učitel se rozhodne smazat určitou kapitolu, může tak učinit pouze v případě, že daná kapitola není potřeba pro splnění jiné kapitoly. Pokud by bylo nutné pro odemčení určité kapitoly mít kapitolu, kterou učitel smazal, studenti by nemohli pokračovat v plnění dalších kapitol, a proto je důležité pečlivě zvážit, zda je možné danou kapitolu smazat. Při mazání kapitoly se musí také smazat všechna splnění studentů, která se týkala této kapitoly. Pokud by student totiž splnil již smazanou kapitolu, zůstal by záznam o splnění v tabulce, což by vedlo k nesprávnému vyhodnocení celkového postupu studenta v souhrnu, protože by se počítaly i splněné kapitoly, které již neexistují.



Obrázek 73 Informativní dialog

Samotné mazání kapitoly funguje pomocí požadavku DELETE, kde se předá ID kapitoly, které se chce smazat.

```
private async Task DeleteChapter() {
    turnOffBtn = true;
    //smazání samotné kapitoly
    await Http.DeleteAsync
    ($"{MyNavigationManager.BaseUri}api/kapitola/delete/{chapter.Id}");
    MyNavigationManager.NavigateTo("/");
    notice.ShowSuccess("Kapitola byla úspěšně vymazaná.", "Smazáno!");
```

Obrázek 74 Volání požadavku pro smazání kapitoly.

To následně vyvolá v kontrolérů metodu v ChapterManager, která se postará o smazání jak samotné kapitoly z DB, tak i o případné smazání z tabulky prerequisites, za předpokladu, že byla použita jako prerekvizita. Dojde i k smazání splnění studentů zmíněné výše a k smazání prerekvizit, které nejsou navázané na žádnou kapitolu.

```
public async Task DeleteChapter(int IdChapter)
{
    Chapter chapter = _dbContext.Chapters.Where(x => x.Id == IdChapter)
                                         .Include(x => x.ChapterPrerequisites)
                                         .SingleOrDefault();
    if (chapter != null)
    {
        _dbContext.Chapters.Remove(chapter);
        //vymazání prerekvizity, kde byla použita tato kapitola
        Prerequisite prerequisite =
            await _dbContext.Prerequisite.Where(x => x.PrerequisiteID == IdChapter)
                                         .SingleOrDefaultAsync();
        if (prerequisite != null)
        {
            _dbContext.Prerequisite.Remove(prerequisite);
        }
        //vymazání prerekvizit, které nenavazují na nic
        foreach (ChapterPrerequisite chapterPrerequisite in chapter.ChapterPrerequisites)
        {
           List<ChapterPrerequisite> chapterPrerequisites =
                await _dbContext.ChapterPrerequisite.Where(x => x.PrerequisiteID
                                                    == chapterPrerequisite.PrerequisiteID)
                                                    .Include(x => x.Prerequisite)
                                                    .ToListAsync();
           if(chapterPrerequisites.Count() == 1)
            {
                _dbContext.Prerequisite.Remove(chapterPrerequisites
                                          .Select(x => x.Prerequisite)
                                          .SingleOrDefault());
            }
        }
        //vymazání splnění a navázaní splnění na studentech
        List <Completion> completion =
        await _dbContext.Completion.Where(x => x.ChapterID == IdChapter).ToListAsync();
        if(completion.Count != 0)
        {
            _dbContext.Completion.RemoveRange(completion);
        }
        await _dbContext.SaveChangesAsync();
    }
```

Obrázek 75 Smazání kapitoly se všemi náležitostmi

8.10 Souhrn studentů

Slouží k zobrazení studentů, kteří spadají pod určitou skupinu. V souhrnu má učitel přehled o počtu splněných kapitol studentů či jejich procentuálním plnění kapitol. Učitel může nahlédnou i přesně jaké kapitoly má student splněné. V souhrnu se může odkázat i na povolení kapitol (úspěšné x neúspěšné) u jednotlivého studenta. Lze zde i smazat určitého studenta ze skupiny.

Zjištěný počtu splněných kapitol probíhá pomocí API požadavku, který zjistí všechny studenty a jejich progres v plnění kapitol. Pro výpočet procentuálního plnění je dále nutné získat maximální počet kapitol.

```
//získání skupiny z databáze
group = await Http.GetFromJsonAsync<Group>
($"{MyNavigationManager.BaseUri}api/Skupina/{GroupToken.teamsContext.TeamId}");
// získání uživatelů a jejich progres v plnění kapitol
students = await Http.GetFromJsonAsync<List<GroupStudent>>
($"{MyNavigationManager.BaseUri}api/studenti/{group.Id}");
//získání počtů všech dostupných kapitol
if (students.Count() > 0)
{
    maxCountChapter = await Http.GetFromJsonAsync<int>
    ($"{MyNavigationManager.BaseUri}api/predmet/{group.Subject.Id}");
}
```

Obrázek 76 Získání studentů pomocí API

API pro zjištění studentů s jejich stavem plnění kapitol z DB nejdříve vyvolá v kontroléru metodu v rozhraní, kde přijímá pouze jeden parametr, a to ID skupiny.

```
public List<GroupStudent> GetStudents(int ID)
{
    return _dbContext.GroupStudent
            .Where(s => s.GroupID == ID)
            .Include(s => s.Student)
            .ThenInclude(s => s.StudentCompletion
            .Where(id => id.Completion.GroupID == ID && id.Completion.ChapterID != 0))
                .ThenInclude(s => s.Completion)
            .ToList();
}
```

Obrázek 77 Zjištění studentů a jejich plnění kapitol patřící pod skupinou

V tomto případě se do listu students uloží studenti z DB, kteří jsou přiřazeni ke konkrétní skupině. Následně se zjistí jejich plnění kapitol a načtou se všechny úspěšně či neúspěšné splněné kapitoly, s výjimkou úvodní prerekvizity, kterou student získá automaticky.

Po zjištění údajů se studenti načítají do tabulky, která zobrazuje jméno a příjmení studenta, počet úspěšně splněných kapitol / počet všech kapitol pod strukturou, procentuální plnění kapitol, tlačítko odkazující na zobrazení všech úspěšně splněných kapitol (progres) a tlačítko odkazující na povolení kapitol. Pokud ve struktuře neexistuje žádná kapitola, vypíše se tato informace.

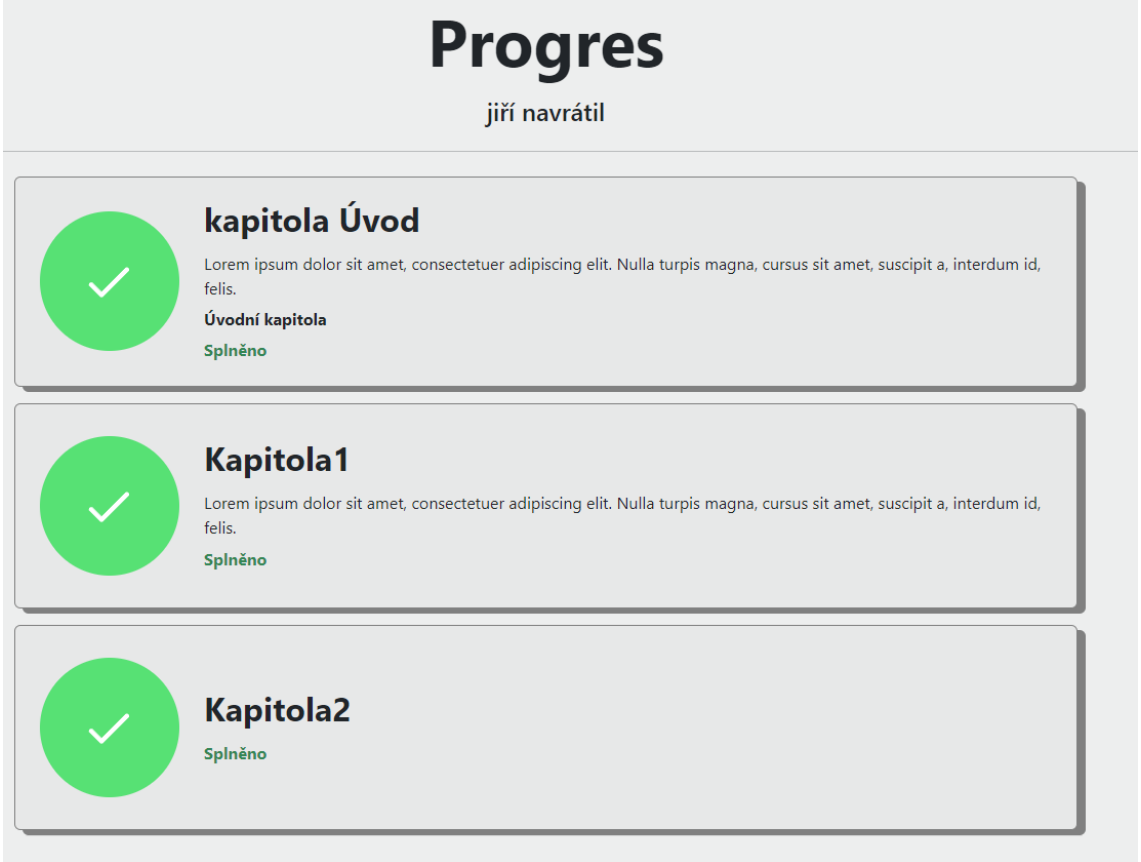
```
<th scope="row">
    <p class="d-flex align-items-center justify-content-center h-100 mt-1 mb-0">
        @student.Student.Name @student.Student.Surname</p></th>
@if (maxCountChapter > 0)
{
    <td><p class="d-flex align-items-center mt-1 mb-0">
        @student.Student.StudentCompletion.Where(x => x.Success == true).Count()/@maxCountChapter</p></td>
    <td class="d-none d-lg-table-cell"><p class="d-flex align-items-center mt-1 mb-0">
        @CalculateAVG(student.Student.StudentCompletion.Where(x => x.Success == true).Count())%</p></td>
    <td><fluent-button appearance="accent" class="d-grid align-items-center"
        @onclick="() => ShowResult(student.Student.Id)"><svg width="28" height="28" fill="none" viewBox="0
    <td><fluent-button appearance="accent" class="d-grid align-items-center"
        @onclick="() => Transfer(student.Student.Id)"><svg width="28" height="28" fill="none" viewBox="0 0
}
else
{
    <td colspan="4"><p class="d-flex align-items-center mt-1 mb-0
    @(dark ? "text-white" : "")">V teamu není žádná kapitola!</p></td>
}
```

Obrázek 78 Zobrazení studenta v souhrnu u frontendu

Obrázek 79 Vykreslení výpisu studenta v souhrnu

Pokud se stiskne tlačítko u „více info“, bude učitel přesměrován na stránku progres, na které jsou zobrazeny podrobnější informace o úspěšně splněných kapitolách konkrétního studenta. Tato informace pomáhá učiteli získat přehled o tom, které kapitoly přesně student splnil (viz Obrázek 80).

Pokud se stiskne tlačítko u „povolení“ učitel se přesměruje na stránku, kde může zaznamenat, zda daný student úspěšně či neúspěšně splnil odemčenou kapitolu (odemčená kapitola z pohledu daného studenta).

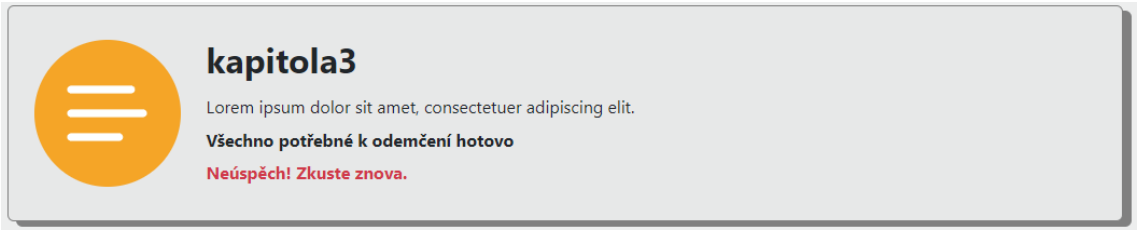


Obrázek 80 Progres u určitého studenta v plnění kapitol

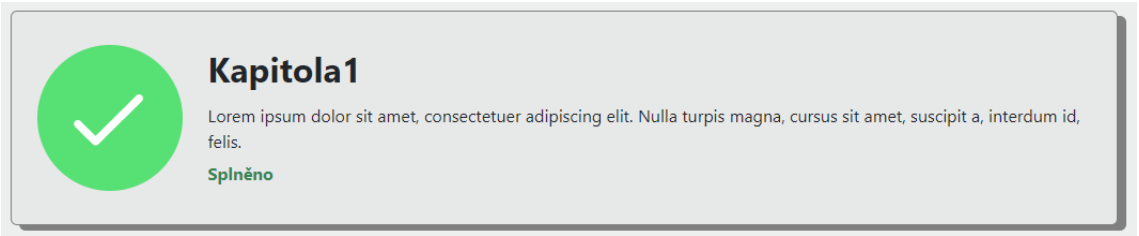
8.11 Povolení jednotlivých kapitol

Tato funkce slouží učiteli pro povolení určitých kapitol konkrétnímu studentovi. Učitel může povolit splnění pouze těch kapitol, které jsou již studentovi odemčeny. Pokud učitel povolí splnění kapitoly, tato kapitola bude považovaná za úspěšně splněnou a studentovi se odemknou další kapitoly na základě definovaných prerekvizit. Pokud učitel zamítne splnění kapitoly, student musí znovu splnit zadání.

Obrázek 81 Povolení kapitoly u studenta



Obrázek 82 Zobrazení studentovi při neúspěšném splnění



Obrázek 83 Zobrazení studentovi při úspěšném splnění

Při načtení stránky pro povolení kapitol se spustí API požadavky pro zjištění údajů, které budou potřeba pro zjištění odemčených kapitol.

První API požadavek zjistí podle uloženého ID MS týmu v sessionstorage, jakou skupinu používáme a jaká struktura je navázaná v DB. Druhý požadavek slouží k načtení všech kapitol včetně jejich prerekvizit. Kapitoly budou načteny na základě ID struktury, která byla získaná v prvním požadavku. Poslední požadavek načte progres studenta (jeho plnění kapitol) na základě ID skupiny a ID studenta.

```
group = await Http.GetFromJsonAsync<Skupina>
($"{MyNavigationManager.BaseUri}api/Skupina/" + (GroupToken.teamsContext.TeamId).ToString());
chapters = await Http.GetFromJsonAsync<List<vyukovy_pavouk.Data.Kapitola>>
($"{MyNavigationManager.BaseUri}api/Kapitoly/" + (Convert.ToInt32(group.PredmetID)));
student = await Http.GetFromJsonAsync<Student>
($"{MyNavigationManager.BaseUri}api/studenti/progres/" + Convert.ToInt32(group.Id) + "/" + id);
```

Obrázek 84 API požadavky při načtení stránky pro povolení kapitol

```
@foreach (vyukovy_pavouk.Data.Chapter chapter in chapters)
    {
        int status = ShowOnlyUnlocked(chapter);
        if(status == 3 || status == 2) {
            NotEmpty = true;
            break;
        }
    }
```

Obrázek 85 první fáze – zjištění dostupnosti odemčených kapitol

Poté proběhne vykreslování stránky, kde přejde do první fáze (viz Obrázek 85 první fáze – zjištění dostupnosti odemčených kapitol), kde se zjistí, zdali je dostupná odemčená kapitola. Pokud není dostupná žádná odemčená kapitola vypíše se tento fakt.

Kontrola probíhá tak, že se projíždí veškeré kapitoly, která struktura obsahuje a poté se s určitou kapitolou pracuje v metodě „ShowOnlyUnlocked“, kde se zjistí, zdali je k určitému studentovi odemčená, splněná úspěšně či neúspěšně nebo uzamčena. V tomto případě se zajímá, pouze o to, zdali je aspoň jedna kapitola odemčená či neúspěšně splněná narozdíl u načítání kapitol na hlavní stránce.

```
ChapterPrerequisite save = new ChapterPrerequisite();
foreach (var completion in student.StudentCompletion
    .Select(x => x.Completion))
{
    //pokud je splněno
    if(completion.ChapterID == chapter.Id && student.StudentCompletion
        .Find(x => x.Completion.Id == completion.Id).Success == true) {
        return 1;
    }
    // neúspěšně splněno
    else if (completion.ChapterID == chapter.Id && student.StudentCompletion
        .Find(x => x.CompletionID == completion.Id).Success == false)
    {
        return 2;
    }
    //ukládání pokud je odemčena
    save = chapter.ChapterPrerequisites
    .Find(x => x.Prerequisite.PrerequisiteID == completion.ChapterID);
    if (save != null) {
        Prerequisites.Add(save);
    }
}
```

Obrázek 86 První část metody ShowOnlyUnlocked

Metoda funguje na úplně stejném principu jako metoda Check, kde její princip je vysvětlen v kapitole 8.8 s tím rozdílem, že vracené hodnoty mají jinou logiku.

```
if(Prerequisites.Count() == chapter.ChapterPrerequisites.Count()) {
    //kontroluje prerekvizitu kapitoly a studenta (neúspěšného splnění)
    // ==> pokud nastane nezobrazí další kapitoly pro odemčení a může se odemknout (na úspěšný stav)
    //pouze ta neúspěšná
    foreach (ChapterPrerequisite chapterPrerequisite in chapter.ChapterPrerequisites)
    {
        int IDChapterPrerequisite = chapterPrerequisite.Prerequisite.PrerequisiteID;
        StudentCompletion studentTestCompletion = new StudentCompletion();
        studentTestCompletion = student.StudentCompletion
        .Where(x => x.Success == false && x.Completion.ChapterID == IDChapterPrerequisite)
        .SingleOrDefault();
        if(studentTestCompletion != null) {
            return 1;
        }
    }
    return 3;
}
else {
    return 1;
}
```

Obrázek 87 Druhá část metody ShowOnlyUnlocked

Podle vrácených hodnot se ví, že:

Tabulka 2 Status kapitol u povolení

Návratová hodnota	Status
1	Je úspěšně splněná nebo v splnění kapitoly ji brání neúspěšné splněné prerekvizity nebo je uzamčena. Tím pádem se na stránce v povolení nevykreslí. Jedna hodnota vrací více věcí, o kterých se ví, že se nebudou vykreslovat na stránce.
2	Je neúspěšně splněna. Tím pádem se vykreslí na stránce, kde bude mít povoleno pouze tlačítko pro povolení splnění kapitoly.
3	Je odemčena. Tím pádem se vykreslí na stránce, kde budou povoleny obě tlačítka pro zamítnutí či povolení splnění kapitoly.

Ve druhé fázi načítání stránky, pokud je aspoň jedna kapitola odemčena či neúspěšně splněna se načtou jednotlivé odemčené či neúspěšně splněné kapitoly. To se pozná podle zmínění metody „ShowOnlyUnlocked“. Podle vrácených hodnot se pak vykreslí jednotlivé možnosti pro povolení odemčené kapitoly.

```
@foreach (vyukovy_pavouk.Data.Chapter chapter in chapters)
{
    int status = ShowOnlyUnlocked(chapter);
    @if (status == 3 || status == 2)
    {
        <tr>
        <td><p class="d-flex align-items-center mt-1 mb-0">@chapter.Name</p></td>
        @if (status == 3)
        {
            <td><fluent-button appearance="accent" Disabled="@clicked"
                @onclick="() => EnableNewChapter(chapter.Id, false)">
                <?xml version="1.0" encoding="iso-8859-1"?>
                <svg version="1.1" id="Capa_1" xmlns="http://www.w3.org/2000/svg" xmlns
            <td><fluent-button appearance="accent" Disabled="@clicked"
                @onclick="() => EnableNewChapter(chapter.Id, true)">
                <svg width="28" height="28" stroke="white" stroke-width="2" fill="none"
                    <path d="m8.5 16.586-3.793-3.793a1 1 0 0 0-1.414 1.414l4.5 4.5a1 1 (
        }
        else
        {
            <td><fluent-button appearance="accent" disabled="true">...
            <td><fluent-button appearance="accent" Disabled="@clicked"
                @onclick="() => RepairChapter(chapter.Id)">
                <svg width="28" height="28" stroke="white" stroke-width="2" fill="none"
                    <path d="m8.5 16.586-3.793-3.793a1 1 0 0 0-1.414 1.414l4.5 4.5a1 1 (
        }
```

Obrázek 88 Druhá fáze – Vykreslení jednotlivých povolení

U vykreslení se řeší dva typy povolení a to:

- Pokud kapitola ještě nebyla splněna, zobrazí se tlačítka pro zamítnutí (při neúspěchu) a pro povolení (při úspěchu). Po zaznamenání tohoto úkonu se v tabulce StudentCompletion u studenta vytvoří záznam (propojení mezi studentem a splněnou kapitolou). Pokud danou kapitolu ještě nikdo nesplnil, musí se předtím vytvořit i záznam v tabulce Completion, který obsahuje ID kapitoly a ID skupiny.

- Pokud kapitola byla splněna neúspěšně. Vykreslí se tlačítka stejně s tím rozdílem, že tlačítko pro zamítnutí bude zakázané. Půjde pouze danou kapitolu povolit. V DB se pouze změní v tabulce StudentCompletion vlastnost success na hodnotu „true“.

```
private async Task EnableNewChapter(int IdChapter, bool success) {
    clicked = true;
    StudentCompletion studentCompletion = new StudentCompletion()
    {Completion = new Completion()};
    studentCompletion.StudentID = student.Id;
    studentCompletion.Success = success;
    studentCompletion.Completion.ChapterID = IdChapter;
    studentCompletion.Completion.GroupID = group.Id;
    await Http.PostAsJsonAsync
    ($"{MyNavigationManager.BaseUri}api/studenti/splneni",studentCompletion);
    student = await Http.GetFromJsonAsync<Student>
    ($"{MyNavigationManager.BaseUri}api/studenti/progres/{group.Id}/{id}");
    if(success) {
        notice.ShowSuccess("Povoleno.", "Úspěch!");
    }
    else {
        notice.ShowSuccess("Nepovoleno.", "Úspěch!");
    }
    clicked = false;
    await InvokeAsync(StateHasChanged);
}
```

Obrázek 89 Prvotní povolení kapitoly (úspěšný x neúspěšný)

Metoda „EnableNewChapter“ se používá u prvního typu povolení. Přijímá dva parametry, ID Kapitoly a informaci o úspěchu, která je předána po stisku tlačítka. V této metodě se vytvoří model studentCompletion a pod ním model completion. Model studentCompletion se naplní daty, konkrétně ID studenta a informaci o úspěchu z parametru. Do modelu completion pod studentCompletion se dosadí ID kapitoly a ID skupiny, ke kterým se záznam vztahuje.

Tento model je následně vytvořen v DB požadavkem POST. Nejdříve vytvoří splnění a poté se naváže na studentCompletion k studentovi. Pokud by v tabulce completion existoval záznam o splnění této kapitoly, zjistí se jeho ID v DB a vytvoří se pouze navázání pomocí studentCompletion, kde se získané ID dosadí za completionID.

Po vytvoření je vyvolán znovu požadavek pro zjištění studenta a jeho plnění kapitol, které se v tomto případě změní oproti stavu před povolení určité kapitoly. Poté je pomocí metody InvokeAsync(StateHasChanged) stránka znovu vykreslena a tím pádem se zobrazí nové možnosti povolení, podle nových získaných dat. Toto se dělá proto, aby nebyla potřeba ruční obnovení stránky pokaždé, kdy dojde k zaznamenání (úspěšné či neúspěšné splnění) kapitoly určitému studentovi.

```
private async Task RepairChapter(int IdChapter) {
    clicked = true;
    StudentCompletion studentCompletion = student.StudentCompletion
    .Where(x => x.Completion.ChapterID == IdChapter && x.Completion.GroupID == group.Id)
    .SingleOrDefault();
    studentCompletion.Success = true;
    await Http.PutAsJsonAsync<StudentCompletion>
    ($"{MyNavigationManager.BaseUri}api/studenti", studentCompletion);
    student = await Http.GetFromJsonAsync<Student>
    ($"{MyNavigationManager.BaseUri}api/studenti/progres/{group.Id}/{id}");
    notice.ShowSuccess("Povoleno.", "Úspěch!");
    clicked = false;
    await InvokeAsync(StateHasChanged);
}
```

Obrázek 90 Oprava zamítnuté kapitoly

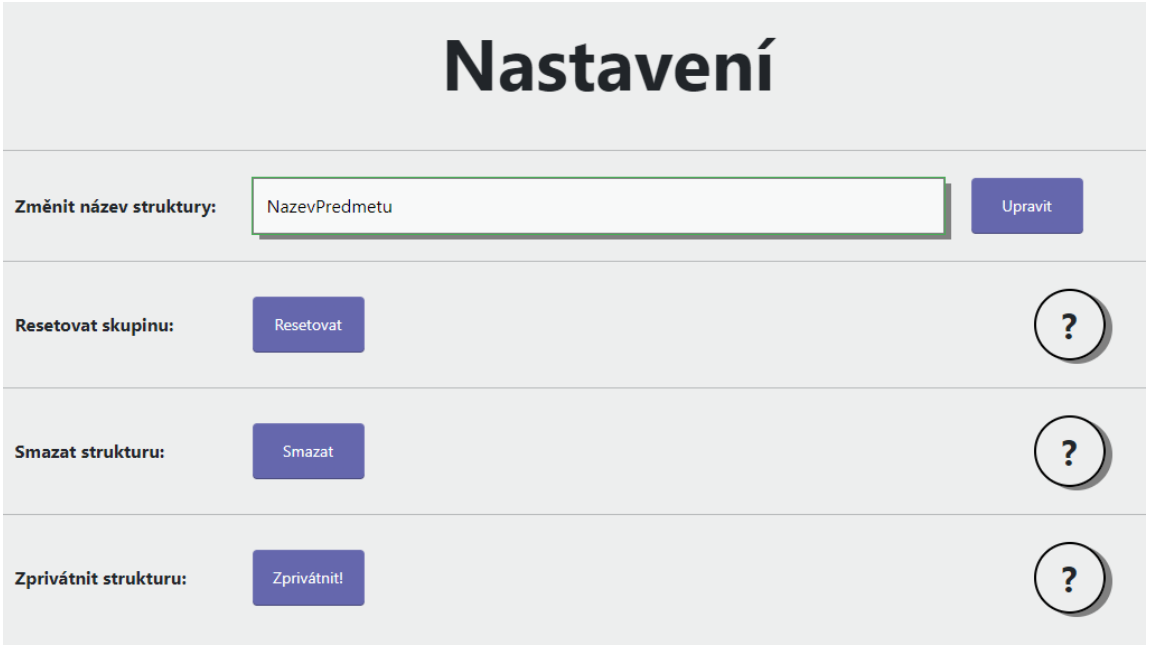
Metoda „RepairChapter“ slouží u druhého typu povolení. Tato metoda přijímá pouze jeden parametr, a to ID kapitoly. V metodě se vytvoří model studentCompletion, kde se hledá ze studentova plnění kapitol z modelu completion danou kapitolu, podle parametru a ID skupiny. To se dělá proto, aby se získalo existující navázání na neúspěšně splněnou kapitolu. Po získání stačí nastavit success na „true“, protože u druhého typu povolení jde pouze povolit kapitolu úspěšně, protože před tím byla neúspěšně splněna. Po požadavku PUT, kde nastane změna v DB se znova načte u studenta jeho splnění kapitol, kde nastala změna. Pak vyvolá metodu InvokeAsync(StateHasChanged), kde se znovu vykreslí stránka, stejně jako u metody „EnableNewChapter“.

8.12Nastavení

Nastavení slouží pouze učiteli k nastavení dané skupiny a struktury.

Nastavení obsahuje:

- **Změnu názvu struktury**
- **Resetování skupiny** – Vymaže ID MS týmu v tabulce group. Následně slouží k smazání veškerých splnění kapitol studentů k dané skupině a zároveň vymaže napojení studentů ke skupině v aplikaci. Struktura, která byla resetovaná lze znovu použít v jakékoliv MS týmu. Po resetování skupiny může učitel znovu přejít k vytvoření struktury.
- **Smazání struktury** –Vymaže celou strukturu včetně jejich kapitol. Zároveň vymaže splnění kapitol k dané skupině a napojení studentů ke skupině v aplikaci. Po smazání struktury může učitel znovu přejít k vytvoření struktury.
- **Změna viditelnosti struktury** – Lze změnit viditelnost, která byla vybrána při založení struktury (veřejná x privátní), podle toho pak budou moci použít (překopírovat) danou strukturu jiný učitelé.



Obrázek 91 Nastavení

Při načtení stránky nastavení dojde k požadavku pro zjištění aktuální skupiny a navázané struktury (předmětu) z DB. Poté dojde k dalšímu požadavku pro získání všech struktur, které existují v DB. To slouží ke kontrole změny názvu struktury. Kontrola zabrání změnám názvu, které by mohly mít existující struktury, a tím zabrání vzniku duplicitních názvů struktur. Po požadavcích se stránka vykreslí.

```
//získání skupiny z databáze a jeho navazaného předmětu
group = await Http.GetFromJsonAsync<Skupina>
($"{MyNavigationManager.BaseUri}api/Skupina/{(GroupToken.teamsContext.TeamId).ToString()}");
//získání všech předmětu --> pro kontrolu
subjects = await Http.GetFromJsonAsync<List<Predmet>>
($"{MyNavigationManager.BaseUri}api/predmet");
IsLoaded = true;
```

Obrázek 92 API požadavky při načítání nastavení

8.12.1 Úprava názvu struktury

Pokud po vyplnění vstupu u změny názvu struktury se klikne na upravit spustí se metoda, která se stará o změnu názvu struktury.

```
private async void ChangeName() {
    turnOffBtns = true;
    searchSubject = subjects.Where(x => x.Nazev == group.predmet.Nazev)
                            .SingleOrDefault();
    if(searchSubject != null) {
        conformityDialog = false;
        return;
    }
    await Http.PutAsJsonAsync($"{MyNavigationManager.BaseUri}api/predmet", group);
    subjects = await Http.GetFromJsonAsync<List<Predmet>>
    ($"{MyNavigationManager.BaseUri}api/predmet");
    await InvokeAsync(StateHasChanged);
    upozorneni.ShowSuccess("Předmět byl úspěšně přejmenován.", "Úspěch!");
    turnOffBtns = false;
}
```

Obrázek 93 Metoda starající se o změnu názvu struktury

Nejdříve se v metodě „ChangeName“ kontroluje, zdali se název vyplněný ve vstupu neshoduje s názvem existující struktury. Pokud by se shodoval, metoda se ukončí a zobrazí se dialogové okno pro informování o této skutečnosti. Pokud ne, pokračuje v PUT požadavku, který má na starost změnu názvu předmětu v DB. V neposlední řadě je znova vyvolán požadavek, pro získání všech struktur a poté je vyvoláno metodou InvokeAsync(StateHasChanged) znovu vykreslení stránky. Získávání všech struktur znovu se dělá kvůli tomu, aby byly stále aktuální informace o všech strukturách (názvů), které jsou použity při kontrole, zdali se nový zadaný název struktury neshoduje s názvem určité struktury v DB.

8.12.2 Resetování skupiny

Pro resetování skupiny je potřeba vyčkat 5 vteřin pro zpřístupnění tlačítka, který vyvolá tuto metodu pro její resetování.

```
private async Task ResetGroup() {
    turnBtnReset = true;
    await Http.DeleteAsync($"{MyNavigationManager.BaseUri}api/skupina/reset/" + group.Id);
    upozorneni.ShowSuccess("MS skupina byla resetovaná.", "Úspěch!");
    await Task.Delay(1000);
    MyNavigationManager.NavigateTo("/tab");
}
```

Obrázek 94 Metoda ResetGroup vyvolávající API pro resetování skupiny

Metoda vyvolá DELETE požadavek, kde se posílá ID skupiny, které se týká. To vyvolá v kontrolérů metodu, která vyvolá metodu „ResetGroup“ v GroupManager.

```
public async Task ResetGroup(int Id)
{
    //vymazání skupiny
    Group group = await _dbContext.Group
        .Where(x => x.Id == Id)
        .Include(x => x.Subject).SingleOrDefaultAsync();
    group.Subject.Private = true;
    group.TmGroup = "";
    _dbContext.Entry(group).State = EntityState.Modified;

    //vymazání napojení
    List<GroupStudent> SkupinaStudent = await _dbContext.GroupStudent
    .Where(x => x.GroupID == Id).ToListAsync();
   _dbContext.RemoveRange(SkupinaStudent);
    //vymazání splnění
    List<Completion> splneni = await _dbContext.Completion
        .Where(x => x.GroupID == Id).ToListAsync();
    _dbContext.RemoveRange(splneni);
    await _dbContext.SaveChangesAsync();
}
```

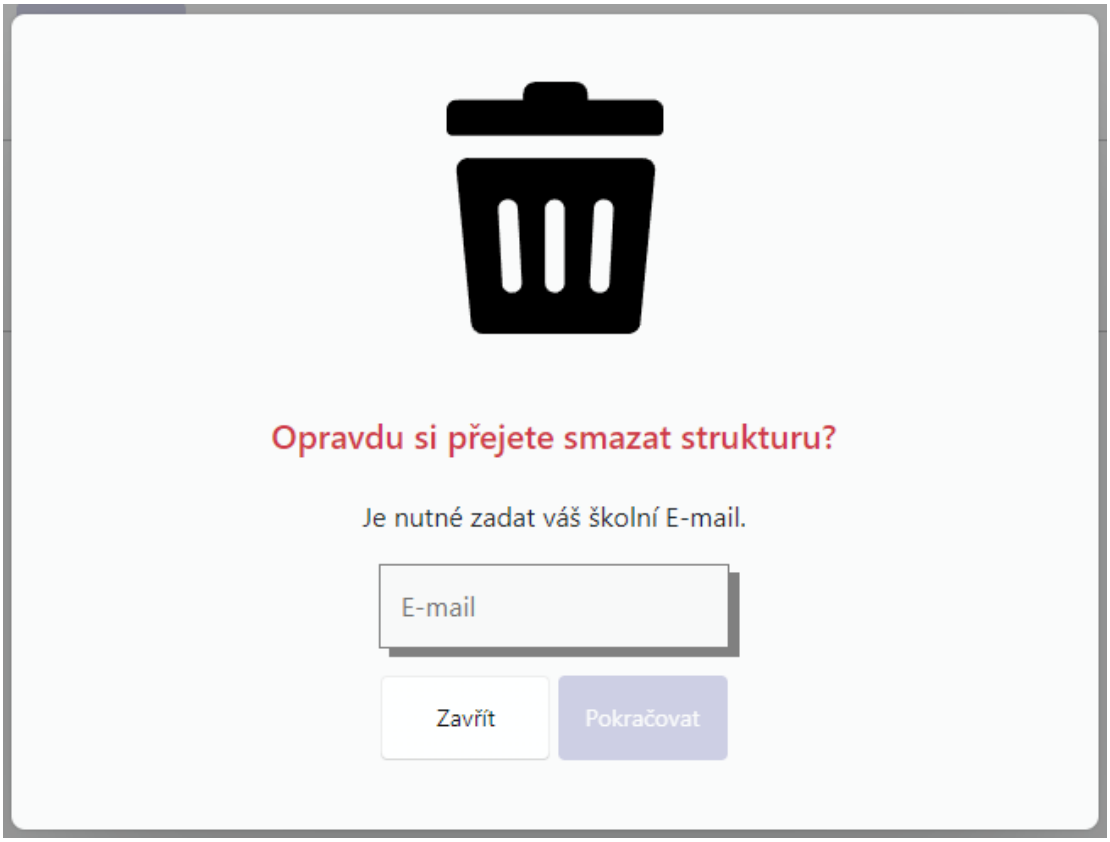
Obrázek 95 Metoda pro resetování skupiny

Danou skupinu resetuje tím způsobem, že dané skupině vymaže hodnotu TmGroup, což obsahuje ID MS týmu, kde se tato skupina včetně struktury nachází. Poté nastaví viditelnost struktury na privátní, aby strukturu mohl znovu použít pouze vlastník. Nakonec se vymažou všechny navázané studenty a jejich splnění v dané skupině.

Strukturu pak může znovu daný učitel použít. Jediná změna proběhne v nastavení TmGroup, kde se nastaví aktuální ID MS týmu, kde se nachází aplikace.

8.12.3 Smazání struktury

Slouží k úplnému smazání struktury včetně přiřazené skupiny. Nejdřív je nutné kliknout na tlačítko „Smazat“, což odkáže uživatele na vyplnění své emailového adresy. Toto je kvůli bezpečnostnímu ověření, že se jedná skutečně o učitele, který založil strukturu. Ověření emailové adresy se provádí pomocí metody „CheckEmail“, která se vyvolá pomocí eventu onkeyup. Pokud emailová adresa odpovídá s emailovou adresou zakladatele je zpřístupněno tlačítko „Pokračovat“.



Obrázek 96 Vyskakovací okno pro zadání emailu

```
<p>Je nutné zadat váš školní E-mail.</p>
<input type="text" class="mb-3"  placeholder="E-mail"
@bind-value="@emailVerification" @bind-value:event="oninput" @onkeyup="CheckEmail">
```

Obrázek 97 Získání emailu ze vstupu

```
private void CheckEmail() {
    if(group.Subject.Created != emailVerification) {
        turnOffBtnContinue = true;
        return;
    }
    turnOffBtnContinue = false;
```

Obrázek 98 Kontrola emailové adresy

Po úspěšné zadání emailové adresy dojde k odpočtu 15 vteřin. Čas slouží pro přečtení popsané skutečnosti, která nastane při mazání struktury. Po odpočtu lze stisknout tlačítko smazat, což vyvolá metodu DeleteSubject, která se stará o smazání struktury a skupiny, které nastane pomocí požadavku DELETE.

```
private async void DeleteSubject() {
    await Http.DeleteAsync
    ($"{MyNavigationManager.BaseUri}api/predmet/" + group.predmet.Id);
    upozorneni.ShowSuccess("Struktura byla smazaná.", "Úspěch!");
    await Task.Delay(1000);
    MyNavigationManager.NavigateTo("/tab");
}
```

Obrázek 99 Metoda vykonávající smazání struktury

8.12.4 Změna viditelnosti struktury

Slouží k úpravě viditelnosti struktury pro ostatní učitelé. Je-li struktura privátní, může jí překopírovat pouze zakladatel v jiných MS týmech. Pokud je struktura veřejná může jí překopírovat kterýkoliv učitel. Po stisku tlačítka pro změnu viditelnosti zavolá metodu „ChangeVisibility“.

```
private async void ChangeVisibility(bool IsPrivate) {
    turnOffBtns = true;
    group.Subject.Private = IsPrivate;
    await Http.PutAsJsonAsync
    ($"{MyNavigationManager.BaseUri}api/predmet/visibility", group.Subject);
    notice.ShowSuccess("Viditelnost předmětu byla změněna.", "Úspěch!");
    turnOffBtns = false;
}
```

Obrázek 100 Metoda starající se o změnu viditelnosti struktury

Metoda zjištěnému modelu, kde je uložena skupina a navázaná struktura změní viditelnost podle parametru, které přijímá od tlačítka. Změněný model odešle pomocí požadavku PUT, který pak v kontrolérů vyvolá metodu v SubjectManager, který vykoná následně změnu v DB.

```
public async Task ChangeVisibilitySubject(Subject subject)
{
    _dbContext.Entry(subject).State = EntityState.Modified;
    await _dbContext.SaveChangesAsync();
}
```

Obrázek 101 Metoda v rozhraní pro změnu viditelnosti struktury

9 ZÁVĚR

V této práci byla vytvořena aplikace pro MS Teams včetně databáze. Aplikace úspěšně rozpozná učitele, studenta či správce sítě. Podle role nastaví oprávnění k přístupu k funkcím aplikace.

Učitel může vytvořit strukturu či použít (překopírovat) existující veřejné struktury vytvořené jinými učiteli či vlastní vytvořené struktury v různých MS týmech. Učitelovi je umožněno spravovat jednotlivé kapitoly (CRUD) i jeho strukturu a skupinu. Spravovat může i jednotlivé studenty připojené ve skupině (povolení jednotlivých kapitol, zobrazení progresu v plnění kapitol atd..).

Studentovi je umožněno co nejvíce samostatně procházet kapitoly a plnit jednotlivá zadání podle svého tempa. Po splnění a korekci zadání učitelem jsou nové kapitoly zpřístupněny podle splněných prerekvizit. Náhledy kapitol jsou jasně grafiky zpracované tak, aby bylo poznat status kapitoly (odemčený, uzavřený, splněný – úspěšně x neúspěšně) a podle toho se řídí přístup k dané kapitole.

Co by mohlo být do budoucna přidáno je možnost přidání více zadání pod jednu kapitolu, která by student musel splnit. Také by bylo možné automatizovat kontrolu některých určitých typů zadání a díky tomu automaticky označit kapitolu jako splněnou nebo nesplněnou, a podle toho odemknout další kapitoly. Další možností by bylo nahradit zobrazení kapitol v seznamu za zobrazení v tzv. „pavouku“. Z pohledu administrátora by mohlo být vytvořeno zobrazení všech vytvořených skupin včetně jejich struktur.

10 SEZNAM OBRÁZKŮ A TABULEK

11 POUŽITÁ LITERATURA

[1] Proprietární software. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2021 [cit. 2022-08-29]. Dostupné z: https://cs.wikipedia.org/wiki/propriet%c3%a1rn%c3%ad_software

[2] LATHAM, Luke, 18-F-cali, Fabio SCOPEL, et al. *ASP.NET Core Blazor* [online]. 22.08.2022, **2022** [cit. 2022-10-01]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-6.0

[3] KOĎOUSKOVÁ, Barbora, 18-F-cali, Fabio SCOPEL, et al. *ARCHITEKTURA MVC: DEFINICE, STRUKTURA, FRAMEWORKY* [online]. 2021, 13.04.2021, **2021** [cit. 2022-10-01]. Dostupné z: https://www.rascasone.com/cs/blog/architektura-mvc-struktura-frameworky

[4] , Honza. *Git a GitHub* [online]. [cit. 2023-01-16]. Dostupné z: https://junior.guru/handbook/git/

[5] JUN, Šimon. Proč je Figma dar z nebes?. *Design* [online]. 8. 5. 2022, **2022** [cit. 2022-08-30]. Dostupné z: https://www.simonjun.cz/blog/proc-je-figma-dar-z-nebes

[6] JUN, Šimon. 28 nejlepších pluginů do Figmy. *Nástroje* [online]. **2022** [cit. 2022-08-30]. Dostupné z: https://www.simonjun.cz/blog/nejlepsi-pluginy-do-figmy

[7] DURANT, Brook a Niveditha NARVA. *Fluent UI Web Components Overview* [online]. 2021, 11/01/2021 [cit. 2022-10-02]. Dostupné z: https://learn.microsoft.com/en-us/fluent-ui/web-components/#what-are-web-components

[8] AFZAL KHAN, Mohammad. [online]. 2021, 11. února 2021 [cit. 2022-10-02]. Dostupné z: https://www.jadeglobal.com/blog/fluent-ui

[9] ZOLA, Andrew. Bootstrap: What is Bootstrap? [online]. 2022 [cit. 2023-02-15]. Dostupné z: https://www.techtarget.com/whatis/definition/bootstrap

[10] , angelgolfer-ms, Danielabom, lilyolason, et al., , Linda-Editor, ed. Overview of Microsoft Graph [online]. 2022, 06/15/2022 [cit. 2023-01-21]. Dostupné z: https://learn.microsoft.com/en-us/graph/overview

[11] *DbContext in Entity Framework 6* [online]. [cit. 2022-12-12]. Dostupné z: https://www.entityframeworktutorial.net/entityframework6/dbcontext.aspx

[12] *The Entity Framework Core DbContext* [online]. [cit. 2022-12-12]. Dostupné z: https://www.learnentityframeworkcore.com/dbcontext

[13] *JavaScript sessionStorage* [online]. [cit. 2022-12-15]. Dostupné z: https://www.javascripttutorial.net/web-apis/javascript-sessionstorage/

[14] VIVEK, Kumar. *Co je Power Apps?* [online]. 2022, 09.08.2022 [cit. 2022-08-27]. Dostupné z: https://docs.microsoft.com/cs-cz/power-apps/powerapps-overview

[15] HERBERT, David. *What is React.js?* [online]. June 27, 2022 [cit. 2022-12-15]. Dostupné z: https://blog.hubspot.com/website/react-js

[16] *Managing Connection Strings in Entity Framework Core* [online]. [cit. 2022-12-12]. Dostupné z: https://www.learnentityframeworkcore.com/connection-strings

[17] *is an Entity in Entity Framework?* [online]. [cit. 2022-12-14]. Dostupné z: https://www.entityframeworktutorial.net/basics/entity-in-entityframework.aspx#reference-navigation-property

[18] Fluent API v Entity Framework Core [online]. [cit. 2023-01-24]. Dostupné z: https://www.entityframeworktutorial.net/efcore/fluent-api-in-entity-framework-core.aspx

[19] *The Fluent API OnDelete Method* [online]. [cit. 2022-12-14]. Dostupné z: https://www.learnentityframeworkcore.com/configuration/fluent-api/ondelete-method

[20] *DbSet in Entity Framework 6* [online]. [cit. 2022-12-14]. Dostupné z: https://www.entityframeworktutorial.net/entityframework6/dbset.aspx

[21] .ruleFor [online]. [cit. 2023-03-10]. Dostupné z: https://fluentvalidation-ts.alexpotter.dev/docs/api/core/rulefor

[22] VICENTE, VANN. What Is a WYSIWYG Editor? [online]. 2021, OCT 8, 2021 [cit. 2023-03-10]. Dostupné z: https://www.howtogeek.com/752396/what-is-a-wysiwyg-editor/

12 ZDROJE OBRÁZKŮ

[1] *Vyukovy-pavouk: Maturitní práce* [online]. In: . [cit. 2023-01-20]. Dostupné z: https://github.com/jirka88/Vyukovy-pavouk

[2] *Design systems: Get started faster with UI kits, wireframe templates, and more*. [online]. [cit. 2022-12-29]. Dostupné z: https://www.figma.com/community/category/design-systems/plugins

[3] *Vytvářejte aplikace rychleji – s méně zdroji: Snižte své náklady na vývoj a zvládněte toho více s méně prostředky. Umožněte všem rychle vytvářet a sdílet aplikace s minimem kódu pomocí Microsoft Power Apps.* [online]. [cit. 2022-12-31]. Dostupné z: https://powerapps.microsoft.com/cs-cz/

[4] Ceny Power Apps: Podívejte se na standardní plány, náklady a dostupnost a začněte pracovat s obchodními aplikacemi. In: *Powerapps.microsoft.com* [online]. 2022 [cit. 2022-12-29]. Dostupné z: https://powerapps.microsoft.com/cs-cz/pricing/?cmpid=powerappsweb-header

13 VYUŽITÉ KNIHOVNY

[1] *Microsoft.EntityFrameworkCore* [online]. [cit. 2022-12-29]. Dostupné z: https://www.nuget.org/packages/Microsoft.EntityFrameworkCore

[2] *Microsoft.EntityFrameworkCore.SqlServer* [online]. [cit. 2022-12-29]. Dostupné z: https://www.nuget.org/packages/Microsoft.EntityFrameworkCore.SqlServer

[3] *Microsoft.EntityFrameworkCore.Tools* [online]. [cit. 2022-12-29]. Dostupné z: https://www.nuget.org/packages/Microsoft.EntityFrameworkCore.Tools

[4] *Newtonsoft.Json* [online]. [cit. 2022-12-29]. Dostupné z: https://www.nuget.org/packages/Newtonsoft.Json/

[5] *Solutaris.InfoWARE.ProtectedBrowserStorage* [online]. [cit. 2022-12-29]. Dostupné z: https://www.nuget.org/packages/Solutaris.InfoWARE.ProtectedBrowserStorage

[6] *Blazored.FluentValidation* [online]. [cit. 2023-01-02]. Dostupné z: https://www.nuget.org/packages/Blazored.FluentValidation/

[7] *Blazored.TextEditor* [online]. [cit. 2022-12-29]. Dostupné z: https://www.nuget.org/packages/Blazored.TextEditor

[8] *Microsoft.Fast.Components.FluentUI* [online]. [cit. 2023-01-05]. Dostupné z: https://www.nuget.org/packages/Microsoft.Fast.Components.FluentUI/

[9] *Microsoft.TeamsFx* [online]. [cit. 2023-01-05]. Dostupné z: https://www.nuget.org/packages/Microsoft.TeamsFx

[10] *Blazored.Toast* [online]. [cit. 2023-01-05]. Dostupné z: https://www.nuget.org/packages/Blazored.Toast

[11] *Microsoft.Graph* [online]. [cit. 2023-01-05]. Dostupné z: https://www.nuget.org/packages/Microsoft.Graph/5.0.0-rc.1

[12] *Microsoft.AspNetCore.Mvc.NewtonsoftJson* [online]. [cit. 2023-01-05]. Dostupné z: https://www.nuget.org/packages/Microsoft.AspNetCore.Mvc.NewtonsoftJson

[13] *Microsoft.AspNetCore.Authentication.JwtBearer* [online]. [cit. 2023-01-05]. Dostupné z: https://www.nuget.org/packages/Microsoft.AspNetCore.Authentication.JwtBearer

14 VYUŽITÉ GRAFICKÉ MATERIÁLY

[1] Settings SVG Vector [online]. [cit. 2023-01-04]. Dostupné z: https://www.svgrepo.com/svg/151141/settings

[2] No Data free icon [online]. [cit. 2023-01-04]. Dostupné z: https://www.flaticon.com/free-icon/no-data_7466139?term=no+data&page=1&position=2&origin=tag&related_id=7466139

[3] Face Error Free Icon [online]. [cit. 2023-01-04]. Dostupné z: https://www.onlinewebfonts.com/icon/376288

[4] Delete Free Icon [online]. [cit. 2023-01-04]. Dostupné z: https://www.onlinewebfonts.com/icon/315340

[5] X SVG Vector [online]. [cit. 2023-01-04]. Dostupné z: https://www.svgrepo.com/svg/438260/x

[6] Add SVG Vector [online]. [cit. 2023-01-04]. Dostupné z: https://www.svgrepo.com/svg/438268/add

[7] Information Round SVG Vector [online]. [cit. 2023-01-04]. Dostupné z: https://www.svgrepo.com/svg/438633/information-round

[8] Disabled SVG Vector [online]. [cit. 2023-01-04]. Dostupné z: https://www.svgrepo.com/svg/79729/disabled

[9] Hand Shake free icon [online]. [cit. 2023-01-04]. Dostupné z: https://www.flaticon.com/free-icon/hand-shake_493808

[10] Oops! 404 Error with a broken robot Flat Illustrations [online]. [cit. 2023-01-04]. Dostupné z: https://storyset.com/illustration/oops-404-error-with-a-broken-robot/rafiki

[11] Pen SVG Vector [online]. [cit. 2023-01-04]. Dostupné z: https://www.svgrepo.com/svg/433895/pen

[12] Success Free Icon [online]. [cit. 2023-01-04]. Dostupné z: https://www.onlinewebfonts.com/icon/395852

[13] List SVG Vector [online]. [cit. 2023-01-04]. Dostupné z: https://www.svgrepo.com/svg/310963/list

15 ZDROJE POUŽITÉ PŘI VÝVOJI

[1] SPASOJEVIC, Marinko. *Migrations and Seed Data With Entity Framework Core* [online]. 2022, 7.8.2022 [cit. 2023-01-04]. Dostupné z: https://code-maze.com/migrations-and-seed-data-efcore/

[2] *How to unapply a migration in ASP.NET Core with EF Core* [online]. [cit. 2023-01-04]. Dostupné z: https://stackoverflow.com/questions/38192450/how-to-unapply-a-migration-in-asp-net-core-with-ef-core

[3] , gavilanch3. *Blazor - Master-Detail - Adding, Editing and Deleting Related Entities - Entity Framework Core* [online]. [cit. 2023-01-04]. Dostupné z: https://www.youtube.com/watch?v=j1mhhYPdIco&t=104s

[4] *Config connection string in .net core 6* [online]. [cit. 2023-01-04]. Dostupné z: https://stackoverflow.com/questions/68980778/config-connection-string-in-net-core-6

[5] Graph Explorer [online]. [cit. 2023-01-29]. Dostupné z: https://developer.microsoft.com/en-us/graph/graph-explorerhttps://developer.microsoft.com/en-us/graph/graph-explorer

[6] DeepL Překladač [online]. [cit. 2023-02-14]. Dostupné z: https://www.deepl.com/translator

[7] MOHANTY, Amit. CRUD Operations Using Blazor, .Net 6.0, Entity Framework Core [online]. In: . 2022 [cit. 2023-02-18]. Dostupné z: https://www.c-sharpcorner.com/article/crud-operations-using-blazor-net-6-0-entity-framework-core/

[8] Learn Entity Framework Core: Your guide to using the latest version of Microsoft's Object Relational Mapper. Learnentityframeworkcore [online]. [cit. 2023-02-18]. Dostupné z: https://www.learnentityframeworkcore.com/

[9] NavigationManager - Get current URL in a Blazor component [online]. In: . 2019 [cit. 2023-02-18]. Dostupné z: https://stackoverflow.com/questions/50102726/navigationmanager-get-current-url-in-a-blazor-component

[10] ASP.NET Core REST API DbContext [online]. [cit. 2023-02-18]. Dostupné z: https://www.pragimtech.com/blog/blazor/asp.net-core-rest-api-dbcontext/

16 SEZNAM PŘÍLOH

Příloha 1: Návod pro učitele

Příloha 2: Návod pro studenta

Příloha 3: Zdrojový kód

Příloha 4: Aplikace k distribuci