Msc önálló laboratórium 2

ASP.NET Core alapú webalkalmazás fejlesztése

Online form, kérdőív és tesztrendszer készítése Blazor alapokon

Szerző: Papp Kristóf

Konzulens: Gincsai Gábor

Dátum: 2020. november 27.

Tartalom

[1 Bevezetés, feladatválasztás 1](#_Toc57370128)

[1.1 Követelmények 1](#_Toc57370129)

[2 Felhasznált technológiák 2](#_Toc57370130)

[2.1 .NET Core 2](#_Toc57370131)

[2.2 ASP.NET Core 2](#_Toc57370132)

[2.3 Blazor 2](#_Toc57370133)

[2.4 MongoDB 4](#_Toc57370134)

[3 Architektúra 4](#_Toc57370135)

[3.1 Rendszer architektúra 5](#_Toc57370136)

[3.2 Az adatbázis felépítése 6](#_Toc57370137)

[3.3 Frontend architektúra 7](#_Toc57370138)

[4 Megvalósítás 7](#_Toc57370139)

[4.1 Bejelentkezés 7](#_Toc57370140)

[4.2 Form létrehozása 9](#_Toc57370141)

[4.3 Saját formok megtekintése és válaszok exportálása 11](#_Toc57370142)

[4.4 Form kitöltése 12](#_Toc57370143)

[4.5 Egyéb megvalósított funkciók 13](#_Toc57370144)

[5 Összefoglalás 13](#_Toc57370145)

[5.1 Tapasztalatok 13](#_Toc57370146)

[5.2 Továbbfejlesztési lehetőségek 14](#_Toc57370147)

[Irodalomjegyzék 15](#_Toc57370148)

[Irodalomjegyzék 13](#_Toc38013122)

# Bevezetés, feladatválasztás

A vírushelyzet következtében az elmúlt időszakban jelentősen megnövekedett az igény az online elérhető szolgáltatások, azon belül is különösen a tanulást támogató és kapcsolattartó rendszerek iránt. Ez okból döntöttem amellett, hogy Msc önálló labor 2 témámként egy olyan rendszert valósítok meg, mellyel lehetőség van online kérdőívek létrehozására és kitöltésére, valamint iskolai vagy egyéb dolgozatok és tesztek lebonyolítására.

## Követelmények

Az alkalmazásban legyen lehetőség személyre szabott formok és online tesztek összeállítására, majd az azokra érkező válaszok adatbázisba való mentésére vagy exportálására. Az alkalmazás tegye lehetővé egyedi formok létrehozását, amelyehez több kérdés is tartozhat. A rendszer támogassa az alábbi kérdés típusokat:

* Szöveges válasz
* Numerikus válasz
* Egyszeres választás előre megadott opciók közül
* Többszörös választás előre megadott opciók közül

Egy kérdés létrehozásakor legyenek megadhatók helyes válaszok, ilyenkor a rendszer a form befejezése után automatikusan értékelje ki a megadott válaszokat, és jelenítse meg az elért pontszámot a kitöltő számára. A válaszadók pontszámai az adott tesztet létrehozó számára is legyenek lekérdezhetők, és a helyes válaszok aránya grafikonon is legyen megtekinthető. Az egyes formokhoz opcionálisan határidő is legyen megadható, ilyenkor az adott form csak a kezdeti és végső dátum közötti időszakban tölthető ki. A rendszer felhasználói különböző módon láthatják a jelenleg elérhető, a már kitöltött és a lejárt kérdőíveket. Ezen kívül minden formhoz tartozhasson egy kitöltési időkorlát is. Ez a form kérdéseinek megtekintésekor kezdődik, és a kitöltőknek a hátralévő időről folyamatos visszajelzést kell kapniuk.

Az alkalmazás támogassa a felhasználók kezelését, legyen lehetőség új felhasználó regisztrálására, és a formok megtekintése, illetve azok kitöltése csak bejelentkezés után legyen elérhető. Minden felhasználó tekinthesse meg az általa kitöltött formokat, de más felhasználók eredményeihez csak az adott form létrehozója férhessen hozzá. A form tulajdonosa a válaszokat legyen képes Excel formátumba exportálni és letölteni.

Az alkalmazás megvalósítása során cél a felhasználóbarát megjelenítés és a jó skálázhatóság, hiszen egy hasonló éles rendszerben akár több szár felhasználó egyidejű kiszolgálására lehet igény.

# Felhasznált technológiák

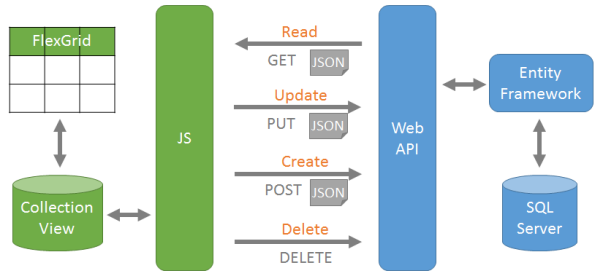
Tanulmányaim során már többször, pozitív tapasztalatokkal használtam a .Net keretrendszert, de korábban mindig csak szerver oldali funkcionalitás implementálásához, így örömmel hallottam a hírt, amikor nemrégiben megjelent a Microsoft új webes keretrendszere a Blazor. Az önálló labor feladatom fontos részét képezte ennek a technológiának a megismerése.

## .NET Core

A Microsoft által 2016-ban bemutatott .NET Core egy ingyenesen hozzáférhető és nyílt forráskódú, általános célú és multiplatform szoftverfejlesztési keretrendszer. A teljes .NET Frameworkhöz képest jelentősen átdolgozott .NET Core moduláris felépítésű, így csak az alapvető osztályokat tartalmazza, további komponensek Nuget csomagok formájában tölthetők le. A .NET nyelvfüggetlen, a C#, VB és F# mellett több mint 30 különböző nyelvet támogat.

## ASP.NET Core

Az ASP.NET Core egy modern, főleg szerver oldali, internetes alkalmazások fejlesztését támogató keretrendszer. Egyaránt támogatja a szerver oldalon renderelt weboldalak megjelenítését (Razor Pages), Single-page webalkalmazások (Blazor, Angular, React) készítését vagy bármely kliens kiszolgálását REST (Representational State Transfer) alapú WebAPI-n keresztül. Az ASP.NET egyaránt rendelkezik .NET Framework és multiplatform .NET Core implementációval.



Alapvető REST kérések

## Blazor

Az ASP.NET Core részét képező Blazor C# nyelven írt webalkalmazások készítését teszi lehetővé. Segítségével az applikáció felhasználói felületét HTML-szerű tag-ekkel, míg a UI logikát és a modell osztályokat C# kóddal adhatjuk meg egy közös, „.razor” kiterjesztésű fájlban. A Blazor a React keretrendszerhez hasonlóan komponens alapú, lehetővé teszi, hogy egy másik fájlban definiált komponenst HTML tag-ként felhasználjunk. Az jelentősen elősegíti az újrafelhasználhatóságot és külső UI komponens könyvtárak létrehozását is lehetővé teszi. Az alábbi kódrészlet egy egyszerű Blazor komponens definiálását, majd felhasználását mutatja.

<h1>@Hours : @Minutes : @Seconds</h1>

@code {

[Parameter]

public DateTime Time { get; set; }

[Parameter]

public EventCallback<string> OnEnd { get; set; }

private int Seconds { get; set; } = 10;

private int Minutes { get; set; } = 10;

private int Hours { get; set; } = 10;

private bool sent { get; set; } = false;

private Timer timer { get; set; }

protected override void OnParametersSet()

{

…

}

void StartCountdown()

{

…

}

public void Dispose()

{

…

}

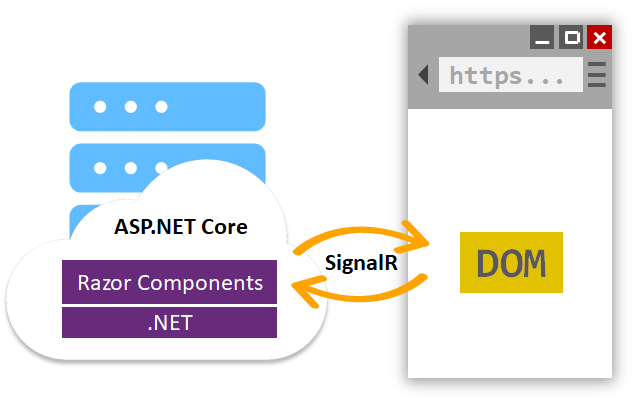
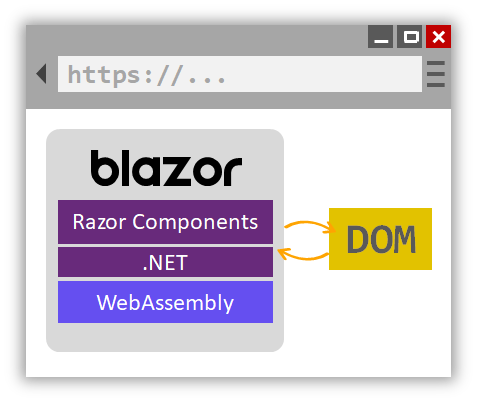
}

<Countdown OnEnd=Finish Time=@Time />

A fenti példán látszik, hogy a Blazor komponensek rendelkezhetnek bemenő paraméterekkel, melyeken keresztül adatokat kaphatnak a szülő komponensüktől (ilyen a fenti példában a Time, amely a visszaszámlálás kiindulását adja meg). A paraméterek egy speciális típusa az EventCallback, mellyel az adott komponens a szülője számára publikálhat eseményeket (a fenti kódrészletben ennek segítségével értesíti az időzítő az őt létrehozó komponenst az időzítés lejártáról).

A Blazor komponensek életciklussal rendelkeznek, mely változásairól életciklusfüggvények felüldefiniálásával értesülhetünk. Ilyen lehet például a komponens paramétereinek beállítása után egy időzítő elindítása vagy a komponens renderelésekor az adatok lekérdezése a szerverről. Utóbbihoz a Blazor keretrendszer aszinkron életcikluseseményeket definiál, így a hosszabb ideig tartó műveletek sem okozzák a UI lefagyását.

A Blazor egyaránt támogat szerver és kliens oldali renderelést is. kliens oldali renderelés esetében az általunk készített komponensek és a .NET runtime WebAssembly nyelvre fordítódik le, melyet a böngésző közvetlenül tud futtatni. Ezzel szemben a szerver oldali renderelés esetén a DOM elóállítása az ASP.NET Core szerver feladata, és azt a böngésző már HTML formában kapja meg. A hatékony és gyors frissítések érdekében a böngésző és a szerver egy folyamatos SignalR kapcsolatot tart fenn.



Kliens és szerver oldali renderelés Blazor esetében

## MongoDB

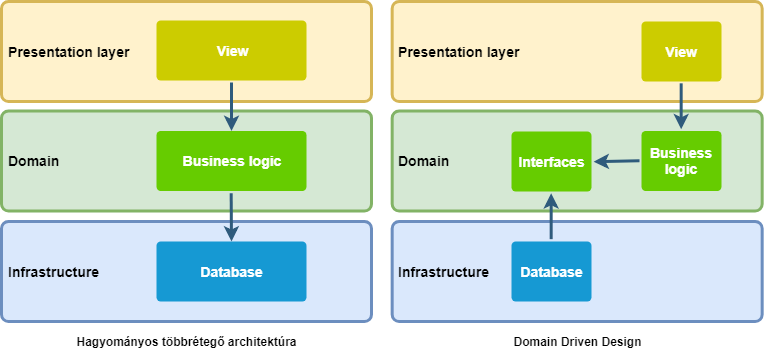
A MongoDB a manapság használt egyik legnépszerűbb általános célú NoSQL adatbáziskezelő. A tárolás alapja a dokumentum, amely lényegében egy JSON/BSON objektumnak felel meg, és kulcs-érték párokat tartalmaz. A MongoDB előnye a hagyományos relációs adatbáziskezelőkkel szemben az ingyenes elérhetőség, a jobb horizontális skálázhatóság és a kötetlen séma. Ezen tulajdonságai miatt esett a választásom a MongoDB-re, hiszen az alkalmazásomban használt formok felépítése jelentősen eltérhet (más számú és típusú kérdést, valamint választ tartalmazhatnak). Természetesen ezek az információk több táblában relációs adatbázisban is tárolhatók lennének, azonban a MongoDB előnye, hogy segítségével a magas szintű programozási nyelvekből megszokott objektumok különösebb transzformációk vagy több táblára való bontás nélkül tárolhatók. A MongoDB saját, szintén a JSON formátumon alapuló lekérdezőnyelvet használ az adatok kinyeréséhez és feldolgozásához. Számtalan keretrendszerhez és programozási nyelvhez létezik osztálykönyvtár a Mongo adatbázisok eléréséhez, többek között a .Net, a Java és a Go is támogatott.

# Architektúra

Az alkalmazás architektúráját több szinten fogom bemutatni. A tervezés során törekedtem az újrafelhasználhatóságra, és a komponensek közti laza csatolásra. Az alkalmazás egyes szintjein más-más architekturális mintákat és tervezési elveket követtem.

## Rendszer architektúra

Az alkalmazás alapvetően kliens-szerver architektúrát követ, így két fő komponensből - egy webes kliensből és egy azt kiszolgáló szerverből - épül fel. A szerveren belül elkülönül a backend üzleti logikai és az adatelérési réteg, így az alkalmazás magas szintű felépítését egy klasszikus három rétegű architektúra határozza meg. A webes kliens és a szerver futása fizikailag is elkülönül egymástól. A Blazor technológiával megvalósított kliens WebAssemby-re fordítva a felhasználók böngészőjében vagy hagyományos szerver oldali kiszolgálással is futhat. A szintén .NET Core alapú szerver egy erre dedikált központi gépen vagy gépeken fut. Jelenleg az adatbázis fizikailag nem különül el a backend logikát futtató komponensektől, de logikailag a két egység egyértelműen külön van választva, így később az adatbázismotor az üzleti logika nagyobb módosítása nélkül lecserélhető. Ez teszi lehetővé a Domain Driven Design és a Repository tervezési minta használata.



**A hagyományos és DDD tervezés rétegeinek összehasonlítása**

A Blazor technológia használatának egyik előnye már az architektúra tervezése során megmutatkozott: A kliens és a szerver által közösen használt DTO (Data Transfer Object) osztályokat nem kell két különböző technológiával duplikáltan megvalósítani, hanem megoszthatók az alkalmazás két része között. Ezáltal a DTO-k módosítása esetén csak egyetlen helyen kell belenyúlni a kódba.

A kliens és a szerver közötti hálózati kommunikációt http kérések és válaszok segítségével oldottam meg. A kliensről indított kéréseket a szerver egy REST API-n keresztül fogadja, majd a kérés feldolgozása után a választ JSON formátumba sorosítva küldi el a kliensnek. A platformfüggetlen REST API lehetővé teszi, hogy a szerver később más (pl.: mobilos vagy desktop) kliens alkalmazásokkal is együttműködjön. Az erőforrások köré szervezett API tervezése során felhasználtam az egyes http igék jelentését és az erőforrásokra egyedi azonosító vagy keresési paraméterek segítségével lehet hivatkozni.

Az adatok perzisztens tárolásáért felelős NoSQL adatbázistnak a MongoDB-t választottam. Az ehhez kapcsolódó kódrészleteket egy külön projektbe szerveztem ki, és a repository mintát követve az egyes MongoDB kollekciókhoz hozzáférő osztályok a BLL-ben (Business Logic) definiált interfészeket valósítanak meg.

A képen szöveg látható

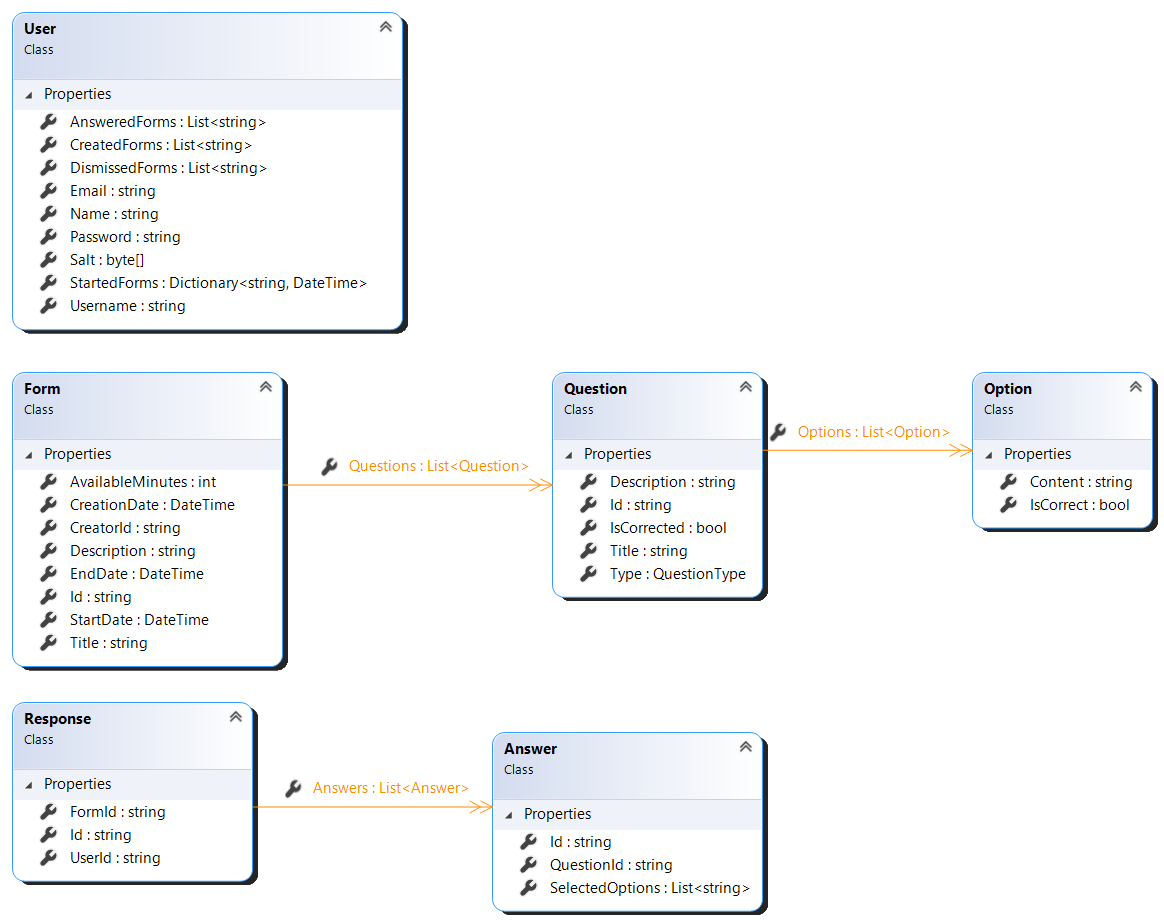
Automatikusan generált leírás

**A visual studio projektek felépítése**

Az **Api** réteg tartalmazza a kontroller osztályokat, és ez felelős az üzleti objektumok DTO-kká történő konvertálásáért, valamint a függőségek injektálásért (Dependency Injection) is. A **BusinessLogic** réteg összetettebb algoritmusokat tartalmaz, mint például az exportálandó Excel fájlok tartalmának előállítása. A **Database** réteg felelős az adatok betöltéséért és perzisztálásáért, a **Web** projekt pedig a Blazor komponenseket és a HTTP kérések indítását végző szolgáltatásokat tartalmazza.

## Az adatbázis felépítése

A MongoDB-ben tárolt entitások felépítése az alábbi:



Az adatok tárolásához három kollekciót használok. A felhasználói adatok a **Users**, a formok és a hozzájuk kapcsolódó kérdések (embedded document-ként) a **Forms**, a válaszok pedig az **Answers** gyűjteménybe kerülnek.

## Frontend architektúra

A Blazor projetben, a kliens oldalon is jól strukturált kódra törekedtem, melyet a mappaszerkezettel és a névterek használatával alakítottam ki. Külön mappába kerültek a model osztályok, a http kommunikációért felelős szolgáltatások, a felületi komponensek, valamint az ezekből felépülő oldalak. A wwwroot mappa tartalmazza a statikus erőforrásokat, mint például az oldal kinézetét leíró css fájlokat.

A képen asztal látható

Automatikusan generált leírás

**A Web projek felépítése**

# Megvalósítás

Az alkalmazás részletes implementációját néhány kiemelt use-case-n keresztül fogom bemutatni.

## Bejelentkezés

A rendszer egyik legalapvetőbb funkciója a regisztrált felhasználók bejelentkezése. A felhasználók a webes felületen felhasználónevük és jelszavuk megadásával tudnak belépni. Ezen mezők megjelenítéséért és az input kezeléséért a LoginForm komponens felelős.

<MatH4>Login to Blazor Form Designer</MatH4>

<EditForm OnValidSubmit=@Submit Model=@Model>

<ObjectGraphDataAnnotationsValidator />

<MatTextField @bind-Value=@Model.Username Outlined="true" Label="Username or Email Address" AutoFocus class="input-field" />

<ValidationMessage For=@(() => Model.Username) />

<MatTextField @bind-Value=@Model.Password Outlined="true" Label="Password" type="password" class="input-field" />

<ValidationMessage For=@(() => Model.Password) />

<MatCheckbox @bind-Value=@Remember Label="Remember me" class="input-checkbox"/>

<MatButton Type="submit" Raised="true" class="input-field">Login</MatButton>

</EditForm>

@code {

[Inject]

private UserService UserService { get; set; }

[Inject]

private ILocalStorageService LocalStorage { get; set; }

private LoginRequest Model { get; set; } = new LoginRequest();

private bool Remember { get; set; }

protected override async Task OnAfterRenderAsync(bool firstRender)

{

var request = await LocalStorage.GetItemAsync<LoginRequest>("user");

if (request != null) { await UserService.Login(request); }

}

private async Task Submit()

{

await UserService.Login(Model);

if (Remember) await LocalStorage.SetItemAsync("user", Model);

}

}

A fenti példakódon csak a komponens lényeges részei látszódnak, annak bemutatására, hogy a komponens betöltődésekor (OnAfterRenderAsync életciklus esemény) az alkalmazás automatikusan megpróbál bejelentkezni a korábban a LocalStorage-ba mentett adatok alapján. A UserService a megfelelő rest API végpont meghívásával jelentkezteti be a felhasználót. Sikeres bejelentkezésnél a válaszüzenet egy JWT tokent is tartalmaz, amely a későbbi kérések során HEADER paraméterként szolgál a felhasználó authentikálására.

A képen szöveg látható

Automatikusan generált leírás

**Szerver oldali validáció sikertelen bejelentkezés esetén**

## Form létrehozása

Egy form létrehozásakor megadható annak címe, témája a form időbeli elérhetőségének intervalluma és a kitöltésére rendelkezésre álló idő.

A képen szöveg látható

Automatikusan generált leírás

**Új form adatainak megadása**

Új kérdést a „create question” gomb segítségével adhatunk hozzá, ilyenkor az új kérdéshez tartozó mezők is megjelennek a felületen. Egy kérdés szerkesztése az alábbi módon jelenik meg:

A képen szöveg látható

Automatikusan generált leírás

**Kérdés hozzáadása vagy szerkesztése**

A kérdés kiválasztott típusától és attól, hogy pontozva lesz-e a kérdés függően jelennek meg válaszlehetőségek megadására szolgáló mezők. Például pontozott, de szöveges válaszlehetőség esetén csak egyetlen (automatikusan helyes) válaszlehetőség megadása támogatott. Egyszeres és többszörös válaszadás esetén az opciókhoz felvehetünk újabbakat vagy törölhetünk a meglévők közül. A kérdés hozzáadásakor (Finish editing question) automatikus validáció történik, melyet a QuestionEditor komponens ValidateOptions() függvénye valósít meg. Mint a kódrészletből látszik rengeteg különböző eset vizsgálata szükséges.

private string ValidateOptions()

{

if ((Model.Type == QuestionType.MultipleChoice.GetAttributeValue() || Model.Type == QuestionType.SingleChoice.GetAttributeValue()) && Model.Options.Where(o => !string.IsNullOrWhiteSpace(o.Content)).Count() < 2)

{

return "Minimum 2 options required for each question.";

}

if (!Model.IsCorrected) return null;

if (Model.Type == QuestionType.Text.GetAttributeValue() || Model.Type == QuestionType.Numeric.GetAttributeValue())

{

if (Model.Options.Where(o => !string.IsNullOrWhiteSpace(o.Content)).Count() != 1) return "This question type requires one correct option.";

}

if (Model.Type == QuestionType.SingleChoice.GetAttributeValue())

{

if (Model.Options.Where(o => !string.IsNullOrWhiteSpace(o.Content)).Where(o => o.IsCorrect).Count() != 1) return "This question type requires one correct option.";

}

if (Model.Type == QuestionType.MultipleChoice.GetAttributeValue())

{

if (Model.Options.Where(o => !string.IsNullOrWhiteSpace(o.Content)).Where(o => o.IsCorrect).Count() < 2) return "This question type requires at least two correct options.";

}

return null;

}

A form létrehozása közben a már hozzáadott kérdések láthatók és szerkesztésükre is van lehetőség. Ekkor a létrehozással azonos űrlap jelenik meg, melynek a korábbi értékekkel már ki lesznek töltve a mezői.

A képen szöveg látható

Automatikusan generált leírás

**A már hozzáadott kérdés nézete**

A form létrehozásakor HTTP POST művelet segítségével jut el a szerverre, ahol a formot létrehozó felhasználó azonosítója a már korábban említett JWT token alapján lesz meghatározva, majd az adatbázisba mentve.

## Saját formok megtekintése és válaszok exportálása

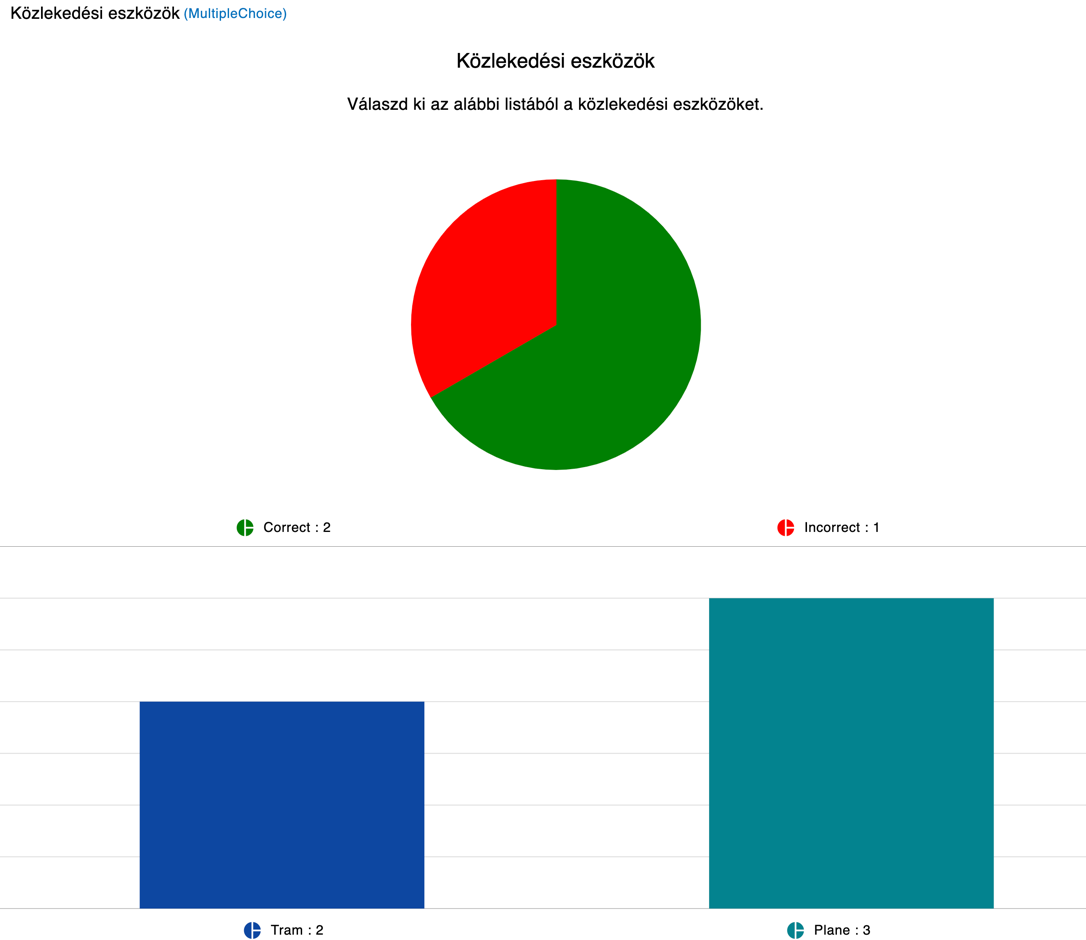
Egy form létrehozása után az alkalmazás automatikusan a MyForms oldalra navigál. ahol az eddig általunk létrehozott formok láthatók.

A képen szöveg látható

Automatikusan generált leírás

**A már létrehozott formok**

A lenyíló fül segítségével statisztikákat jeleníthetünk meg a beérkezett válaszokról. Minden pontozott kérdés esetén látható a helyes és helytelen válaszok aránya, valamint kiválasztásos típusú (Single és Multiple Choice) esetén a válaszok eloszlása is megtekinthető.



**Válaszadási statisztikák**

Szintén ezen az oldalon van lehetőség a válaszok exportálására, ilyenkor egy Excel fájl töltődik le a weboldalról, amely összesítő nézetet és kérdésenkénti részletes nézeteket is tartalmaz.

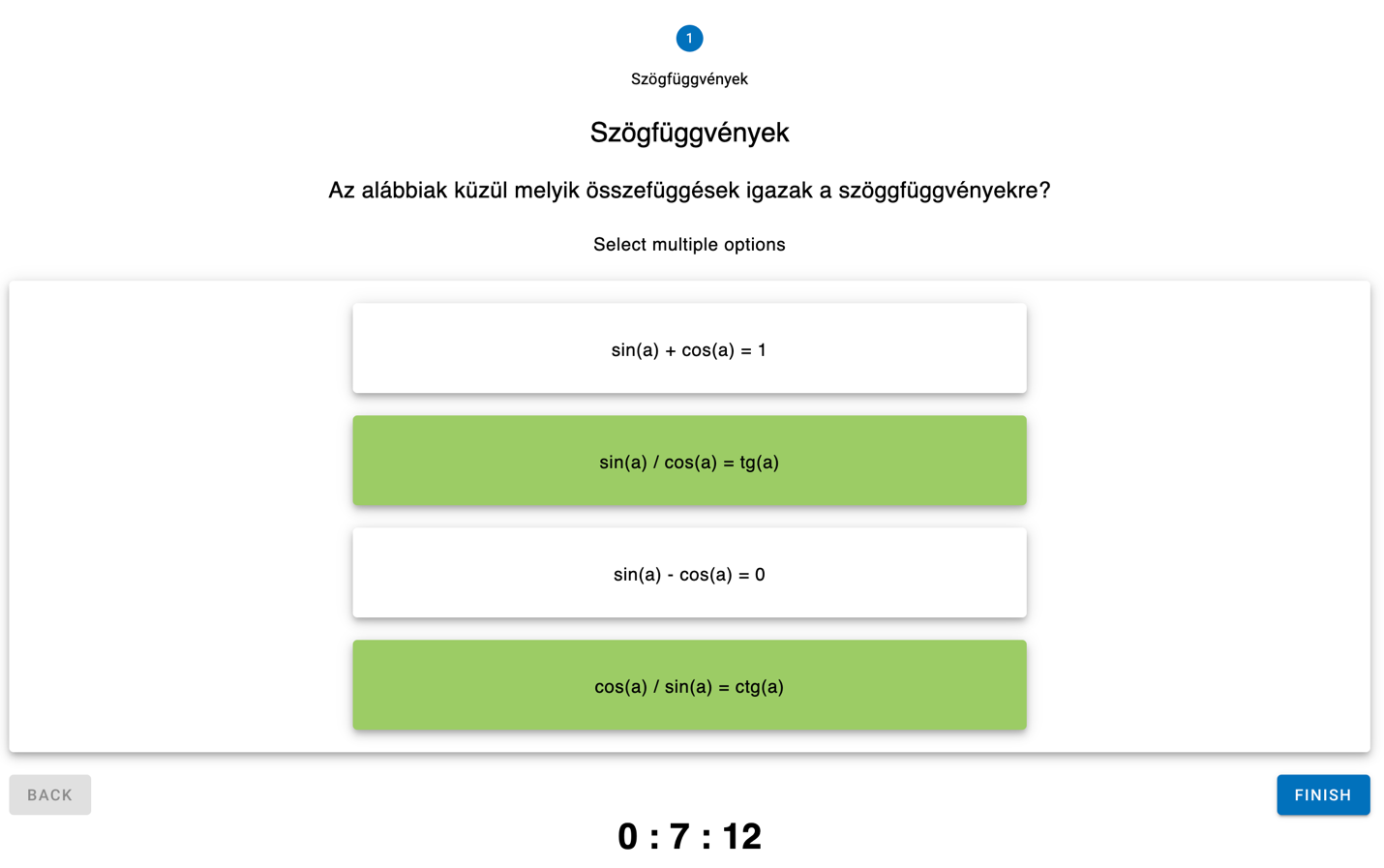
## Form kitöltése

Formok kitöltésére a forms oldalon van lehetőség. Itt elérhetőség alapján csoportosítva találjuk meg a kérdőíveket/teszteket.

A képen szöveg látható

Automatikusan generált leírás

A kitöltés sorén a kérdések között előre és hátra is lépkedhetünk, ilyenkor a korábban megadott válaszaink nem vesznek el, a végén pedig a „finish” gombbal adhatjuk be. A kitöltésből hátralévő időt egy időzítő jelzi, az idő lejártakor automatikusan mentésre kerül a formunk a szerveren is.



**Válaszadás és a fennmaradó idő**

## Egyéb megvalósított funkciók

Az alkalmazásban lehetőség van felhasználók regisztrációjára, egy teszt kitöltése után a felhasználók láthatják az elért pontjaikat és megtekinthetik a helyes válaszokat is.

# Összefoglalás

## Tapasztalatok

Az alkalmazás elkészítése során alaposan megismerkedtem a Blazor technológiával, gyakorlatban is kipróbáltam a MongoDB adatbázis nyújtotta lehetőségeket és elmélyítettem .NET Core fejlesztői ismereteimet. Habár egy viszonylag új technológiáról van szó, a Blazor máris rengeteg UI komponens könyvtárral rendelkezik, melyek közül a feladat során a MatBlazor-t használtam fel. Ez jelentősen meggyorsította a fejlesztés folyamatát, egységes felhasználói élményt nyújt és lényegesen kevesebb css stílus megírását tette szükségessé. Pozitív tapasztalat a DTO osztályok megoszthatósága a kliens és a szerver kódja között, és .NET fejlesztők számára hatalmas előny a UI logika C# nyelvű implementációjának lehetősége. Blazor esetén egyszerű a kliens és szerver oldali renderelés közötti átjárás, a komponensek teljes egészében újrahasználhatók, lényegében csak a Startup osztály lecserélése szükséges.

## Továbbfejlesztési lehetőségek

A félév közben több továbbfejlesztési javaslat is eszembe jutott, azonban a szűkös időkeret miatt nem mindegyik megvalósítására volt alkalmam. Ezek közül a legkiemelkedőbbek:

* A tesztek csak a felhasználók bizonyos csoportja számára legyenek láthatók, ehhez a felhasználói csoportok kialakítására és tesztek adott csoporthoz rendelhetőségére lenne szükség.
* Jelszóval védhető kérdőívek esetén csak a jelszó birtokában lenne lehetőség a kérdések megtekintésére és a válaszadásra.
* Különböző pontszámok és pontozási módszerekkel megoldható lenne, hogy más kérdése más-más pontszámot érjenek vagy helytelen válasz esetén pontlevonás történjen.

# Irodalomjegyzék