STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor: 12 – Tvorba učebních pomůcek, didaktická technologie

AI Discovery

AI Discovery

Autor: Tomáš Ulrych

Škola: Střední škola informatiky a finančních služeb,
Klatovská 200 G, 301 00 Plzeň

Kraj: Plzeňský

Konzultant: Mgr. Lucie Martínková

Rok: 2026

Prohlášení

Prohlašuji, že jsem svou práci SOČ vypracoval samostatně a použil jsem pouze prameny a literaturu uvedené v seznamu bibliografických záznamů.

Beru na vědomí, že nejpozději odevzdáním slovesné vědecké práce do veřejné soutěže Středoškolská odborná činnost, stejně jako odevzdáním jejích příloh a dalších připojených děl, např. audiovizuálních, fotografických, výtvarných, architektonických apod. (dále jen „soutěžní dílo“), dochází ke zveřejnění díla podle § 4 odst. 1 zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů (dále jen „autorský zákon“). Totéž platí pro pozdější odevzdání doplněného, změněného, upraveného nebo opraveného díla.

Beru na vědomí, že zveřejněním díla, jehož součástí je vynález, se tento vynález stává součástí stavu techniky podle § 5 odst. 1, 2 zákona č. 527/1990 Sb., o vynálezech, průmyslových vzorech a zlepšovacích návrzích, ve znění pozdějších předpisů (dále jen „patentový zákon“), což zakládá překážku pro udělení patentu podle § 3 odst. 1 patentového zákona.

Beru na vědomí, že vyhlašovatel soutěže je podle § 61 odst. 1 autorského zákona per analogiam oprávněn užít soutěžní dílo pro účely zajištění průběhu soutěže, zejména k zajištění transparentnosti soutěže a veřejnosti obhajob soutěžních prací. V odůvodněném rozsahu je tedy vyhlašovatel po dobu účasti autora v soutěži oprávněn zejména:

- zhotovovat rozmnoženiny díla, je-li to nezbytné k seznámení účastníků soutěže, porotců nebo veřejnosti se soutěžní prací;
- zapůjčit originál nebo rozmnoženinu díla účastníkům soutěže, porotcům nebo veřejnosti, přitom dbá na bezpečné nakládání s dílem;
- vystavovat originál nebo rozmnoženinu díla v průběhu soutěžních přehlídek a doprovodných akcí;
- sdělovat dílo veřejnosti v nehmotné podobě, a to především počítačovou nebo obdobnou sítí.

Dále prohlašuji, že při tvorbě této práce jsem nepoužil nástroje AI.

V Plzni dne _____

_____

Tomáš Ulrych

Anotace

Práce se zabývá návrhem a implementací webové aplikace AI Discovery pro podporu a řízení soutěží projektového dne AiDiscovery na základních a středních školách. Cílem bylo vytvořit nástroj nahrazující dosavadní neefektivní organizaci soutěží založenou na papírových formulářích a tabulkových procesorech. Aplikace byla vytvořena pomocí technologií ASP.NET Core, Blazor Server a Entity Framework Core. Klíčovou vlastností je synchronizace hodnocení v reálném čase mezi všemi účastníky prostřednictvím knihovny SignalR. Řešení umožňuje správu soutěží a týmů, automatický výpočet průměrných hodnocení a grafickou vizualizaci výsledků. Aplikace byla úspěšně ověřena v praxi na několika základních a středních školách v Plzeňském a Moravskoslezském kraji.

Klíčová slova: Blazor Server; Entity Framework Core; SignalR; webová aplikace; real-time komunikace

Annotation

This paper deals with the design and implementation of the AI Discovery web application for supporting and managing competitions during the AiDiscovery project day at primary and secondary schools. The goal was to create a tool replacing the previously inefficient methods of competition management based on paper forms and spreadsheets. The application was developed using ASP.NET Core, Blazor Server, and Entity Framework Core technologies. A key feature is real-time score synchronization between all participants through the SignalR library. The solution enables competition and team management, automatic calculation of average scores, and graphical visualization of results. The application was successfully tested in practice at several schools in the Plzeň and Moravian-Silesian regions.

Keywords: Blazor Server; Entity Framework Core; SignalR; web application; real-time communication

Obsah

Úvod

V současné digitální éře máme nové možnosti pro organizaci a řízení soutěží. Tradiční způsoby správy soutěží, které se spoléhaly na papírové formuláře a manuální zpracování výsledků, již nestačí požadavkům na rychlost, přesnost a transparentnost, které účastníci očekávají. Využití nástrojů jako je Excel a podobných se také neosvědčilo.

Tato práce se zabývá návrhem a implementací webové aplikace AiDiscovery. Aplikace představuje moderní řešení pro správu soutěží s průběžným bodováním v reálném čase. Aplikace je navržena tak, aby umožnila efektivní ovládání týmů, kol soutěže a poskytovala okamžité zpětné vazby všem účastníkům prostřednictvím webových technologií.

Cíle práce

Hlavním cílem této práce je navrhnout a implementovat robustní webovou aplikaci založenou na moderní technologii Blazor Server pro projektový den AiDiscovery. Aplikace musí splňovat následující klíčové požadavky:

Prvním cílem je vytvoření kompletního systému pro správu soutěží, který umožní mentorům jednoduše zakládat nové soutěže, přidávat týmy a řídit průběh jednotlivých kol. Systém musí být dostatečně flexibilní, aby podporoval různé typy soutěží a zároveň dostatečně intuitivní pro snadné používání.

Druhým významným cílem je implementace průběžného bodování v reálném čase pomocí technologie SignalR. Toto řešení zajistí, že všichni mentoři budou mít okamžitý přehled o aktuálním stavu soutěže bez nutnosti manuálního obnovování stránek.

Třetím cílem je vytvoření editoru skóre, který umožní mentorům zadávat hodnocení v různých kategoriích a systém bude automaticky vypočítávat průměrné hodnoty. Tento editor musí být uživatelsky přívětivý a poskytovat okamžitou zpětnou vazbu.

Posledním cílem je ověření funkčnosti celého řešení prostřednictvím testovacích scénářů, které pokryjí všechny hlavní funkcionality aplikace od založení soutěže až po zobrazení finálních výsledků.

Postup řešení

Práce je strukturována do několika logických celků, které na sebe postupně navazují. Teoretická část poskytuje základy pro pochopení použitých technologií a principů. Začínáme přehledem webových architektur v .NET, kde vysvětlím rozdíly mezi různými přístupy a zdůvodním výběr konkrétních technologií aplikace.

Následuje podrobné seznámení s Entity Framework Core a principy práce s datovou vrstvou, včetně návrhových vzorů pro zajištění konzistentních dat při víceuživatelských přístupech. Teoretickou část uzavírá kapitola o principech tvorby interaktivních uživatelských rozhraní v Blazoru.

Aplikační část pak představuje praktickou implementaci zmíněných řešení, začínající návrhem celkové architektury aplikace a pokračující implementací backendu a frontendu.

Kapitola 1

Teoretická část

Tato kapitola vysvětluje technologie použité při vývoji aplikace AiDiscovery.

1.1 Blazor

Blazor je moderní framework od společnosti Microsoft, který je součástí platformy ASP.NET Core a slouží k tvorbě interaktivních webových rozhraní [2]. Jeho zásadní inovací je, že umožňuje vývojářům psát logiku aplikací v jazyce C# namísto tradičního JavaScriptu. Protože je Blazor postaven na moderním .NETu, plně využívá jeho multiplatformní podporu. Framework je vyvíjen jako open-source a staví na dvou přístupech (Server a WebAssembly), které jsou od verze .NET 8 plně integrovány do sjednoceného modelu „Blazor Web App“ [2].

1.1.1 Proč Blazor pro AiDiscovery

Pro aplikaci AiDiscovery jsem zvolil Blazor z několika důvodů. Klíčovým faktorem je integrace s knihovnou SignalR, která zajišťuje komunikaci v reálném čase a umožnuje synchronizovanou spolupráci více uživatelů najednou (blíže popsáno v kapitole 1.3). Další významnou výhodou režimu Blazor Server je možnost přímé interakce s datovou vrstvou přes Entity Framework Core. To eliminuje nutnost vytvářet API rozhraní a zefektivňuje vývoj. O správu uživatelů a zabezpečení se pak stará knihovna ASP.NET Core Identity [5].

1.1.2 Dependency Injection

Dependency Injection je návrhový vzor. Řeší problém propojení jednotlivých částí aplikace [1]. Princip spočívá v tom, že místo toho, aby si objekt sám vytvářel závislosti, které potřebuje, jsou mu tyto objekty předány zvenčí.

Představme si to na příkladu: služba pro správu soutěží potřebuje přístup k databázi a k systému pro odesílání real-time zpráv. Místo toho, aby si sama vytvořila připojení k databázi,

jsou jí tyto závislosti „vloženy“ při vytvoření. Výhodou je, že mohu snadno vyměnit skutečnou databázi za testovací, aniž bychom museli měnit kód služby.

V aplikaci AiDiscovery používám Dependency Injection pro předání databázového kontextu a rozhraní `IHubContext` (pro přístup k SignalR) do služby CompetitionService.

1.2 Blazor Server a interaktivní renderovací modely

Blazor je technologie od společnosti Microsoft, která umožňuje vývojářům vytvářet interaktivní webové aplikace v jazyce *C#* namísto tradičního JavaScriptu [2]. Od verze .NET 8 (v projektu je využíván .NET 9 SDK s Interactive Server render mode) se Blazor vyvinul do sjednoceného modelu nazývaného **Blazor Web App**, který umožňuje kombinovat různé režimy vykreslování v rámci jedné aplikace, čímž poskytuje maximální flexibilitu.

Dnes existují tři interaktivní varianty [2]:

- **Interactive Server:** Odpovídá původnímu Blazor Serveru; veškerá aplikační logika běží na straně serveru. Tento režim byl zvolen pro aplikaci AiDiscovery, protože vyžaduje stálé spojení se serverem pro synchronizaci dat.
- **Interactive WebAssembly:** Aplikace se zkompiluje a stáhne do prohlížeče uživatele, kde je následně vykonávána. Výhodou je možnost offline provozu a využití výkonu klienta, nevýhodou však větší objem stahovaných dat a nemožnost přímého přístupu k databázi na serveru.
- **Interactive Auto:** Nejmodernější režim, který při prvním načtení využije server pro okamžitou odezvu a následně na pozadí stáhne potřebné soubory pro přechod na WebAssembly, aby aplikace dále fungovala lokálně u klienta.

1.2.1 Jak Blazor Server funguje

V režimu Blazor Server běží celá aplikace na serveru. Komunikace probíhá pomocí trvalého spojení přes knihovnu **SignalR** [4].

Když uživatel provede akci, například klikne na tlačítko, informace se odešle na server přes SignalR spojení. Server akci zpracuje, případně uloží data do databáze, a vypočítá změny v uživatelském rozhraní. Zpět do prohlížeče se odesílají pouze tyto změny, nikoliv celá stránka, což zajišťuje velmi rychlou odezvu aplikace.

1.2.2 Výhody Blazor Server pro AiDiscovery

Hlavním důvodem pro volbu režimu Interactive Server v aplikaci AiDiscovery je skutečnost, že trvalé spojení SignalR je přímou součástí jeho architektury, což značně usnadňuje implementaci prvků pracujících v reálném čase [2]. Tento model navíc umožňuje komponentám přímou

interakci s databází přes Entity Framework Core bez nutnosti vytváření a zabezpečování samostatného rozhraní API [3]. Díky tomu, že veškerá výpočetní logika zůstává na straně serveru, stahuje prohlížeč uživatele pouze minimální množství dat. Dalším významným přínosem je bezpečnost, jelikož citlivá business logika nikdy neopouští serverové prostředí, a je tak chráněna před neoprávněnými zásahy zvenčí. V neposlední řadě je velkou výhodou možnost psát celou aplikaci jednotně v jazyce *C#*. To vyhovuje vývojářům, kteří preferují tento typově bezpečný jazyk před JavaScriptem. I přesto však Blazor v případě potřeby plně podporuje integraci funkcionalit postavených na JavaScriptu prostřednictvím mechanismu JS Interop [2].

1.2.3 Blazor WebAssembly – alternativa

Interactive WebAssembly je druhý režim. Aplikace se celá zkompiluje a stáhne do prohlížeče, kde pak běží. Pokud je aplikace nakonfigurována jako Progressive Web App (PWA), může fungovat i bez připojení k internetu [2]. Nevýhodou je větší počáteční stahování (několik megabajtů) a nutnost vytvořit API pro komunikaci s databází na serveru.

Pro AiDiscovery jsem zvolil Blazor Server z praktických důvodů. Komponenty mají přímý přístup k databázi přes Entity Framework Core bez nutnosti vytvářet samostatné API. Navíc se do prohlížeče stahuje minimum dat, což urychluje první načtení aplikace.

1.3 SignalR a komunikace v reálném čase

SignalR je knihovna pro obousměrnou komunikaci mezi serverem a prohlížečem v reálném čase [4]. Umožňuje serveru aktivně posílat zprávy prohlížečům, aniž by si o ně musely samy žádat.

1.3.1 Proč potřebuji real-time komunikaci

V tradičním webovém modelu (založeném na protokolu HTTP) vždy iniciuje komunikaci klient, pošle požadavek a čeká na odpověď. Pokud se na serveru něco změní, prohlížeč se o tom dozví až při dalším požadavku, například při obnovení stránky.

Pro aplikaci AiDiscovery je tento přístup nevhodný. Když mentor zadá hodnocení týmu, ostatní účastníci by museli neustále obnovovat stránku, aby viděli aktuální výsledky. S pomocí SignalR server okamžitě informuje všechny připojené prohlížeče o změně a zobrazení se aktualizuje automaticky.

1.3.2 Koncept Hubů

Hub je serverová třída, která slouží jako centrální bod pro komunikaci [4]. Lze jej přirovnat k telefonní ústředně, kde všichni klienti jsou připojeni k hubu a hub rozesílá zprávy těm správným příjemcům.

V aplikaci AiDiscovery používám hub nazvaný ScoreHub, který zajišťuje rozesílání aktualizací skóre všem účastníkům soutěže a oznámení o zahájení nebo ukončení kola. Slouží také pro příjem hodnocení od mentorů z mobilních zařízení. Jakmile server tato data zpracuje a uloží, Hub automaticky rozešle notifikaci o nových výsledcích všem připojeným klientům.

1.3.3 Skupiny v SignalR

SignalR umožňuje organizovat připojené klienty do skupin [4]. To využívám, když nechci rozeslat zprávy všem, ale jen vybrané skupině uživatelů.

V aplikaci AiDiscovery má každá soutěž svou vlastní skupinu. Když mentor zadá hodnocení ve hře číslo 1, zpráva se pošle jen mentorům této soutěže, nikoliv mentorům ostatních soutěží. Tím zajistím, že mentoři vidí jen relevantní informace.

Když mentor otevře stránku konkrétní soutěže, automaticky se připojí do příslušné skupiny. Při opuštění stránky se ze skupiny odpojí.

1.3.4 Automatické zotavení z výpadků

SignalR dokáže automaticky řešit problémy s připojením, ale tato funkce musí být explicitně nakonfigurována pomocí metody `withAutomaticReconnect()` [4]. Pokud se po této konfiguraci spojení přeruší, například při krátkém výpadku internetu, SignalR se pokusí znovu připojit a uživatel nemusí obnovovat stránku ručně. Navíc SignalR automaticky vybírá nejvhodnější způsob komunikace podle možností prohlížeče a sítě. Preferuje WebSockets jako nejrychlejší a nejefektivnější metodu, ale pokud nejsou dostupné, přepne na alternativní způsoby [4].

1.4 Entity Framework Core a práce s databází

Entity Framework Core (zkráceně EF Core) je framework pro práci s databází, který umožňuje pracovat s daty pomocí objektů v jazyce C# místo psaní SQL dotazů [3]. Tento přístup se nazývá ORM – Object-Relational Mapping.

1.4.1 Co je ORM

ORM funguje jako překladatel mezi databázovými tabulkami a programovacími objekty [3]. Místo toho, abych psal SQL dotazy, pracuji s objekty a kolekcemi přímo v programovacím jazyce.

Výhodou tohoto přístupu je především bezpečnost, protože ORM při správném použití chrání před SQL injection útoky. Dále zvyšuje produktivitu díky menšímu množství kódu pro běžné operace s daty. Poskytuje také typovou kontrolu, kdy se chyby odhalí již při kompilaci programu, ne až za běhu. ORM umožňuje nezávislost na databázi. Lze snadno přejít na jiný databázový systém. Konkrétní využití ORM v kódu je popsáno v kapitole 2.2.2.

Transakce v Entity Framework Core

Transakce představují klíčový mechanismus pro zajištění integrity a konzistence dat [3]. Umožňují seskupit více operací do jedné jednotky, což zaručuje, že se buď úspěšně provedou všechny definované operace, nebo se v případě chyby neprovede žádná z nich.

V rámci Entity Framework Core je každé volání metody `SaveChangesAsync()` automaticky obaleno do transakce [3]. V aplikaci AiDiscovery je tohoto principu využito zejména při procesu vytváření nové soutěže, kdy musí být entita soutěže i všechny přidružené týmy vytvořeny atomicky. Pokud by během procesu došlo k chybě, transakce zajistí návrat databáze do původního stavu.

Pro správu souběžného přístupu více mentorů spoléhá aplikace na architektonickou prevenci konfliktů. Vzhledem k tomu, že je systém navržen tak, aby každý mentor v daný moment upravoval pouze své specifické hodnocení, nepředpokládají se datové konflikty.

1.4.2 DbContext

DbContext je hlavní třída pro práci s databází v EF Core [3]. Můžeme si ji představit jako „okno“ do databáze, přes které provádíme všechny operace jako je čtení, vytváření, úpravy a mazání dat.

V aplikaci AiDiscovery používám třídu `CompetitionContext`, která obsahuje přístup ke třem tabulkám: `Competitions` pro soutěže, `Teams` pro týmy a `Rounds` pro kola s hodnocením.

DbContext automaticky sleduje změny provedené na objektech. Když změníme vlastnost objektu, například název týmu, a zavoláme uložení, EF Core sám vygeneruje potřebný SQL příkaz pro aktualizaci databáze.

1.4.3 DbContextFactory v Blazor Server

Pro aplikace postavené na Blazor Server je doporučeno používat takzvanou „factory"(rozhraní `IDbContextFactory`) pro vytváření `DbContext` [2]. Důvodem je, že v Blazor Server žijí běžné služby (Scoped) po celou dobu připojení uživatele. Standardní `DbContext` není navržen pro souběžné použití (není thread-safe) a pokud by v rámci jednoho připojení proběhly dvě operace naráz (např. načítání dat a příchozí notifikace), aplikace by selhala.

Factory vytváří novou, izolovanou instanci `DbContext` pro každou databázovou operaci, čímž se předchází konfliktům. Po dokončení operace se instance automaticky uvolní z paměti.

1.4.4 SQLite databáze

Pro ukládání dat používám **SQLite** [7]. Jedná se o lehkou souborovou databázi, která nevyžaduje instalaci samostatného databázového serveru. Celá databáze je uložena v jednom souboru. To výrazně zjednodušuje nasazení aplikace i zálohování.

SQLite je vhodná především pro menší aplikace a prototypování. Pro větší nasazení s mnoha současnými uživateli by bylo možné přejít na robustnější databázový systém, jako je **PostgreSQL** nebo **SQL Server**. Díky využití EF Core by tento přechod vyžadoval pouze minimální změny v konfiguraci, bez nutnosti přepisovat logiku aplikace [3].

1.5 Datový model aplikace

Datový model popisuje strukturu dat v aplikaci, a to jaké entity (objekty) existují a jak jsou mezi sebou propojeny.

1.5.1 Entity aplikace

Pro datový model aplikace AiDiscovery jsem navrhl tři hlavní entity:

- **`Competition` (Soutěž)** – Představuje jednu instanci investiční hry. Obsahuje konfigurační údaje (název, počet týmů, počet mentorů) a stavové informace, jako je číslo aktuálně probíhajícího kola a příznak, zda je kolo uzamčeno (ukončeno).
- **`Team` (Tým)** – Reprezentuje soutěžní tým. Obsahuje název týmu a cizí klíč (Foreign Key) odkazující na soutěž, do které tým patří.
- **`Round` (Hodnocení kola)** – Uchovává hodnocení jednoho týmu v konkrétním kole. Skládá se z bodů v kategoriích:
  - `Presentation` (Prezentace) a `Idea` (Nápad) – hodnoceno vždy.
  - `Prototype` (Prototyp) – hodnoceno od 2. kola.
  - `Teamwork` (Týmová práce) – hodnoceno od 3. kola.

  Entita také obsahuje vypočítanou vlastnost `Valuation`, která automaticky určuje celkové skóre sečtením relevantních kategorií.

1.5.2 Vztahy mezi entitami

Entity jsou propojeny vztahy, které odpovídají reálné logice soutěže. Jedna soutěž obsahuje více týmů (vztah 1:n) a jeden tým má více kol s hodnocením (opět 1:n).

Tyto vztahy jsou nastaveny s kaskádovým mazáním. Když smažu soutěž, automaticky se smažou všechny její týmy a jejich hodnocení. Tím se zajistí konzistence dat a nevzniká „nepořádek“ v databázi.

1.5.3 Pravidla hodnocení podle kola

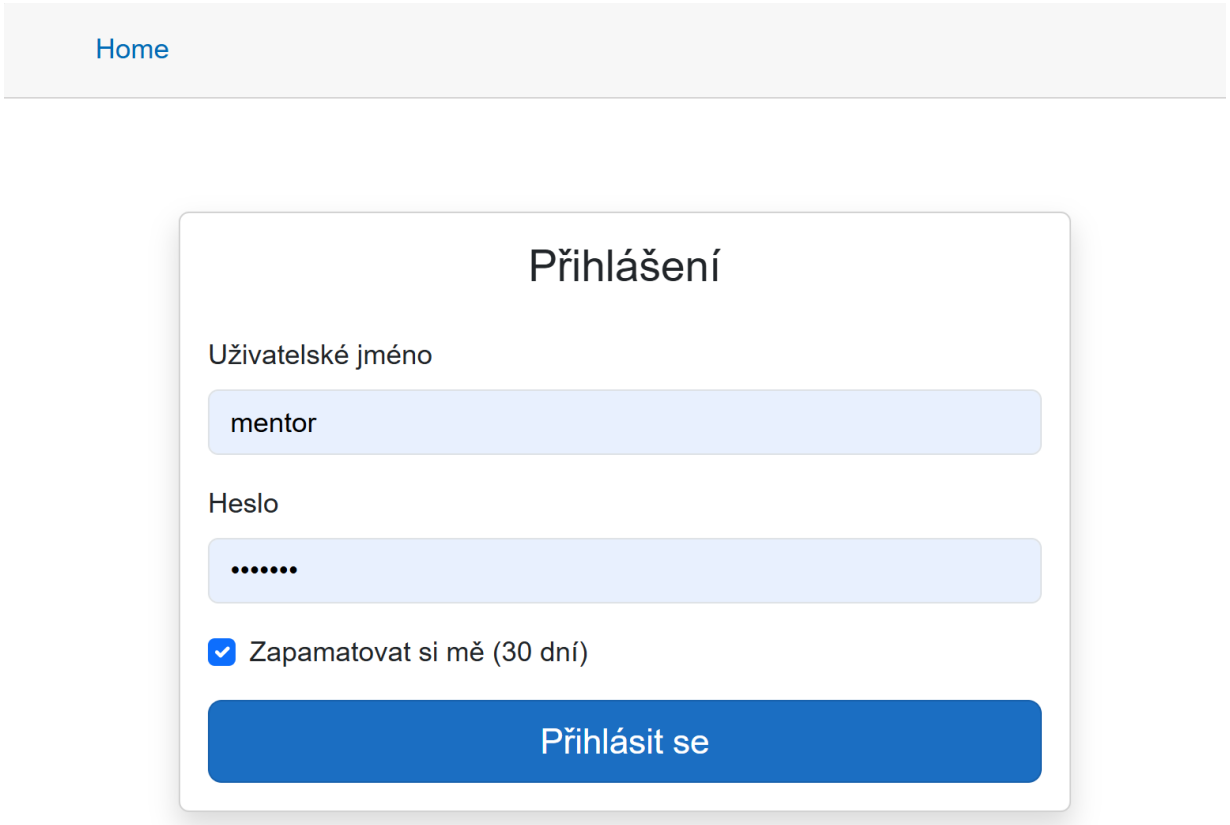
Důležitou součástí datového modelu je logika výpočtu celkového skóre, která se liší podle čísla kola.

V prvním kole se hodnotí pouze prezentace a nápad, protože týmy teprve začínají a nemají ještě funkční prototyp. Ve druhém kole se přidává hodnocení prototypu, protože týmy už mají první verzi svého produktu. Od třetího kola se hodnotí všechny čtyři kategorie včetně týmové práce.

Tato pravidla odpovídají průběhu reálné investiční hry, kde týmy postupně rozvíjejí své projekty.

1.6 ASP.NET Core Identity

ASP.NET Core Identity je komplexní framework od Microsoftu určený pro správu uživatelů [5]. Poskytuje hotové a bezpečné řešení pro registraci, přihlašování, správu hesel a ochranu osobních údajů. V aplikaci autentizaci využívám pouze jako bezpečnostní prvek pro obecné přihlášení do aplikace.



Obrázek 1.1: Přihlašovací stránka aplikace

1.6.1 Co Identity zajišťuje

Identity se stará o bezpečné ukládání hesel, která jsou kryptograficky hashována a nikdy se neukládají v čitelné podobě [5]. Dále spravuje přihlášení pomocí cookies pro udržení přihlášeného stavu mezi požadavky. Umožňuje ochranu stránek, tedy omezení přístupu jen pro přihlášené uživatele.

1.6.2 Použití v AiDiscovery

V aplikaci AiDiscovery používám Identity pro ochranu mentorských stránek. Stránky pro správu soutěží, řízení průběhu kol a editor skóre jsou přístupné pouze přihlášeným mentorům.

Stránka pro hlasování mentorů z mobilních zařízení je záměrně veřejná, aby mentoři nemuseli složitě zadávat přihlašovací údaje na telefonu během probíhající soutěže. Toto řešení upřednostňuje plynulost akce před striktní bezpečností. Spoléhám na to, že mentor v průběhu projektového dne nedává účastníkům naskenovat hlasovací QR kód, který je určený pouze pro něj.

Při prvním spuštění aplikace se automaticky vytvoří výchozí uživatelský účet s přednastavenými přihlašovacími údaji, aby byla aplikace ihned použitelná.

1.7 Blazor komponenty

Komponenta je základní stavební jednotka uživatelského rozhraní v Blazoru [2]. Celá aplikace se skládá z komponent, které lze skládat do sebe a znovu používat.

1.7.1 Struktura komponenty

Každá komponenta je uložena v souboru s příponou .razor [2]. Skládá se ze dvou částí. První částí je návrh, tedy HTML kód definující vzhled a elementy komponenty, obohacený o speciální syntaxi pro vkládání dat a logiky. Druhou částí je logika `@code`, tedy C# kód definující chování komponenty, reakce na události a práci s daty. Tyto dvě části jsou v jednom souboru, což usnadňuje orientaci.

1.7.2 Stránky aplikace

Aplikace AiDiscovery obsahuje několik hlavních stránek, z nichž každá je technicky implementována jako samostatná Blazor komponenta (s direktivem `@page`):

- **Seznam soutěží** – Zobrazuje přehled všech soutěží s možností vytvořit novou nebo otevřít existující.
- **Řízení soutěže** – Hlavní stránka pro správu průběhu soutěže, spouštění kol a sledování hodnocení v reálném čase.

- **Editor skóre** – Poskytuje kompletní tabulku pro ruční úpravu hodnocení všech týmů ve všech kolech.
- **Hlasování mentorů** – Mobilní rozhraní optimalizované pro zadávání hodnocení z telefonu.
- **Souhrn kol** – Zobrazuje grafický vývoj hodnocení, určeno pro projekci na plátno během soutěže.
- **Přihlášení** – Formulář pro autentizaci uživatele.

1.7.3 Životní cyklus komponenty

Komponenta prochází několika fázemi během své existence [2]. Při inicializaci se komponenta vytvoří, načtou se data ze serveru a připojí se k SignalR skupině. Při renderování server vypočítá změny v UI a odešle je do prohlížeče přes SignalR. Při aktualizaci, například když přijde SignalR zpráva o změně dat, se komponenta překreslí. A při zrušení, tedy při opuštění stránky, se komponenta odhlásí od SignalR skupiny a smaže se z paměti.

1.7.4 Komunikace mezi komponentami

Komponenty mezi sebou komunikují několika způsoby [2]. Pomocí parametrů rodičovská komponenta předává data potomkovi, například ID soutěže. Pomocí událostí potomek informuje rodiče o změnách, například o kliknutí na tlačítko. A pomocí sdílených služeb komponenty sdílejí data přes společnou službu registrovanou v Dependency Injection.

V AiDiscovery používám sdílenou službu CompetitionService, která uchovává informaci o aktuálně zobrazené soutěži. Díky tomu existuje například navigační lišta "zpět na soutěž".

1.8 Shrnutí použitých technologií

Pro vývoj aplikace AiDiscovery jsem použil tyto technologie: .NET jako základní framework pro webovou aplikaci [1], Blazor Server pro interaktivní uživatelské rozhraní v jazyce C# [2], SignalR pro real-time komunikaci mezi uživateli [4], Entity Framework Core pro práci s databází pomocí objektů [3], SQLite jako lehkou souborovou databázi [7], ASP.NET Core Identity pro přihlašování a správu uživatelů [5] a upravený Bootstrap pro vizuální styl [9].

Všechny tyto technologie jsou součástí ekosystému .NET od Microsoftu a vzájemně se doplňují. Díky tomu je celá aplikace napsána v jednom programovacím jazyce (C#) a používá konzistentní přístupy napříč všemi vrstvami – od databáze přes business logiku až po uživatelské rozhraní. Pouze pro vykreslování grafu si vypomáhám JavaScriptem.

Kapitola 2

Aplikační část

Tato kapitola se věnuje praktické realizaci aplikace AiDiscovery.

2.1 Návrh architektury aplikace

Správný návrh architektury je základem úspěšné aplikace. Pro AiDiscovery jsem zvolil jednoduchou, ale efektivní architekturu, která odděluje jednotlivé části aplikace.

2.1.1 Celková architektura systému

Aplikace AiDiscovery využívá moderní přístup k vývoji webových aplikací Blazor Server [2]. Architektura je rozdělena do několika vrstev, kde každá má svůj jasný účel.

Prezentační vrstva obsahuje Blazor komponenty (soubory s příponou .razor), které tvoří uživatelské rozhraní aplikace. Komponenty jsou umístěny ve složce `Components/Pages/` a zahrnují stránky pro správu soutěží, zadávání skóre a zobrazení výsledků. Tato vrstva komunikuje s uživatelem.

Servisní vrstva implementuje veškerou logiku hry. Hlavní službou je `CompetitionService`, která zajišťuje operace nad soutěžemi, týmy a koly. Služby jsou registrovány pomocí Dependency Injection a jsou zodpovědné za komunikaci s databází i za odesílání real-time notifikací přes SignalR.

Datová vrstva využívá Entity Framework Core pro komunikaci s SQLite databází [3]. Obsahuje databázový kontext `CompetitionContext`, který dědí z `IdentityDbContext` pro podporu autentizace uživatelů. K tomuto účelu navíc používáme ASP.NET Core Identity [5].

Real-time komunikační vrstva je implementována pomocí SignalR hubu `ScoreHub`, který zajišťuje obousměrnou komunikaci mezi serverem a všemi připojenými klienty [4]. Díky tomu se klientům změny v hodnocení okamžitě projeví u všech týmů v soutěži.

Komunikace mezi vrstvami probíhá prostřednictvím rozhraní (interface), což umožňuje testování a snadnou výměnu implementací. Například servisní vrstva je definována rozhraním `ICompetitionService`, které definuje dostupné operace.

2.1.2 Návrh doménového modelu

Doménový model tvoří základní stavební kámen backendové logiky aplikace a slouží k mapování reálných procesů do objektově orientované struktury. V aplikaci AiDiscovery je model navržen tak, aby reálně reflektoval princip soutěže. Vysvětlíme si konkrétněji, jak fungují entity, které jsme si již představili v kapitole 1.5.1

Velmi důležitý prvek modelu je entita `Competition`, která funguje jako hlavní objekt zastřešující celou soutěž. Tato třída obsahuje základní informace jako název soutěže, datum vytvoření, počet týmů, počet kol a počet mentorů. Důležitou vlastností je také `CurrentRoundNumber`, která sleduje aktuálně probíhající kolo, a `InputsLocked`, která určuje, zda je možné zadávat hodnocení.

Pod touto hlavní entitou se nachází třída `Team` reprezentující soutěžní tým. Každý tým má svůj název a kolekci kol (`Rounds`), ve kterých soutěží. V kontextu business logiky vystupuje tým jako jeden startup při projektovém dnu.

Entita `Round` představuje hodnocení konkrétního týmu v konkrétním kole. Obsahuje čtyři hodnotící kategorie: `Presentation`, `Prototype`, `Idea` a `Teamwork`. Všechny hodnotící atributy jsou definovány jako datový typ `double?` (nullable double). Tento typ je nezbytný pro ukládání průměrných hodnot s desetinnou přesností. Zároveň nám umožňuje pracovat s neúplnými daty v reálném čase, kdy hodnota `null` reprezentuje kategorii, která na své průměrné hodnocení teprve čeká.

Klíčovým aspektem entity `Round` je computed property `Valuation`, která implementuje logiku výpočtu celkového skóre definovanou v kapitole 1.5.3.

2.2 Implementace backendu

Backend aplikace AiDiscovery tvoří robustní základ, který zajišťuje spolehlivé ukládání dat, efektivní zpracování business logiky a real-time komunikaci s klienty.

2.2.1 Blazor konfigurace a startup

Využití frameworku .NET 9 umožňuje soustředit veškerou konfiguraci do souboru `Program.cs`, který nahrazuje dřívější rozdělení na `Program` a `Startup` [1]. V tomto bodě probíhá registrace služeb do Dependency Injection kontejneru a definice middlewarové pipeline pro zpracování HTTP požadavků.

Služba `CompetitionService` je registrována jako `Scoped`. V architektuře Blazor Server je tento životní cyklus vázán na SignalR okruh (*circuit*), což zajišťuje izolaci dat pro každého připojeného mentora [2]. Pro bezpečnou práci s databází v asynchronním prostředí Blazoru využívá aplikace `IDbContextFactory`. Tato továrna vytváří krátkodobé instance `DbContext`, čímž předchází konfliktům při souběžném přístupu z více vláken.

Systém dále integruje ASP.NET Core Identity pro zabezpečení přístupu [5]. Při prvním spuštění aplikace automaticky inicializuje databázové schéma a vytvoří výchozí uživatelský účet `mentor`. Tento mechanismus umožňuje okamžité nasazení aplikace bez nutnosti dodatečné manuální konfigurace.

2.2.2 Entity Framework Core implementace

Databázový kontext `CompetitionContext` představuje hlavní vstupní bod pro práci s databází [3]. Dědí z `IdentityDbContext`, čímž automaticky získává podporu pro tabulky uživatelů a rolí.

```
public class CompetitionContext : IdentityDbContext<ApplicationUser>
{
    public CompetitionContext(DbContextOptions<CompetitionContext> options)
        : base(options)
    {
    }

    public DbSet<Competition> Competitions { get; set; }
    public DbSet<Team> Teams { get; set; }
    public DbSet<Round> Rounds { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);

        // Konfigurace vztahu Competition -> Teams s kaskádovým mazáním
        modelBuilder.Entity<Competition>()
            .HasMany(p => p.Teams)
            .WithOne(c => c.Competition)
            .OnDelete(DeleteBehavior.Cascade);

        // Konfigurace vztahu Team -> Rounds s kaskádovým mazáním
        modelBuilder.Entity<Team>()
            .HasMany(p => p.Rounds)
            .WithOne(c => c.Team)
            .OnDelete(DeleteBehavior.Cascade);
    }
}
```

Listing 2.1: CompetitionContext - databázový kontext - využití ORM

2.2.3 Databázové migrace

Databázové migrace jsou mechanismus pro správu změn ve struktuře databáze během vývoje aplikace [3]. Migrace umožňují verzovat schéma databáze podobně, jako zdrojový kód.

Princip fungování migrací

Když vytvoříme nebo upravíme entity v našem ORM modelu, schéma databáze se automaticky nezmění. Migrace představují soubory s kódem, které popisují přesné změny potřebné pro synchronizaci databázového schématu s aktuálním stavem doménového modelu.

V Entity Framework Core vytvoříme migraci pomocí příkazu:

```
dotnet ef migrations add NazevMigrace
```

Tento příkaz vygeneruje tři soubory do složky `Migrations/`:

- `[timestamp]_NazevMigrace.cs` – obsahuje metody `Up()` a `Down()` pro aplikaci a vrácení změn
- `[timestamp]_NazevMigrace.Designer.cs` – metadata migrace
- `CompetitionContextModelSnapshot.cs` – aktuální stav celého modelu

Pro aplikaci migrace na databázi použijeme:

```
dotnet ef database update
```

2.2.4 Implementace servisní vrstvy

Servisní vrstva implementuje veškerou business logiku aplikace. Rozhraní `ICompetitionService` definuje dostupné operace.

```
1  public interface ICompetitionService
2  {
3      // Sprava stavu pro layout
4      int? CurrentCompetitionId { get; }
5      event Action? OnStateChange;
6      void SetCurrentCompetitionId(int? competitionId);
7
8      // Datove operace
9      Task<List<Competition>> GetCompetitionsAsync();
10     Task<Competition?> GetCompetitionAsync(int id);
11     Task<Competition> CreateCompetitionAsync(string name, int teams,
12         int rounds, int numberOfMentors);
13     Task DeleteCompetitionAsync(int competitionId);
14     Task UpdateCurrentRoundAsync(int competitionId, int currentRoundNumber);
15     Task UpdateSpecificScoreAsync(int roundId, string propertyName,
16         double newValue, int competitionId);
17     Task NotifyRoundSavedAsync(int competitionId);
18 }
```

Listing 2.2: ICompetitionService - rozhraní servisní vrstvy

Implementace služby `CompetitionService` využívá `IDbContextFactory` pro vytváření zmiňovaných dočasných databázových kontextů a `IHubContext<ScoreHub>` pro odesílání SignalR notifikací [4].

Důležitou metodou je `UpdateSpecificScoreAsync`, která využívá hodnocení mentorů pro dynamické nastavení hodnoty konkrétní vlastnosti entity Round.

2.2.5 SignalR Hub implementace

SignalR Hub představuje centrální bod pro real-time komunikaci a slouží jako komunikační brána mezi serverem a připojenými klienty [4]. Třída `ScoreHub` implementuje metody pro správu skupin a zpracování aktualizací skóre. Pro efektivní distribuci zpráv využívám koncept skupin, kde každá soutěž má svou vlastní izolovanou skupinu.

```
public class ScoreHub : Hub
{
    public async Task JoinCompetitionGroup(int competitionId)
    {
        string groupName = $"Competition_{competitionId}";
        await Groups.AddToGroupAsync(Context.ConnectionId, groupName);
    }

    public async Task LeaveCompetitionGroup(int competitionId)
    {
        string groupName = $"Competition_{competitionId}";
        await Groups.RemoveFromGroupAsync(Context.ConnectionId, groupName);
    }
}
```

Listing 2.3: Implementace ScoreHubu

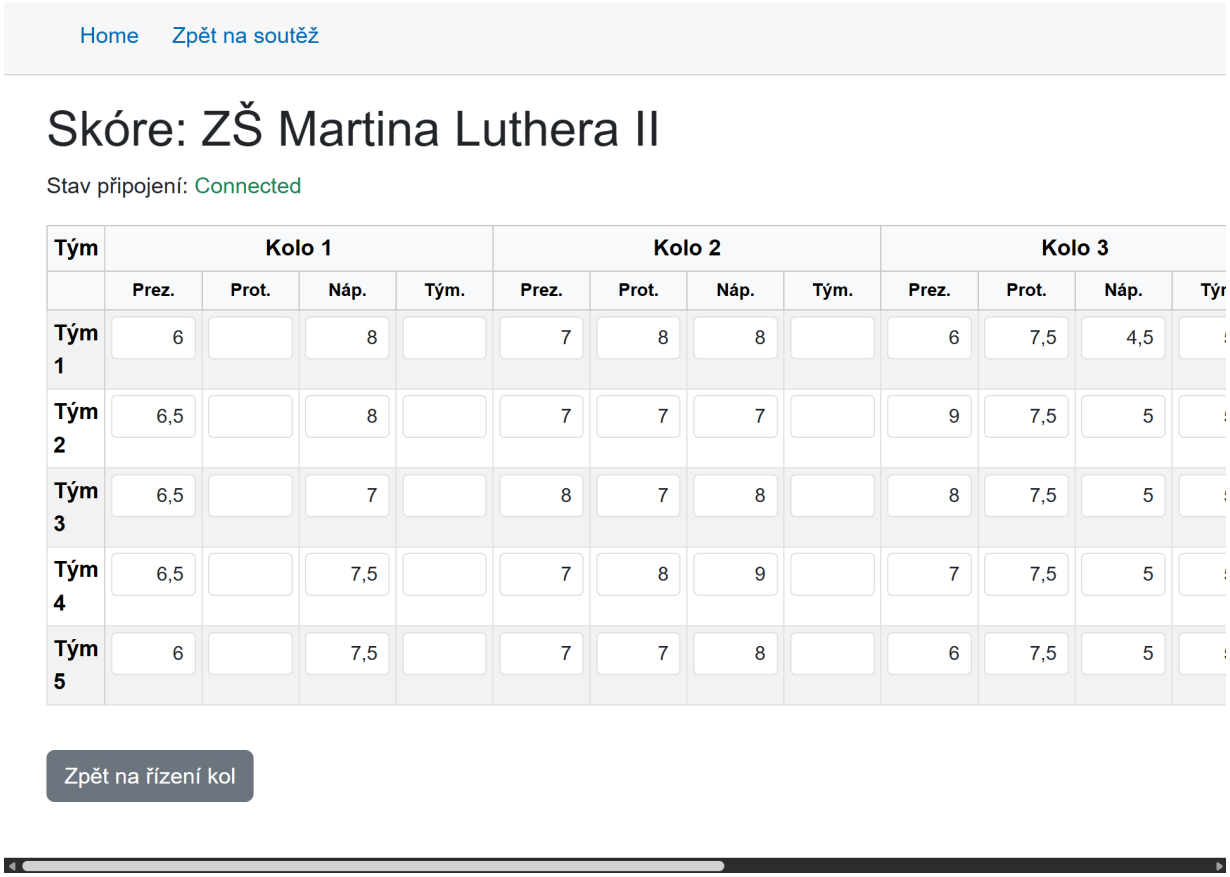
Mechanismus doručování zpráv

Na rozdíl od jednoduchého rozesílání zpráv všem připojeným zařízením (tzv. broadcast), využívá SignalR v aplikaci AiDiscovery cílené adresování [4]. Technicky se nejedná o situaci, kdy by zpráva byla odeslána všem a následně zahozena klientem. Server SignalR udržuje v paměti seznam ID připojení přiřazených k jednotlivým skupinám.

Při odeslání aktualizace server nejdříve identifikuje všechna aktivní spojení (`ConnectionId`) v dané skupině a zprávu odešle pouze těmto konkrétním adresátům. Tímto mechanismem filtrace na straně serveru dochází k výrazné minimalizaci síťového provozu, což je klíčové pro plynulý chod aplikace při zapojení většího počtu škol najednou. Servisní vrstva aplikace k tomuto účelu využívá injektované rozhraní `IHubContext<ScoreHub>`, které umožňuje odesílat notifikace přímo z obchodní logiky služby `CompetitionService`.

2.3 Implementace frontendu

Frontend aplikace využívá Blazor Server komponenty s interaktivním renderováním [2]. Komponenty kombinují HTML, CSS s C# logikou v souborech s příponou .razor.



Tým	Kolo 1				Kolo 2				Kolo 3			
	Prez.	Prot.	Náp.	Tým.	Prez.	Prot.	Náp.	Tým.	Prez.	Prot.	Náp.	Týr
Tým 1	6		8		7	8	8		6	7,5	4,5	
Tým 2	6,5		8		7	7	7		9	7,5	5	
Tým 3	6,5		7		8	7	8		8	7,5	5	
Tým 4	6,5		7,5		7	8	9		7	7,5	5	
Tým 5	6		7,5		7	7	8		6	7,5	5	

Obrázek 2.1: Kompletní editor skóre zobrazující hodnocení všech týmů ve všech kolech

2.3.1 Struktura Blazor komponent

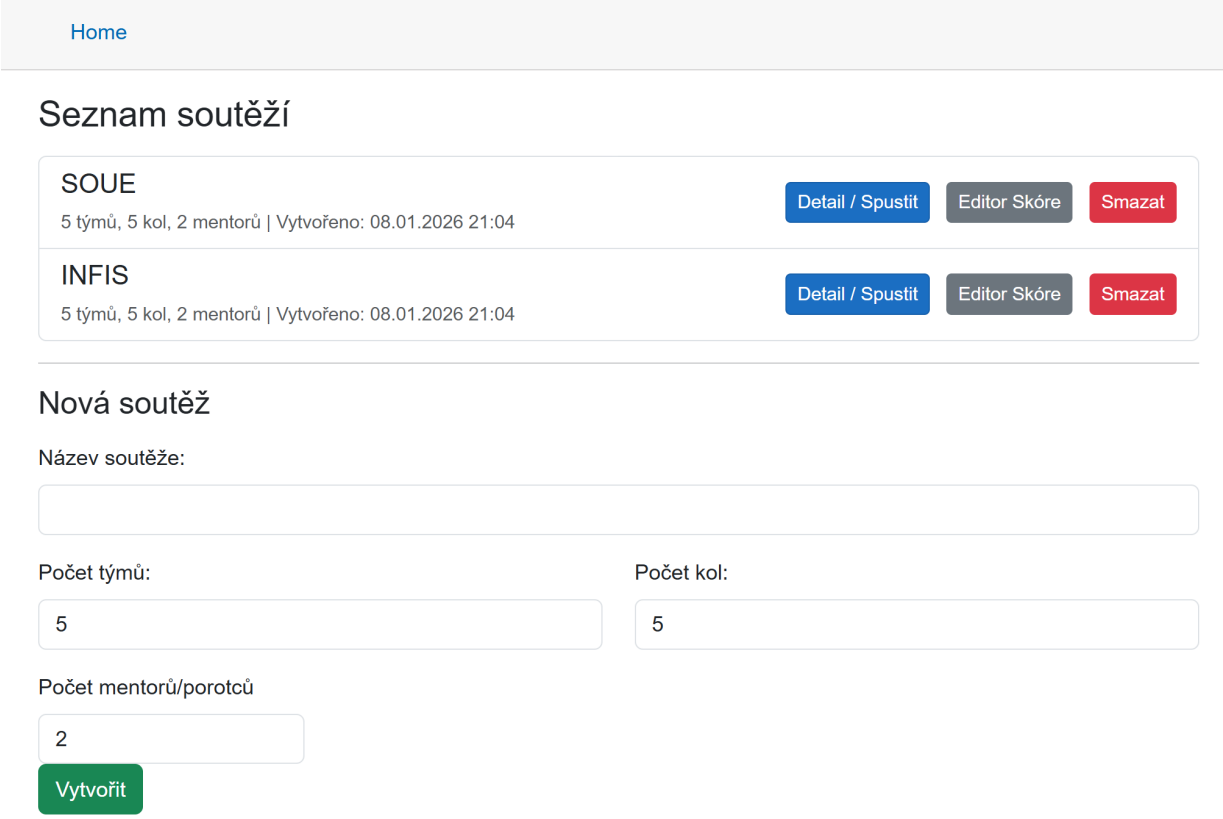
Aplikace obsahuje několik stránek:

- `Competitions.razor` - seznam soutěží a formulář pro vytvoření nové soutěže
- `CompetitionDetails.razor` - řízení průběhu soutěže a zadávání hodnocení
- `Scores.razor` - kompletní editor skóre pro všechna kola
- `MentorVoting.razor` - rozhraní pro hlasování mentorů na mobilních zařízeních
- `RoundSummary.razor` - grafické zobrazení valuace/finanční situace jednotlivých týmů

Každá komponenta implementuje rozhraní `IAsyncDisposable` pro správné uvolnění SignalR připojení při opuštění stránky.

2.3.2 Seznam soutěží a vytvoření nové soutěže

Komponenta `Competitions.razor` zobrazuje přehled všech soutěží a umožňuje vytvářet nové. Využívá interaktivní `EditForm`.



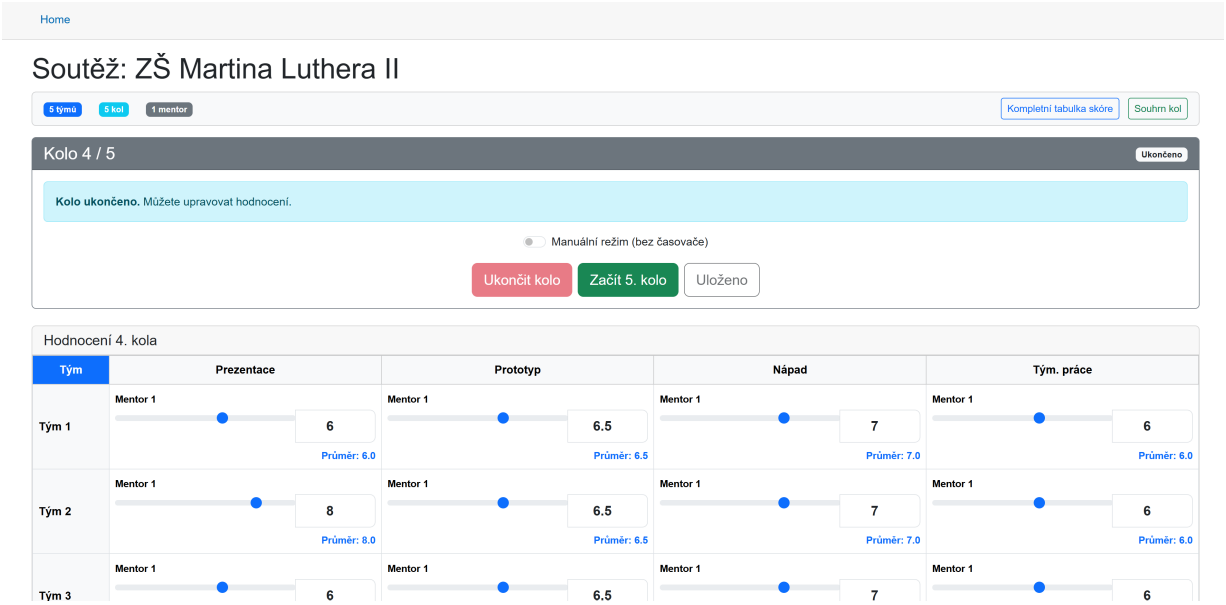
Obrázek 2.2: Seznam soutěží s formulářem pro vytvoření nové soutěže

2.3.3 Řízení průběhu soutěže

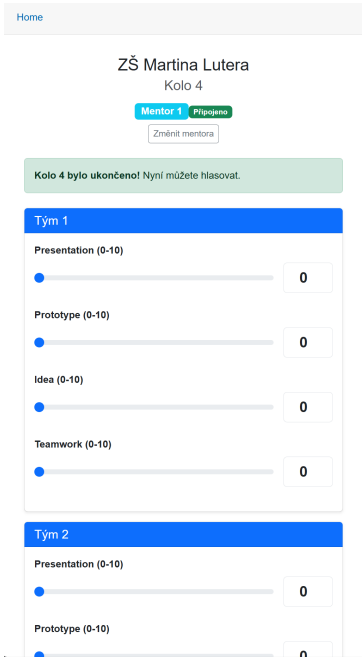
Komponenta `CompetitionDetails.razor` je nejkomplexnější částí aplikace. Umožňuje spouštět a ukončovat kola, zadávat hodnocení od mentorů a ukládat výsledky.

2.3.4 Hlasování mentorů na mobilních zařízeních

Komponenta `MentorVoting.razor` poskytuje zjednodušené rozhraní optimalizované pro mobilní zařízení. Mentor si nejprve vybere své číslo a poté může zadávat hodnocení pomocí posuvníků.



Obrázek 2.3: Řízení průběhu soutěže s tabulkou hodnocení pro aktuální kolo

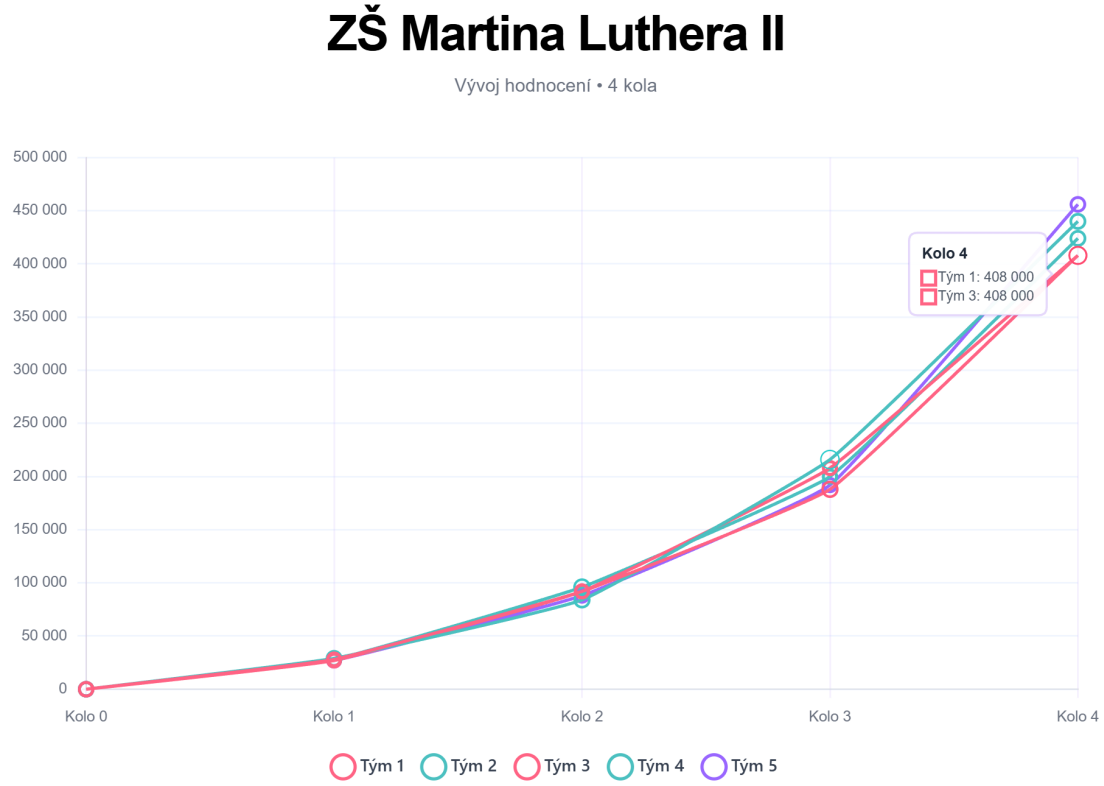


Obrázek 2.4: Grafické zobrazení vývoje hodnocení týmů v průběhu soutěže

2.3.5 Grafické zobrazení výsledků

Komponenta `RoundSummary.razor` zobrazuje vývoj hodnocení týmů pomocí interaktivního grafu. Graf se aktualizuje automaticky po uložení každého kola díky SignalR notifikacím.

Ačkoliv je celá aplikace psaná v C#, pro vykreslení grafu využívám knihovnu **Chart.js** volanou přes mechanismus JS Interop [2]. Blazor umožňuje volat JavaScriptové funkce z C# kódu pomocí rozhraní `IJSRuntime`. Komponenta po načtení dat zavolá JavaScriptovou funkci, která inicializuje graf. Při příchodu SignalR notifikace o změně dat se graf dynamicky překreslí.



Obrázek 2.5: Grafické zobrazení vývoje hodnocení týmů v průběhu soutěže

2.4 Implementace časovače pro řízení kol

Časovač automaticky řídí průběh kol soutěže pomocí dvou synchronizovaných timerů. Jeden timer zajišťuje přesné ukončení kola po předdefinované délce, druhý aktualizuje zbývající čas v rozhraní, takže mentoři vidí informaci v reálném čase bez nutnosti obnovování stránky. První kolo trvá 10 minut, zatímco všechna ostatní trvají 5 minut.

2.4.1 Architektura

Round Timer – hlavní časovač na celkovou délku kola (10 min první kolo, 5 min další). Spustí se jednou a vyvolá ukončení kola.

UI Timer – pomocný časovač, aktualizuje zobrazení zbývajícího času.

2.4.2 Synchronizace s Blazorem

Časovače běží na pozadí, proto musí být jejich obsluha přesunuta do synchronizačního kontextu daného SignalR okruhu pomocí metody `InvokeAsync()` [2]. Komponenta implementuje `IAsyncDisposable` pro správné uvolnění časovačů při opuštění stránky.

2.4.3 Manuální režim

Pro flexibilnější ovládání je k dispozici manuální režim, který vypne automatické časovače a umožní ruční řízení kol.

2.5 Autentizace a autorizace

Zabezpečení aplikace je implementováno s pomocí frameworku ASP.NET Core Identity, který poskytuje základ pro správu uživatelských identit [5]. Autentizační stav uživatele je spravován službou `AuthenticationStateProvider`, která zprostředkovává informaci o přihlášení do všech komponent pomocí parametru `Task<AuthenticationState>`.

Přístup ke stránkám s citlivými operacemi (správa soutěží, editace skóre, řízení kol) je omezen pomocí direktivy:

```
@attribute [Authorize]
```

Tento atribut zajišťuje deklarativní kontrolu přístupu. Pokud se na stránku pokusí přistoupit anonymní uživatel bez platného autentizačního cookie, komponenta AuthorizeRouteView tento stav detekuje a zobrazí sekci <NotAuthorized>, která zajistí přesměrování uživatele na přihlašovací formulář (`/login`).

2.5.1 Konfigurace Identity

Identity je nakonfigurováno v `Program.cs` se zjednodušenými požadavky na heslo vhodnými pro vývoj a testování [5]:

```
builder.Services.AddIdentity<ApplicationUser, IdentityRole>(options =>
{
    options.Password.RequireDigit = false;
    options.Password.RequireLowercase = false;
    options.Password.RequireUppercase = false;
```

```
    options.Password.RequireNonAlphanumeric = false;
    options.Password.RequiredLength = 6;
})
.AddEntityFrameworkStores<CompetitionContext>()
.AddDefaultTokenProviders();
```

Listing 2.4: Konfigurace ASP.NET Identity

Třída `ApplicationUser` dědí z `IdentityUser` a může být rozšířena o další vlastnosti specifické pro budoucí potřeby aplikace.

2.5.2 Automatické vytvoření výchozího uživatele

Při startu aplikace se automaticky vytvoří výchozí uživatel `mentor`, pokud ještě neexistuje:

```
using (var scope = app.Services.CreateScope())
{
    var userManager = scope.ServiceProvider
        .GetRequiredService<UserManager<ApplicationUser>>();

    var mentorUser = await userManager.FindByNameAsync("mentor");
    if (mentorUser == null)
    {
        mentorUser = new ApplicationUser
        {
            UserName = "mentor",
            Email = "mentor@aiolympiada.cz",
            EmailConfirmed = true
        };

        var result = await userManager.CreateAsync(mentorUser, "HESLO");
        if (result.Succeeded)
        {
            app.Logger.LogInformation("Default user 'mentor' created.");
        }
    }
}
```

Listing 2.5: Vytvoření výchozího uživatele

2.6 Vývojové prostředí a nasazení aplikace

Pro spuštění aplikace potřebujeme nainstalovat **.NET 9 SDK**, nástroj pro sestavení a běh .NET aplikací. Databáze SQLite se vytvoří automaticky při prvním spuštění, není nutná žádná instalace [7]. Jako vývojové prostředí doporučuji JetBrains Rider.

Postup spuštění je jednoduchý: stáhneme projekt příkazem `git clone`, obnovíme závislosti pomocí `dotnet restore` a spustíme aplikaci příkazem `dotnet run --project AiDiscovery`. Veškerý zdrojový kód webové aplikace AiDiscovery je dostupný na

https://github.com/TomasULR/AiDiscoveryGit.

2.6.1 Kontejnerizace pomocí Dockeru

Docker umožňuje zabalit aplikaci i se všemi závislostmi do jednoho kontejneru, který lze spustit na jakémkoliv počítači se stejným výsledkem [6].

Aplikace používá vícestupňový build. Nejdříve se aplikace zkompiluje (build), poté se připraví k publikaci a nakonec se vytvoří runtime. Díky tomu je výsledný kontejner menší a rychlejší.

Docker Compose slouží k definici a spuštění celé aplikace i s databází pomocí jediného příkazu [6].

Příkazy: `docker-compose up -d --build` spustí aplikaci, `docker-compose logs -f` zobrazí výpisy z aplikace.

2.7 Testovací scénáře

Před nasazením do provozu byla aplikace otestována na těchto scénářích: vytvoření nové soutěže, zahájení a ukončení kola, hlasování mentorů přes QR kód na mobilech, synchronizace hodnocení v reálném čase mezi více prohlížeči a přepínání mezi automatickým a manuálním režimem.

Shrnutí

Projekt již prošel fází pilotního nasazení a byl úspěšně ověřen v praxi na několika vzdělávacích institucích. Konkrétně byl projektový den AiDiscovery s aplikací AiDiscovery odehrán na 26. základní škole v Plzni, 1. základní škole v Plzni, ve třech běžích na Základní škole Martina Luthera a také na SŠPEI MTA v Ostravě. Nasazení probíhalo přímo v rámci plnohodnotného projektového dne AiDiscovery, což umožnilo pozorovat interakci žáků i mentorů s aplikací v reálném čase. Tato praktická validace potvrdila použitelnost vytvořeného řešení. Zároveň poskytla klíčovou zpětnou vazbu pro identifikaci oblastí vyžadujících zlepšení. Díky těmto pilotním akcím byla získána relevantní data o funkcionalitě systému a zkušenosti mentorů, která slouží jako podklad pro další rozvoj aplikace.

Pro účely projektového dne jsem vytvořil doplňkovou aplikaci AiDiscovery Tutorial. Ačkoliv tato aplikace není hlavním předmětem této práce, jejím úkolem je vysvětlit účastníkům principy promptování, jazykových modelů a počítačové vize. Zájemci si ji mohou vyzkoušet na adrese https://ai-experiments.cloud.nvias.org.

Summary

The AiDiscovery application has already been tested in practice at several schools. Specifically, the AiDiscovery project day took place at the 26th Elementary School in Plzeň, the 1st Elementary School in Plzeň, the Martin Luther Elementary School, and also at SŠPEI in Ostrava.

The application was used directly during the full project day. This allowed us to observe how students and mentors interact with the app in real time. This practical test confirmed that the solution works well. It also provided important feedback to identify areas that need improvement. Thanks to these events, we gathered relevant data about the system's functionality and the mentors' experience, which helps with the future development of the app.

Furthermore, for the purposes of the project day, I developed a supplementary application called AiDiscovery Tutorial. Although this application is not the primary subject of this paper, its objective is to explain the principles of prompting, large language models, and computer vision to the participants. It is available for testing at https://ai-experiments.cloud.nvias.org.

Závěr

Aplikace AiDiscovery úspěšně implementuje všechny stanovené cíle:

1. **Správa soutěží** - administrátor může vytvářet soutěže s konfigurovatelným počtem týmů, kol a mentorů
2. **Real-time aktualizace** - díky SignalR všichni účastníci vidí změny okamžitě
3. **Editor skóre** - mentoři mohou zadávat hodnocení z mobilních zařízení, průměry se automaticky vypočítávají
4. **Vizualizace výsledků** - grafické zobrazení vývoje hodnocení v průběhu soutěže
5. **Autentizace** - přístup je chráněn pomocí ASP.NET Identity

Zpětná vazba z pilotních nasazení potvrdila použitelnost aplikace v reálném prostředí a identifikovala místa pro další rozvoj. Mezi budoucí rozšíření patří například podpora více typů soutěží a rozšířené statistiky.

Všechny stanovené cíle práce byly splněny. Aplikace je připravena k dalšímu praktickému využití a případnému rozšíření.

Seznam použitých zdrojů

1. MICROSOFT. *ASP.NET Core documentation* [online]. Microsoft Docs, 2024 [cit. 2025-01-05]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/

2. MICROSOFT. *Blazor documentation* [online]. Microsoft Docs, 2024 [cit. 2025-01-05]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/blazor/

3. MICROSOFT. *Entity Framework Core documentation* [online]. Microsoft Docs, 2024 [cit. 2025-01-05]. Dostupné z: https://learn.microsoft.com/en-us/ef/core/

4. MICROSOFT. *ASP.NET Core SignalR documentation* [online]. Microsoft Docs, 2024 [cit. 2025-01-05]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/signalr/

5. MICROSOFT. *ASP.NET Core Identity documentation* [online]. Microsoft Docs, 2024 [cit. 2025-01-05]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity

6. DOCKER, INC. *Docker Documentation* [online]. Docker Docs, 2024 [cit. 2025-01-05]. Dostupné z: https://docs.docker.com/

7. SQLITE. *SQLite Documentation* [online]. SQLite, 2024 [cit. 2025-01-05]. Dostupné z: https://www.sqlite.org/docs.html

8. STACK OVERFLOW. *Stack Overflow – Where Developers Learn, Share, & Build Careers* [online]. Stack Exchange, 2024 [cit. 2025-01-05]. Dostupné z: https://stackoverflow.com/

9. BOOTSTRAP. *Bootstrap Documentation* [online]. Bootstrap Team, 2024 [cit. 2025-01-05]. Dostupné z: https://getbootstrap.com/docs/