

**VÁCI SZAKKÉPZÉSI CENTRUM
BORONKAY GYÖRGY
MŰSZAKI TECHNIKUM ÉS GIMNÁZIUM**

VIZSGAREMEK

**Jávor Márton Áron
Szabó Tamás János
Varga-Labóczki Vazul**

2024.

VÁCI SZAKKÉPZÉSI CENTRUM
BORONKAY GYÖRGY
MŰSZAKI TECHNIKUM ÉS GIMNÁZIUM



VIZSGAREMEK

BoroHFR

Konzulens: Kemenes Tamás
Wiezl Csaba

Készítette: Jávor Márton Áron
Szabó Tamás János
Varga-Labóczki Vazul

Hallgatói nyilatkozat

Alulírottak, ezúton kijelentjük, hogy a vizsgaremek saját, önálló munkánk, és korábban még sehol nem került publikálásra.

Jávor Márton Áron

Szabó Tamás
János

Varga-Labóczki
Vazul

1. Konzultációs lap

Vizsgázók neve: Jávor Márton Áron,
Szabó Tamás János,
Varga-Labóczki Vazul

Vizsgaremek címe: BoroHFR – Iskolai közösségi rendszer

Program nyújtotta szolgáltatások:

- Események, feladatok nyomon követése
- Erőforrások (fájlok, ötletek, információk) megosztása a közösséggel
- Valós idejű szöveges kommunikáció
- Regisztráció
- Meghívó küldése
- Adminisztráció (SysAdmin, Admin, User)
- E-mailek (visszaigazolás, jelszó módosítás)

Sorszám	A konzultáció időpontja	A konzulens aláírása
1.	2023.10.17.	
2.	2023.11.21.	
3.	2023.12.13.	
4.	2024.01.16.	
5.	2024.02.20.	
6.	2024.03.08.	

A vizsgaremek beadható:
Vác, 2024.

.....

Konzulens

A vizsgaremeket átvettem:
Vác, 2024.

.....

A szakképzést folytató
intézmény felelőse

2. Tartalomjegyzék

Hallgatói nyilatkozat	3
1. Konzultációs lap	4
2. Tartalomjegyzék	5
3. Témaválasztás	8
4. Fejlesztői dokumentáció	9
4.1 Fejlesztői környezet	9
4.1.1 Visual Studio Code	9
4.1.2 Visual Studio	9
4.1.3 phpMyAdmin	9
4.1.4 JetBrains Rider	9
4.1.5 Android Studio	9
4.2 Használt programozási nyelvek, technológiák, könyvtárak	10
4.2.1 HTML	10
4.2.2 CSS	10
4.2.3 JavaScript	10
4.2.4 C#	10
4.2.5 CShtml	11
4.2.6 ASP.NET Core MVC	11
4.2.7 Entity Framework Core	11
4.2.8 Bootstrap	11
4.2.9 jQuery	11
4.2.10 JSON	12
4.2.11 JWT	12
4.2.12 MySQL	12
4.2.13 FullCalendar	12
4.2.14 Docker	13
4.2.15 SignalR	13
4.2.16 Flutter	13
4.2.17 Dart	13
4.3 Adatbázis	14
4.3.1 Adatbázis alapadatai	14
4.3.2 Táblák	16
4.4 Szerepkörök	32

4.4.1	Azonosítatlan felhasználó (vendég).....	33
4.4.2	Azonosított felhasználó megerősítetlen e-mail címmel.....	33
4.4.3	Felhasználó	33
4.4.4	Adminisztrátor.....	33
4.4.5	Rendszeradminisztrátor.....	34
4.4.6	Szerepkör diagram	34
4.5	Az alkalmazás felépítése	34
4.5.1	Az alkalmazás részei.....	34
4.5.2	A rendszer működése.....	34
4.5.3	A rendszer részletes leírása	35
4.5.4	Mobilalkalmazás leírása	60
4.6	Tesztelés.....	64
4.6.1	Backend tesztelés	64
4.6.2	API tesztelés példa.....	65
4.6.3	Kontroller tesztelés	71
4.6.4	Kontroller tesztelés példa	72
4.6.5	Frontend tesztelés	75
4.6.6	Frontend tesztelés példa	76
4.7	Továbbfejlesztési lehetőségek	78
4.7.1	Felhasználói profil.....	78
4.7.2	Fájlmenedzsment	79
4.7.3	Hirdetőtábla	79
4.7.4	Értesítések	79
4.7.5	Chat.....	79
4.7.6	Naptár.....	79
4.7.7	Csoportok	79
4.7.8	Privát bejegyzések	80
4.7.9	Intézmény szintű használat	80
4.7.10	Rendszeradminisztrátor biztonság	80
4.7.11	Mobilalkalmazás naptár.....	80
4.7.12	Biometrikus azonosítás	80
4.7.13	Webalkalmazásban elérhető funkciók	80
5.	Felhasználói dokumentáció	81
5.1	A program rövid ismertetése	81
5.2	A szerver futtatása (helyben).....	81

5.2.1	Docker (ajánlott)	81
5.2.2	Standalone (forráskódból)	82
5.2.3	Standalone	83
5.3	HTTPS konfigurálása	83
5.3.1	Az alkalmazás konfigurálása az első indításkor	84
5.4	Az alkalmazás használata felhasználóknak	87
5.4.1	Bejelentkezés	87
5.4.2	Regisztráció	89
5.4.3	Webalkalmazás felület	90
5.4.4	Mobilalkalmazás felület	94
6.	Irodalomjegyzék	99
6.1	Könyvek	99
6.2	Online cikkek	99
6.3	Felhasznált anyagok, eszközök	99
7.	Mellékletek	100
7.1	Online elérhetőség	100
7.2	Forráskód	100

3. Témaválasztás

Az alapötlet a lecke-füzet digitális kiváltása volt: ne kelljen egy külön füzetet hordani, elég legyen egy könnyen elérhető app segítségével rögzíteni a néhány sornyi információt. És ha már digitális, miért ne osszuk meg ezt az információt az osztálytársainkkal? És ha már ez az információ elérhető, logikus lépés például az órarend, az események és hozzájuk tartozó információ megosztása, kérdések esetére a szöveges kommunikáció. A feladatokra elkészített megoldás közzététele is az alapötletek közé tartozott.

Diákként újra és újra átéltem az iskolai kommunikáció nehézségét is. Az osztálycsoportokban nincs benne mindenki, gyakori a spam, és alapvetően nem tanulmányi célt szolgálnak. Szóban nem lehet mindent megbeszélni, és a tapasztalat azt mutatja, hogy ha csak az emlékezetünkre hagyatkozunk, abból végül káosz lesz, mert mindenki máshogy emlékszik, és nem visszakereshetők a dolgok. Ez problémát okozhat hiányzás esetén is. Szóval nincs egy központi és általános megoldás, ahol megtalálhatóak és könnyen kereshetők az információk.

Az iskola és a tanárok által használt kommunikációs csatornák tapasztalatom szerint nem egységesek, több különböző platformot (Teams, e-mail, Google Classroom, stb.) kell aktívan figyelni, különben az ember könnyen lemarad valami fontosról, márpedig az embernek lenne jobb elfoglaltsága is.

A BoroHFR (Boronkay Házi Feladat Rendszer) ezekre a kérdésekre próbál megoldást nyújtani. Egységes rendszer, melyet egy osztályközösség szervez és tart karban, és az iskolával kapcsolatos információk mind könnyedén megtalálhatóak rajta. Elég, ha egy ember észreveszi a Teams-re feltöltött feladatot, és rögzíti a BoroHFR-ben. A rendszer követi a határidőket is, így mindig a legfontosabb feladatokra fordíthatja a felhasználó diák az idejét. A rendszer egy olcsó miniszámítógépről is futtatható a felhasználók alacsony száma miatt, és nem kell külső szolgáltatóra hagyatkozni, így helyben akár az internetkapcsolat megszűnése eseté is működőképes maradhat.

A rendszer másrészt kiváló állatorvosi ló programozói szemszögből: webfejlesztés, adatbáziskezelés, mobilapplikáció-fejlesztés, az ehhez tartozó API, valós idejű kommunikáció és társaik mind a munkafolyamat részét képezik.

4. Fejlesztői dokumentáció

4.1 Fejlesztői környezet

4.1.1 Visual Studio Code

A Visual Studio Code (VS Code) egy nyílt forráskódú kódszerkesztő. Testre szabható, emellett számos programozási nyelvhez és fejlesztői környezethez alkalmazkodik. Alapértelmezetten támogatja a népszerűbb programozási nyelveket, például a C-t, C++-t, C#-t, Pythont, Javát, JavaScriptet és még sok másikat. Kevésbé ismert programozási nyelveket, keretrendszereket támogató csomagokat is könnyen lehet importálni. Maga a programozói felület is személyre szabható témákkal, billentyűparancsokkal és kiegészítőkkel. A fejlesztői környezet elérhető Windows-on, macOS-en és Linuxon.

4.1.2 Visual Studio

A Visual Studio egy teljes körű fejlesztői környezet. Olyan szoftver, mely lehetővé teszi a fejlesztőknek, hogy alkalmazásokat tervezzenek, kódoljanak és teszteljenek. Beépített fordítót, hibakezelőt is kínál. Főleg C# és C++ irányult a támogatottság, de felismer más nyelveket is. Több változatban érhető el: Visual Studio Community, Professional és Enterprise. Ezek különböző szintű szolgáltatásokat és funkciókat kínálnak, amelyek más-más fejlesztési célokhoz és igényekhez szolgálnak.

4.1.3 phpMyAdmin

A phpMyAdmin egy ingyenes, nyílt forráskódú webalapú adminisztrációs eszköz MySQL és MariaDB adatbáziskezeléshez. Gyakran használt műveleteket a webes felületen lehet könnyedén végrehajtani, emellett képes általunk generált SQL-utasításokat is kezelni.

4.1.4 JetBrains Rider

A JetBrains Rider egy az IntelliJ platformon és a ReSharperen alapuló keresztplatform .NET IDE, amely a legtöbb .NET projekt típus támogatására alkalmas beleértve a Mono alapú projektek is. A támogatásnak köszönhetően az alkalmazások széles körét programozhatja le, beleértve .NET asztali alkalmazásokat, szolgáltatásokat és könyvtárakat, Unity játékokat, Xamarin appokat, ASP.NET és ASP.NET Core webes alkalmazásokat. Az egyszerre több runtime környezet futtatásán és hibakeresésén túl maga a Rider is több platformon is fut: Windowson, macOS-en és Linuxon.

4.1.5 Android Studio

Az Android Studio egy integrált fejlesztői környezet Android alkalmazásfejlesztéshez. Ez egy olyan szoftver, amely lehetővé teszi a fejlesztők

számára, hogy könnyen és hatékonyan hozzanak létre Android alkalmazásokat. Az Android Studio számos funkciót kínál, mint például a kódszerkesztés, az alkalmazások tervezése és tesztelése, valamint az alkalmazások futtatása és hibakeresése emulátorokon vagy valós eszközökön. Ez az egyik legelterjedtebb eszköz az Android platformon való fejlesztéshez.

4.2 Használt programozási nyelvek, technológiák, könyvtárak

4.2.1 HTML

A HTML (Hyper Text Markup Language) a standard jelölőnyelv, mely a világháló oldalainak létrehozására, tervezésére használnak. Ez biztosítja a tartalom struktúráját és elrendezését a világhálón. Az HTML-t gyakran használják együtt a CSS (Cascading Style Sheets) és a JavaScript nyelvekkel, hogy fokozzák a weboldalak prezentációját és interaktivitását. A CSS-t az HTML elemek stílusozására és formázására használják, míg a JavaScriptet a weboldalak dinamikus viselkedésének és interaktivitásának hozzáadására. Együtt az HTML, a CSS és a JavaScript alkotják a webfejlesztés alapját.

4.2.2 CSS

A CSS (Cascading Style Sheets) egy stíluslapnyelv, amelyet a weboldalak megjelenítésének formázására és stílusozására használnak. A CSS lehetővé teszi a webfejlesztők számára, hogy elválasszák az egyes webes tartalmak struktúráját és tartalmát a megjelenésétől és stílusától. A CSS-t általában külső stíluslapokként. A CSS-vel kialakított stílusok a különféle webböngészőkben egységesen jelennek meg, ami az egyik legnagyobb előnye ennek a technológiának.

4.2.3 JavaScript

A JavaScript egy dinamikus szkriptnyelv, amelyet általában weboldalak fejlesztésére használnak. Az egyik legelterjedtebb és legfontosabb programozási nyelv a webfejlesztésben. A JavaScript lehetővé teszi az interaktív webes tartalom létrehozását, amely dinamikusabban viselkedik és reagál a felhasználói interakciókra. Nem csak a weboldalak interaktivitásának létrehozására szolgál, hanem teljes értékű alkalmazások, webes játékok és sok más szoftver is fejleszthető vele.

4.2.4 C#

A C# (C-sharp) egy modern, objektumorientált programozási nyelv. Elsősorban a .NET platformhoz kapcsolódik, és az egyik leggyakrabban használt nyelv a Windows alkalmazások, webalkalmazások és szolgáltatások fejlesztéséhez. A C#-t széles körben használják üzleti alkalmazások, játékok, webalkalmazások, adatszolgáltatások és még sok más fejlesztésére.

4.2.5 CShtml

A ".cshtml" kiterjesztés általában a C# (C-Sharp) nyelvet használó Razor motorral rendelkező weboldalakhoz kapcsolódik. Ezek a fájlok olyan weboldalakhoz kapcsolódnak, amelyek dinamikus tartalmat. A Razor egy szerver-oldali sablonmotor, amely lehetővé teszi a C# és HTML szintaxis keverését. Ezáltal lehetővé teszi a fejlesztők számára, hogy dinamikus tartalmat generáljanak és jelenítsenek meg a weboldalakon a C# kód és az HTML markup kombinálásával.

4.2.6 ASP.NET Core MVC

Az ASP.NET Core MVC egy keretrendszer a webalkalmazások fejlesztéséhez. Az ASP.NET a Microsoft webalkalmazásokhoz készült technológiájának egy általános elnevezése, míg az "MVC" a Model-View-Controller tervezési mintát jelenti. Az "ASP.NET Core MVC" keretrendszer segítségével a fejlesztők teljes körű, dinamikus és interaktív webalkalmazásokat készíthetnek, amelyek alkalmazkodnak a modern fejlesztési gyakorlatokhoz és elvárásokhoz. A keretrendszer tartalmazza az alapvető funkciókat és eszközöket a webalkalmazások fejlesztéséhez, például útválasztást, modellátvitelt, nézetrendező motort, hibakezelést és sok más.

4.2.7 Entity Framework Core

Az Entity Framework Core egy könnyűsúlyú és keresztplatformos objektum-relációs leképzési (ORM) keretrendszer a .NET Core és az Entity Framework (EF) továbbfejlesztett változata. Az EF Core a Microsoft által fejlesztett, nyílt forráskódú keretrendszer, amely segítségével az alkalmazásfejlesztők könnyen és hatékonyan tudnak adatbázisokkal kommunikálni az alkalmazásokban. Segítségével az alkalmazásfejlesztők könnyen és hatékonyan tudnak adatbázisokkal kommunikálni az alkalmazásaikban, így lehetővé téve az adatok hatékony kezelését és manipulálását.

4.2.8 Bootstrap

A Bootstrap egy ingyenes és nyílt forráskódú frontend keretrendszer weboldalak és webalkalmazások fejlesztéséhez. Elsősorban HTML, CSS és JavaScript alapú. A Bootstrap célja, hogy segítsen az egyenletes és gyorsan fejlődő webes projektek létrehozásában, minimalizálva a tervezési időt és növelve a projekt hatékonyságát. A keretrendszer egy nagyon népszerű eszköz a webfejlesztők körében, mivel lehetővé teszi az egyszerű és gyorsan fejlődő webes projektek létrehozását, miközben biztosítja a reszponzív design és a konzisztens megjelenés előnyeit.

4.2.9 jQuery

jQuery egy könnyűsúlyú, gyors és keresztplatformos JavaScript könyvtár, amelyet főleg webfejlesztés során használnak. Célja, hogy egyszerűbbé és hatékonyabbá tegye a JavaScript alapú webfejlesztést. Népszerűsége és

kényelme miatt sok webfejlesztő használja azáltalános feladatok megkönnyítésére és az alkalmazások gyors fejlesztésére.

4.2.10 JSON

A JSON (JavaScript Object Notation) egy könnyen olvasható adatcsere formátum, amely gépi feldolgozásra alkalmas. A JSON adatokat egy szöveges formátumban ábrázolja, és adatstruktúrákat, például objektumokat és tömböket képes reprezentálni.

4.2.11 JWT

JWT (JSON Web Token) egy nyílt, kompakt és önálló módon átvihető módszer a biztonságos információk cseréjére egy JSON formátumban. A JWT-t általában az azonosítás és hitelesítés során használják webes alkalmazásokban és API-kban. A JWT-k széles körben használják az azonosítás és hitelesítés során, például a felhasználó bejelentkezési állapotának és az engedélyeknek az átvitelére, valamint a biztonságos kommunikációhoz az alkalmazások és a kliensek között. A JWT-k nagyon hasznosak, mivel könnyen kezelhetők és továbbíthatók HTTP kérések fejléceiben vagy URL paraméterként, és a tartalmuk könnyen leolvasható a fejlesztők számára.

4.2.12 MySQL

A MySQL, ahogy a neve is mutatja, egy SQL-alapú adatbázis-kezelő rendszer vagy DBMS. A nyílt forráskódú szoftvert jelenleg az Oracle, a Java programozási nyelvet is kifejlesztő vállalat tartja karban. A MySQL képes az adatok táblázatokban történő tárolására, kezelésére és megjelenítésére. Kliens-szerver rendszerként működik. Míg az adatbázis szerverként működik, amelyen minden lényeges információ tárolódik, addig a szoftver kliensnek tekinthető. A szoftver segítségével a relációs adatbázis felhasználói különböző lekérdezéseket fogalmazhatnak meg az SQL lekérdezési nyelven, és küldhetik el azokat az adatbázis-rendszernek. Ezeket aztán a MySQL feldolgozza, ezért az adathozzáférés is fontos eleme a MySQL-nek. A MySQL-t nagyfokú platformfüggetlenség jellemzi. A MySQL több mint 20 különböző platformon használható, beleértve a népszerű Windows, macOS és Linux operációs rendszereket is. Ráadásul a MySQL telepítése szuper egyszerű.

4.2.13 FullCalendar

A FullCalendar egy JavaScript alapú naptár könyvtár, amely lehetővé teszi a dinamikus és interaktív naptárak létrehozását webes alkalmazásokban. Gyakran használják ütemezési funkciók, események megjelenítése és kezelése, illetve más időalapú funkciók implementálására. A FullCalendar egy erőteljes eszköz a naptáralapú alkalmazások és ütemezési rendszerek készítéséhez, és nagyon népszerű a webfejlesztők körében a könnyű használhatósága és a széles körű testreszabhatósága miatt.

4.2.14 Docker

A Docker egy platform és egy eszközrendszer, amely lehetővé teszi a szoftveralkalmazások konténerekben történő futtatását és kezelését. A konténerizálás technológiája révén a Docker lehetővé teszi az alkalmazások és azok függőségeinek csomagolását, ami megkönnyíti az alkalmazások konzisztens és megbízható módon való futtatását különböző környezetekben. A Docker nagyon népszerűvé vált a modern szoftverfejlesztésben és üzemeltetésben, mivel segít egyszerűsíteni és automatizálni az alkalmazások telepítését, futtatását és skálázását a konténerek technológiájának felhasználásával.

4.2.15 SignalR

SignalR egy valós idejű webes alkalmazások készítését lehetővé tevő keretrendszer. Segítségével könnyen kialakíthatók olyan alkalmazások, amelyek valós időben kommunikálnak a kliensek és a szerver között. Ez a keretrendszer lehetővé teszi a kétirányú kommunikációt a böngészők és a szerver között, így például az események, üzenetek, adatok valós idejű átvitele vagy frissítése lehetséges. Ez rendkívül hasznos lehet élő chat alkalmazásokban, valós idejű frissítéseket tartalmazó rendszerekben, játékokban, vagy bármilyen alkalmazásban, ahol fontos a valós idejű kommunikáció. SignalR támogatja több platformot, beleértve a .NET keretrendszert, JavaScriptet, Java-t és más programozási nyelveket is. Rugalmas és könnyen használható API-t biztosít, amely lehetővé teszi a fejlesztők számára, hogy könnyen beépítsék a valós idejű kommunikációt a meglévő alkalmazásaikba.

4.2.16 Flutter

A Flutter egy nyílt forráskódú, felhasználói felületi keretrendszer, amelyet a Google fejlesztett ki. Célja alkalmazások gyors és könnyű fejlesztése különböző platformokra, például Androidra és iOS-re. A Flutter segítségével a fejlesztők egyszerre több platformra tudnak kódot írni, minimalizálva ezzel a platformok közötti különbségekkel járó komplexitást. A Flutter alkalmazások magas fokú testreszabhatóságot és gyors, élvezetes felhasználói élményt kínálnak. Az alkalmazásokat úgynevezett widgetek segítségével építhetik fel, amelyek az alapvető építőkövei a felhasználói felületnek. A Flutter a Dart nyelven alapul, így az alkalmazások teljes mértékben Dartban írhatók.

4.2.17 Dart

Dart egy programozási nyelv, amelyet a Google fejlesztett ki az általános célú alkalmazások és különösen a webes és mobil alkalmazások fejlesztésére. A Dart célja, hogy egyszerűen érthető és könnyen használható legyen, miközben erős típusellenőrzést és modern nyelvi funkciókat biztosít.

A Dart nyelvet elsősorban a Flutter keretrendszerhez használják. A Dart nyelv számos funkciót és előnyt kínál a fejlesztőknek, például erős típusellenőrzés, gyors végrehajtás, aszinkron programozás.

A Dart egy modern, hatékony programozási nyelv, amely lehetővé teszi a fejlesztők számára a magas szintű alkalmazások készítését a Flutter keretrendszerrel.

4.3 Adatbázis

4.3.1 Adatbázis alapadatai

Adatbázis szerver: MariaDB 10.x/11.x vagy kompatibilis MySQL verzió (A fejlesztés során MariaDB 10.4.28-at használtunk)

Adatbázis neve: borohfr

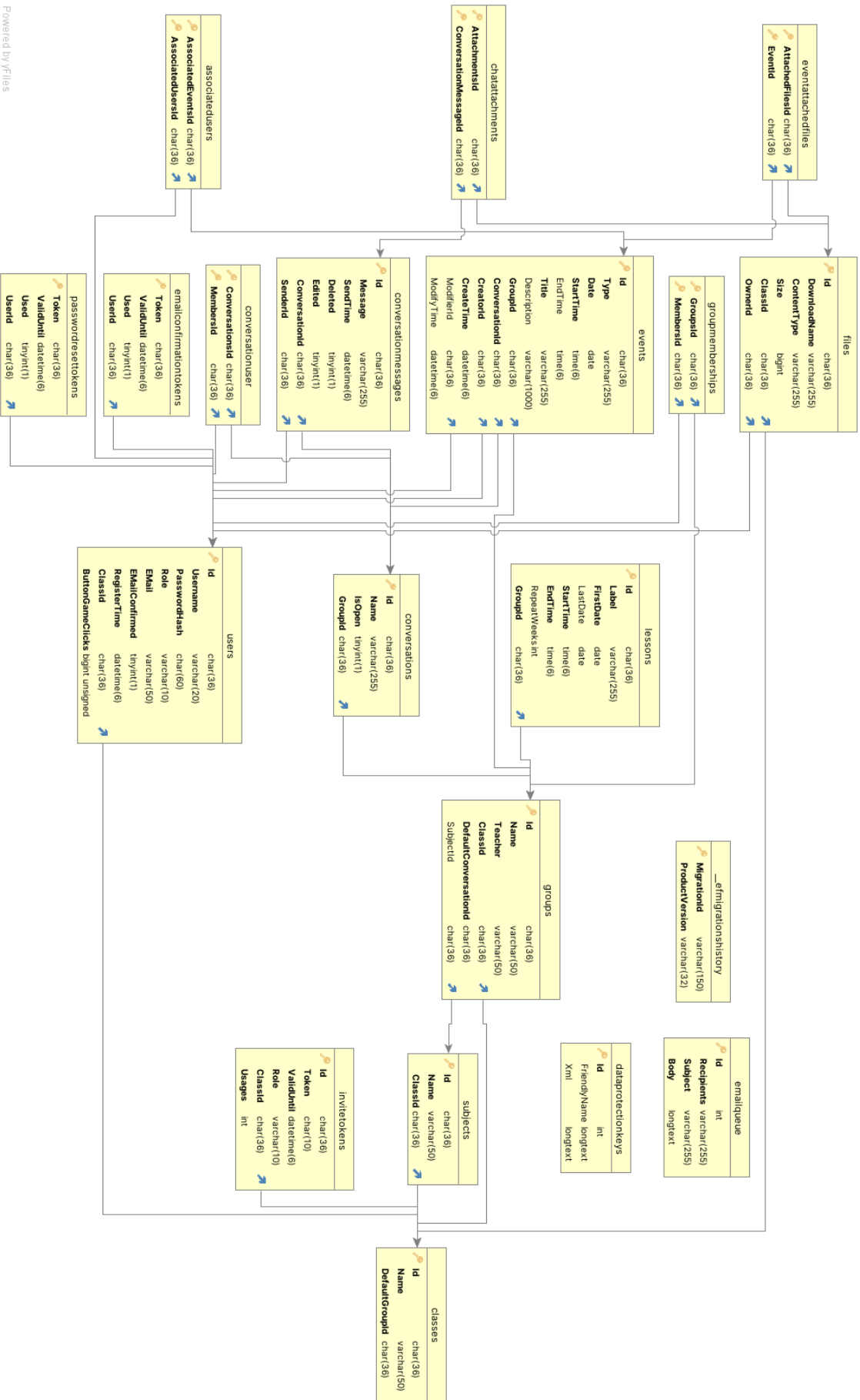
Illesztés: utf8mb4_general_ci

Használt adatbázismotor: InnoDB

Adatbázist létrehozó SQL parancs:




```
CREATE DATABASE IF NOT EXISTS `borohfr`  
DEFAULT CHARACTER SET utf8mb4  
COLLATE utf8mb4_general_ci;
```

Az adatbázisszerkezet a következő oldalon látható.



4.3.2 Táblák

4.3.2.1 Az „associatedusers” tábla

#	Név	Típus	Illesztés	Tulajdonságok	Nulla	Alapértelmezett
1	AssociatedEventsId 	char(36)	ascii_general_ci		Nem	Nincs
2	AssociatedUsersId  	char(36)	ascii_general_ci		Nem	Nincs

Many-to-many kapcsolótábla, eseményeket és feladatokat rendel felhasználóhoz. Például ebben a táblában van tárolva, ha egy felhasználó késznek jelöl egy feladatot.

SQL parancs:

```
CREATE TABLE `associatedusers` (  
  `AssociatedEventsId` char(36) CHARACTER SET ascii  
  COLLATE ascii_general_ci NOT NULL,  
  `AssociatedUsersId` char(36) CHARACTER SET ascii  
  COLLATE ascii_general_ci NOT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4  
COLLATE=utf8mb4_general_ci;  
  
ALTER TABLE `associatedusers`  
  ADD PRIMARY KEY  
  (`AssociatedEventsId`,`AssociatedUsersId`),  
  ADD KEY `IX_AssociatedUsers_AssociatedUsersId`  
  (`AssociatedUsersId`);  
  
ALTER TABLE `associatedusers`  
  ADD CONSTRAINT  
  `FK_AssociatedUsers_Events_AssociatedEventsId` FOREIGN  
  KEY (`AssociatedEventsId`) REFERENCES `events` (`Id`) ON  
  DELETE CASCADE,  
  ADD CONSTRAINT  
  `FK_AssociatedUsers_Users_AssociatedUsersId` FOREIGN KEY  
  (`AssociatedUsersId`) REFERENCES `users` (`Id`) ON  
  DELETE CASCADE;
```


4.3.2.2 A „chatattachments” tábla

#	Név	Típus	Illesztés	Tulajdonságok	Nulla	Alapértelmezett
1	AttachmentsId 🔑	char(36)	ascii_general_ci		Nem	Nincs
2	ConversationMessageId 🔑🔗	char(36)	ascii_general_ci		Nem	Nincs

Many-to-many kapcsolótábla, fájl csatolmányokat rendel üzenetekhez.


SQL parancs:

```
CREATE TABLE `chatattachments` (
  `AttachmentsId` char(36) CHARACTER SET ascii COLLATE
  ascii_general_ci NOT NULL,
  `ConversationMessageId` char(36) CHARACTER SET ascii
  COLLATE ascii_general_ci NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_general_ci;

ALTER TABLE `chatattachments`
  ADD PRIMARY KEY
  (`AttachmentsId`,`ConversationMessageId`),
  ADD KEY `IX_ChatAttachments_ConversationMessageId`
  (`ConversationMessageId`);

ALTER TABLE `chatattachments`
  ADD CONSTRAINT
  `FK_ChatAttachments_ConversationMessages_ConversationMes
  sageId` FOREIGN KEY (`ConversationMessageId`) REFERENCES
  `conversationmessages` (`Id`) ON DELETE CASCADE,
  ADD CONSTRAINT
  `FK_ChatAttachments_Files_AttachmentsId` FOREIGN KEY
  (`AttachmentsId`) REFERENCES `files` (`Id`) ON DELETE
  CASCADE;
```

4.3.2.3 A „classes” tábla

#	Név	Típus	Illesztés	Tulajdonságok	Nulla	Alapértelmezett
1	Id 	char(36)	ascii_general_ci		Nem	Nincs
2	Name	varchar(50)	utf8mb4_general_ci		Nem	Nincs
3	DefaultGroupId	char(36)	ascii_general_ci		Nem	Nincs

Egy osztály nevét és alapértelmezett csoportjának azonosítóját tárolja.

SQL parancs:

```
CREATE TABLE `classes` (
  `Id` char(36) CHARACTER SET ascii COLLATE
  ascii_general_ci NOT NULL,
  `Name` varchar(50) NOT NULL,
  `DefaultGroupId` char(36) CHARACTER SET ascii COLLATE
  ascii_general_ci NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_general_ci;

ALTER TABLE `classes`
  ADD PRIMARY KEY (`Id`);
```

4.3.2.4 A „conversationmessages” tábla

#	Név	Típus	Illesztés	Tulajdonságok	Nulla	Alapértelmezett
1	Id 	char(36)	ascii_general_ci		Nem	Nincs
2	Message	varchar(255)	utf8mb4_general_ci		Nem	Nincs
3	SendTime	datetime(6)			Nem	Nincs
4	Deleted	tinyint(1)			Nem	Nincs
5	Edited	tinyint(1)			Nem	Nincs
6	ConversationId 	char(36)	ascii_general_ci		Nem	Nincs
7	SenderId 	char(36)	ascii_general_ci		Nem	Nincs

Üzeneteket és hozzájuk kapcsolódó extra adatokat tárol.

SQL parancs:

```



CREATE TABLE `conversationmessages` (
  `Id` char(36) CHARACTER SET ascii COLLATE
ascii_general_ci NOT NULL,
  `Message` varchar(255) NOT NULL,
  `SendTime` datetime(6) NOT NULL,
  `Deleted` tinyint(1) NOT NULL,
  `Edited` tinyint(1) NOT NULL,
  `ConversationId` char(36) CHARACTER SET ascii COLLATE
ascii_general_ci NOT NULL,
  `SenderId` char(36) CHARACTER SET ascii COLLATE
ascii_general_ci NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_general_ci;

ALTER TABLE `conversationmessages`
  ADD PRIMARY KEY (`Id`),
  ADD KEY `IX_ConversationMessages_ConversationId`
(`ConversationId`),
  ADD KEY `IX_ConversationMessages_SenderId`
(`SenderId`);

ALTER TABLE `conversationmessages`
  ADD CONSTRAINT
`FK_ConversationMessages_Conversations_ConversationId`
FOREIGN KEY (`ConversationId`) REFERENCES
`conversations` (`Id`) ON DELETE CASCADE,
  ADD CONSTRAINT
`FK_ConversationMessages_Users_SenderId` FOREIGN KEY
(`SenderId`) REFERENCES `users` (`Id`) ON DELETE
CASCADE;

```

4.3.2.5 A „conversations” tábla




#	Név	Típus	Illesztés	Tulajdonságok	Nulla	Alapértelmezett
1	Id 	char(36)	ascii_general_ci		Nem	Nincs
2	Name	varchar(255)	utf8mb4_general_ci		Nem	Nincs
3	IsOpen	tinyint(1)			Nem	Nincs
4	GroupId 	char(36)	ascii_general_ci		Nem	Nincs

Kommunikációs csatornákat („beszélgetéseket”) tárol, üzeneteket fog össze egy egységbe.

SQL parancs:

```
CREATE TABLE `conversations` (  
  `Id` char(36) CHARACTER SET ascii COLLATE  
  ascii_general_ci NOT NULL,  
  `Name` varchar(255) NOT NULL,  
  `IsOpen` tinyint(1) NOT NULL,  
  `GroupId` char(36) CHARACTER SET ascii COLLATE  
  ascii_general_ci NOT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4  
COLLATE=utf8mb4_general_ci;  
  
ALTER TABLE `conversations`  
  ADD PRIMARY KEY (`Id`),  
  ADD KEY `IX_Conversations_GroupId` (`GroupId`);  
  
ALTER TABLE `conversations`  
  ADD CONSTRAINT `FK_Conversations_Groups_GroupId`  
  FOREIGN KEY (`GroupId`) REFERENCES `groups` (`Id`) ON  
  DELETE CASCADE;
```

4.3.2.6 A „conversationuser” tábla


#	Név	Típus	Illesztés	Tulajdonságok	Nulla	Alapértelmezett
1	ConversationsId	 char(36)	ascii_general_ci		Nem	Nincs
2	MembersId	  char(36)	ascii_general_ci		Nem	Nincs

Meny-to-many kapcsolótábla, felhasználókat rendel kommunikációs csatornákhöz („beszélgetésekhez”).

SQL parancs:

```
CREATE TABLE `conversationuser` (  
  `ConversationsId` char(36) CHARACTER SET ascii COLLATE  
  ascii_general_ci NOT NULL,  
  `MembersId` char(36) CHARACTER SET ascii COLLATE  
  ascii_general_ci NOT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4  
COLLATE=utf8mb4_general_ci;  
  
ALTER TABLE `conversationuser`  
  ADD PRIMARY KEY (`ConversationsId`, `MembersId`),  
  ADD KEY `IX_ConversationUser_MembersId` (`MembersId`);  
  
ALTER TABLE `conversationuser`  
  ADD CONSTRAINT  
  `FK_ConversationUser_Conversations_ConversationsId`  
  FOREIGN KEY (`ConversationsId`) REFERENCES  
  `conversations` (`Id`) ON DELETE CASCADE,  
  ADD CONSTRAINT `FK_ConversationUser_Users_MembersId`  
  FOREIGN KEY (`MembersId`) REFERENCES `users` (`Id`) ON  
  DELETE CASCADE;
```

4.3.2.7 A „dataprotectionkeys” tábla

#	Név	Típus	Illesztés	Tulajdonságok	Nulla	Alapértelmezett
1	Id 	int(11)			Nem	Nincs
2	FriendlyName	longtext	utf8mb4_general_ci		Igen	NULL
3	Xml	longtext	utf8mb4_general_ci		Igen	NULL

Adatvédelmi kulcsokat tárol az ASP.NET Core keretrendszer számára.



SQL parancs:

```
CREATE TABLE `dataprotectionkeys` (
  `Id` int(11) NOT NULL,
  `FriendlyName` longtext DEFAULT NULL,
  `Xml` longtext DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_general_ci;

ALTER TABLE `dataprotectionkeys`
  ADD PRIMARY KEY (`Id`);

ALTER TABLE `dataprotectionkeys`
  MODIFY `Id` int(11) NOT NULL AUTO_INCREMENT;
```

4.3.2.8 Az „emailconfirmationtokens” tábla


#	Név	Típus	Illesztés	Tulajdonságok	Nulla	Alapértelmezett
1	Token 	char(36)	ascii_general_ci		Nem	Nincs
2	ValidUntil	datetime(6)			Nem	Nincs
3	Used	tinyint(1)			Nem	Nincs
4	UserId 	char(36)	ascii_general_ci		Nem	Nincs

Email-cím megerősítő tokeneket tárol. A tokeneket a felhasználó e-mailben kapja meg, ezzel tudja igazolni, hogy valós címet adott meg regisztrációkor.

SQL parancs:

```
CREATE TABLE `emailconfirmationtokens` (  
  `Token` char(36) CHARACTER SET ascii COLLATE  
  ascii_general_ci NOT NULL,  
  `ValidUntil` datetime(6) NOT NULL,  
  `Used` tinyint(1) NOT NULL,  
  `UserId` char(36) CHARACTER SET ascii COLLATE  
  ascii_general_ci NOT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4  
COLLATE=utf8mb4_general_ci;  
  
ALTER TABLE `emailconfirmationtokens`  
  ADD PRIMARY KEY (`Token`),  
  ADD KEY `IX_EmailConfirmationTokens_UserId`  
  (`UserId`);  
  
ALTER TABLE `emailconfirmationtokens`  
  ADD CONSTRAINT  
  `FK_EmailConfirmationTokens_Users_UserId` FOREIGN KEY  
  (`UserId`) REFERENCES `users` (`Id`) ON DELETE CASCADE;
```

4.3.2.9 Az „emailqueue” tábla



#	Név	Típus	Illesztés	Tulajdonságok	Nulla	Alapértelmezett
1	Id 	int(11)			Nem	Nincs
2	Recipients	varchar(255)	utf8mb4_general_ci		Nem	Nincs
3	Subject	varchar(255)	utf8mb4_general_ci		Nem	Nincs
4	Body	longtext	utf8mb4_general_ci		Nem	Nincs

E-mail küldési sort valósít meg. Csak akkor van szerepe, ha a program leállításakor még vannak elküldetlen üzenetek, ilyenkor a következő indításkor a rendszer folytatja/újrapróbálja a kézbesítést.

SQL parancs:

```
CREATE TABLE `emailqueue` (  
  `Id` int(11) NOT NULL,  
  `Recipients` varchar(255) NOT NULL,  
  `Subject` varchar(255) NOT NULL,  
  `Body` longtext NOT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4  
COLLATE=utf8mb4_general_ci;  
  
ALTER TABLE `emailqueue`  
  ADD PRIMARY KEY (`Id`);  
  
ALTER TABLE `emailqueue`  
  MODIFY `Id` int(11) NOT NULL AUTO_INCREMENT;
```

4.3.2.10 Az „eventattachedfiles” tábla

#	Név	Típus	Illesztés	Tulajdonságok	Nulla	Alapértelmezett
1	AttachedFilesId 	char(36)	ascii_general_ci		Nem	Nincs
2	EventId 	char(36)	ascii_general_ci		Nem	Nincs

Many-to-many kapcsolótábla, fájlokat rendel eseményekhez. Erre az eseményhez csatolt fájlok miatt van szükség.




SQL parancs:

```
CREATE TABLE `eventattachedfiles` (
  `AttachedFilesId` char(36) CHARACTER SET ascii COLLATE
  ascii_general_ci NOT NULL,
  `EventId` char(36) CHARACTER SET ascii COLLATE
  ascii_general_ci NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_general_ci;

ALTER TABLE `eventattachedfiles`
  ADD PRIMARY KEY (`AttachedFilesId`,`EventId`),
  ADD KEY `IX_EventAttachedFiles_EventId` (`EventId`);

ALTER TABLE `eventattachedfiles`
  ADD CONSTRAINT `FK_EventAttachedFiles_Events_EventId`
  FOREIGN KEY (`EventId`) REFERENCES `events` (`Id`) ON
  DELETE CASCADE,
  ADD CONSTRAINT
  `FK_EventAttachedFiles_Files_AttachedFilesId` FOREIGN
  KEY (`AttachedFilesId`) REFERENCES `files` (`Id`) ON
  DELETE CASCADE;
```

4.3.2.11A „events” tábla

#	Név	Típus	Illesztés	Tulajdonságok	Nulla	Alapértelmezett
1	Id 	char(36)	ascii_general_ci		Nem	Nincs
2	Type	varchar(255)	utf8mb4_general_ci		Nem	Nincs
3	Date 	date			Nem	Nincs
4	StartTime	time(6)			Nem	Nincs
5	EndTime	time(6)			Igen	NULL
6	Title	varchar(255)	utf8mb4_general_ci		Nem	Nincs
7	Description	varchar(1000)	utf8mb4_general_ci		Igen	NULL
8	GroupId 	char(36)	ascii_general_ci		Nem	Nincs
9	ConversationId 	char(36)	ascii_general_ci		Nem	Nincs
10	CreatorId 	char(36)	ascii_general_ci		Nem	Nincs
11	CreateTime	datetime(6)			Nem	Nincs
12	ModifierId 	char(36)	ascii_general_ci		Igen	NULL
13	ModifyTime	datetime(6)			Igen	NULL

Eseményeket és feladatokat tárol. A két tárolt típus nagyon hasonló mezőket használ, ezért vannak egy táblában.

SQL parancs:

```
CREATE TABLE `events` (  
  `Id` char(36) CHARACTER SET ascii COLLATE  
ascii_general_ci NOT NULL,  
  `Type` varchar(255) NOT NULL,  
  `Date` date NOT NULL,  
  `StartTime` time(6) NOT NULL,  
  `EndTime` time(6) DEFAULT NULL,  
  `Title` varchar(255) NOT NULL,  
  `Description` varchar(1000) DEFAULT NULL,  
  `GroupId` char(36) CHARACTER SET ascii COLLATE  
ascii_general_ci NOT NULL,  
  `ConversationId` char(36) CHARACTER SET ascii COLLATE  
ascii_general_ci NOT NULL,  
  `CreatorId` char(36) CHARACTER SET ascii COLLATE  
ascii_general_ci NOT NULL,  
  `CreateTime` datetime(6) NOT NULL,  
  `ModifierId` char(36) CHARACTER SET ascii COLLATE  
ascii_general_ci DEFAULT NULL,  
  `ModifyTime` datetime(6) DEFAULT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4  
COLLATE=utf8mb4_general_ci;  
  
ALTER TABLE `events`  
  ADD PRIMARY KEY (`Id`),  
  ADD KEY `IX_Events_ConversationId` (`ConversationId`),  
  ADD KEY `IX_Events_CreatorId` (`CreatorId`),  
  ADD KEY `IX_Events_Date` (`Date`),  
  ADD KEY `IX_Events_GroupId` (`GroupId`),  
  ADD KEY `IX_Events_ModifierId` (`ModifierId`);  
  
ALTER TABLE `events`  
  ADD CONSTRAINT  
`FK_Events_Conversations_ConversationId` FOREIGN KEY  
(`ConversationId`) REFERENCES `conversations` (`Id`) ON  
DELETE CASCADE,  
  ADD CONSTRAINT `FK_Events_Groups_GroupId` FOREIGN KEY  
(`GroupId`) REFERENCES `groups` (`Id`) ON DELETE  
CASCADE,  
  ADD CONSTRAINT `FK_Events_Users_CreatorId` FOREIGN KEY  
(`CreatorId`) REFERENCES `users` (`Id`) ON DELETE  
CASCADE,  
  ADD CONSTRAINT `FK_Events_Users_ModifierId` FOREIGN  
KEY (`ModifierId`) REFERENCES `users` (`Id`);
```


4.3.2.12A „files” tábla

#	Név	Típus	Illesztés	Tulajdonságok	Nulla	Alapértelmezett
1	Id 🔑	char(36)	ascii_general_ci		Nem	Nincs
2	DownloadName	varchar(255)	utf8mb4_general_ci		Nem	Nincs
3	ContentType	varchar(255)	utf8mb4_general_ci		Nem	Nincs
4	Size	bigint(20)			Nem	Nincs
5	ClassId 🔑	char(36)	ascii_general_ci		Nem	Nincs
6	OwnerId 🔑	char(36)	ascii_general_ci		Nem	00000000-0000-0000-0000-000000000000

Fájlok adatait tárolja. A fájlok maguk nem az adatbázisban, hanem a fájlrendszerben vannak eltárolva, ez a tábla extra adatokat tárol rájuk vonatkozóan. Megállapítható egy fájl tulajdonosa, hogy melyik osztályba tartozik, hány bájt a mérete, valamint a Content-Type http header tartalmát és a fájl eredeti (átalakított, csak betűket, számokat, pontot és alulvonást tartalmazó) nevét is.

SQL parancs:

```
CREATE TABLE `files` (
  `Id` char(36) CHARACTER SET ascii COLLATE
  ascii_general_ci NOT NULL,
  `DownloadName` varchar(255) NOT NULL,
  `ContentType` varchar(255) NOT NULL,
  `Size` bigint(20) NOT NULL,
  `ClassId` char(36) CHARACTER SET ascii COLLATE
  ascii_general_ci NOT NULL,
  `OwnerId` char(36) CHARACTER SET ascii COLLATE
  ascii_general_ci NOT NULL DEFAULT '00000000-0000-0000-
  0000-000000000000'
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_general_ci;

ALTER TABLE `files`
  ADD PRIMARY KEY (`Id`),
  ADD KEY `IX_Files_ClassId` (`ClassId`),
  ADD KEY `IX_Files_OwnerId` (`OwnerId`);

ALTER TABLE `files`
  ADD CONSTRAINT `FK_Files_Classes_ClassId` FOREIGN KEY
  (`ClassId`) REFERENCES `classes` (`Id`) ON DELETE
  CASCADE,
  ADD CONSTRAINT `FK_Files_Users_OwnerId` FOREIGN KEY
  (`OwnerId`) REFERENCES `users` (`Id`) ON DELETE CASCADE;
```

4.3.2.13A „groupmemberships” tábla

#	Név	Típus	Illesztés	Tulajdonságok	Nulla	Alapértelmezett
1	GroupsId	char(36)	ascii_general_ci		Nem	Nincs
2	MembersId	char(36)	ascii_general_ci		Nem	Nincs

Many-to-many kapcsolótábla, felhasználókat (a csoport tagjait) rendel csoportokhoz.

SQL parancs:

```
CREATE TABLE `groupmemberships` (
  `GroupsId` char(36) CHARACTER SET ascii COLLATE
  ascii_general_ci NOT NULL,
  `MembersId` char(36) CHARACTER SET ascii COLLATE
  ascii_general_ci NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_general_ci;

ALTER TABLE `groupmemberships`
  ADD PRIMARY KEY (`GroupsId`,`MembersId`),
  ADD KEY `IX_GroupMemberships_MembersId` (`MembersId`);

ALTER TABLE `groupmemberships`
  ADD CONSTRAINT `FK_GroupMemberships_Groups_GroupsId`
  FOREIGN KEY (`GroupsId`) REFERENCES `groups` (`Id`) ON
  DELETE CASCADE,
  ADD CONSTRAINT `FK_GroupMemberships_Users_MembersId`
  FOREIGN KEY (`MembersId`) REFERENCES `users` (`Id`) ON
  DELETE CASCADE;
```

4.3.2.14A „groups” tábla

#	Név	Típus	Illesztés	Tulajdonságok	Nulla	Alapértelmezett
1	Id	char(36)	ascii_general_ci		Nem	Nincs
2	Name	varchar(50)	utf8mb4_general_ci		Nem	Nincs
3	Teacher	varchar(50)	utf8mb4_general_ci		Nem	Nincs
4	ClassId	char(36)	ascii_general_ci		Nem	Nincs
5	DefaultConversationId	char(36)	ascii_general_ci		Nem	Nincs
6	SubjectId	char(36)	ascii_general_ci		Igen	NULL

Csoportok adatait tárolja. A csoport az osztályon belüli alapegység, alapvetően egy adott tantárgy egy adott oktató által tanított részét jelenti, de más (pl. nem tantárgyhoz kapcsolódó) csoportok is létrehozhatók.

SQL parancs:

```
CREATE TABLE `groups` (  
  `Id` char(36) CHARACTER SET ascii COLLATE  
  ascii_general_ci NOT NULL,  
  `Name` varchar(50) NOT NULL,  
  `Teacher` varchar(50) NOT NULL,  
  `ClassId` char(36) CHARACTER SET ascii COLLATE  
  ascii_general_ci NOT NULL,  
  `DefaultConversationId` char(36) CHARACTER SET ascii  
  COLLATE ascii_general_ci NOT NULL,  
  `SubjectId` char(36) CHARACTER SET ascii COLLATE  
  ascii_general_ci DEFAULT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4  
COLLATE=utf8mb4_general_ci;  
  
ALTER TABLE `groups`  
  ADD PRIMARY KEY (`Id`),  
  ADD KEY `IX_Groups_ClassId` (`ClassId`),  
  ADD KEY `IX_Groups_SubjectId` (`SubjectId`);  
  
ALTER TABLE `groups`  
  ADD CONSTRAINT `FK_Groups_Classes_ClassId` FOREIGN KEY  
  (`ClassId`) REFERENCES `classes` (`Id`) ON DELETE  
  CASCADE,  
  ADD CONSTRAINT `FK_Groups_Subjects_SubjectId` FOREIGN  
  KEY (`SubjectId`) REFERENCES `subjects` (`Id`);
```

4.3.2.15 Az „invitetokens” tábla


#	Név	Típus	Illesztés	Tulajdonságok	Nulla	Alapértelmezett
1	Id 🔑	char(36)	ascii_general_ci		Nem	Nincs
2	Token 🔑	char(10)	utf8mb4_general_ci		Nem	Nincs
3	ValidUntil	datetime(6)			Nem	Nincs
4	Role	varchar(10)	utf8mb4_general_ci		Nem	Nincs
5	ClassId 🔑	char(36)	ascii_general_ci		Nem	Nincs
6	Usages	int(11)			Nem	0

meghívó tokeneket tárol. A meghívó tokeneket az adminisztrátorok tudják létrehozni, ezután az adott osztályba „Usages” számú új felhasználó regisztrálhat a token segítségével.

SQL parancs:

```
CREATE TABLE `invitetokens` (  
  `Id` char(36) CHARACTER SET ascii COLLATE  
  ascii_general_ci NOT NULL,  
  `Token` char(10) NOT NULL,  
  `ValidUntil` datetime(6) NOT NULL,  
  `Role` varchar(10) NOT NULL,  
  `ClassId` char(36) CHARACTER SET ascii COLLATE  
  ascii_general_ci NOT NULL,  
  `Usages` int(11) NOT NULL DEFAULT 0  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4  
COLLATE=utf8mb4_general_ci;  
  
ALTER TABLE `invitetokens`  
  ADD PRIMARY KEY (`Id`),  
  ADD UNIQUE KEY `IX_InviteTokens_Token` (`Token`),  
  ADD KEY `IX_InviteTokens_ClassId` (`ClassId`);  
  
ALTER TABLE `invitetokens`  
  ADD CONSTRAINT `FK_InviteTokens_Classes_ClassId`  
  FOREIGN KEY (`ClassId`) REFERENCES `classes` (`Id`) ON  
  DELETE CASCADE;
```

4.3.2.16 A „lessons” tábla



#	Név	Típus	Illesztés	Tulajdonságok	Nulla	Alapértelmezett
1	Id 	char(36)	ascii_general_ci		Nem	Nincs
2	Label	varchar(255)	utf8mb4_general_ci		Nem	Nincs
3	FirstDate	date			Nem	Nincs
4	LastDate	date			Igen	NULL
5	StartTime	time(6)			Nem	Nincs
6	EndTime	time(6)			Nem	Nincs
7	RepeatWeeks	int(11)			Igen	NULL
8	GroupId 	char(36)	ascii_general_ci		Nem	Nincs

Tanórákat tartalmaz. A tanórák egy adott csoporthoz kapcsolódnak és ismétlődhetnek.

SQL parancs:

```
CREATE TABLE `lessons` (  
  `Id` char(36) CHARACTER SET ascii COLLATE  
  ascii_general_ci NOT NULL,  
  `Label` varchar(255) NOT NULL,  
  `FirstDate` date NOT NULL,  
  `LastDate` date DEFAULT NULL,  
  `StartTime` time(6) NOT NULL,  
  `EndTime` time(6) NOT NULL,  
  `RepeatWeeks` int(11) DEFAULT NULL,  
  `GroupId` char(36) CHARACTER SET ascii COLLATE  
  ascii_general_ci NOT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4  
COLLATE=utf8mb4_general_ci;  
  
ALTER TABLE `lessons`  
  ADD PRIMARY KEY (`Id`),  
  ADD KEY `IX_Lessons_GroupId` (`GroupId`);  
  
ALTER TABLE `lessons`  
  ADD CONSTRAINT `FK_Lessons_Groups_GroupId` FOREIGN KEY  
  (`GroupId`) REFERENCES `groups` (`Id`) ON DELETE  
  CASCADE;
```

4.3.2.17A „passwordresettokens” tábla

#	Név	Típus	Illesztés	Tulajdonságok	Nulla	Alapértelmezett
1	Token 	char(36)	ascii_general_ci		Nem	Nincs
2	ValidUntil	datetime(6)			Nem	Nincs
3	Used	tinyint(1)			Nem	Nincs
4	UserId 	char(36)	ascii_general_ci		Nem	Nincs

Jelszó visszaállításhoz használható tokeneket tárol. A felhasználók szükség esetén e-mailben kapják meg a token, ha elfelejtették a jelszavukat.

SQL parancs:

```
CREATE TABLE `passwordresettokens` (  
  `Token` char(36) CHARACTER SET ascii COLLATE  
  ascii_general_ci NOT NULL,  
  `ValidUntil` datetime(6) NOT NULL,  
  `Used` tinyint(1) NOT NULL,  
  `UserId` char(36) CHARACTER SET ascii COLLATE  
  ascii_general_ci NOT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4  
COLLATE=utf8mb4_general_ci;  
  
ALTER TABLE `passwordresettokens`  
  ADD PRIMARY KEY (`Token`),  
  ADD KEY `IX_PasswordResetTokens_UserId` (`UserId`);  
  
ALTER TABLE `passwordresettokens`  
  ADD CONSTRAINT `FK_PasswordResetTokens_Users_UserId`  
  FOREIGN KEY (`UserId`) REFERENCES `users` (`Id`) ON  
  DELETE CASCADE;
```

4.3.2.18A „subjects” tábla

#	Név	Típus	Illesztés	Tulajdonságok	Nulla	Alapértelmezett
1	Id 🔑	char(36)	ascii_general_ci		Nem	Nincs
2	Name	varchar(50)	utf8mb4_general_ci		Nem	Nincs
3	ClassId 🔑	char(36)	ascii_general_ci		Nem	Nincs

Tantárgyakat tárol, melyek csoportokhoz rendelhetők.

SQL parancs:

```
CREATE TABLE `subjects` (  
  `Id` char(36) CHARACTER SET ascii COLLATE  
  ascii_general_ci NOT NULL,  
  `Name` varchar(50) NOT NULL,  
  `ClassId` char(36) CHARACTER SET ascii COLLATE  
  ascii_general_ci NOT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4  
COLLATE=utf8mb4_general_ci;  
  
ALTER TABLE `subjects`  
  ADD PRIMARY KEY (`Id`),  
  ADD KEY `IX_Subjects_ClassId` (`ClassId`);  
  
ALTER TABLE `subjects`  
  ADD CONSTRAINT `FK_Subjects_Classes_ClassId` FOREIGN  
  KEY (`ClassId`) REFERENCES `classes` (`Id`) ON DELETE  
  CASCADE;
```

4.3.2.19A „users” tábla

#	Név	Típus	Illesztés	Tulajdonságok	Nulla	Alapértelmezett
1	Id 🔑	char(36)	ascii_general_ci		Nem	Nincs
2	Username 🔑	varchar(20)	utf8mb4_general_ci		Nem	Nincs
3	PasswordHash	char(60)	utf8mb4_general_ci		Nem	Nincs
4	Role	varchar(10)	utf8mb4_general_ci		Nem	Nincs
5	EMail 🔑	varchar(50)	utf8mb4_general_ci		Nem	Nincs
6	EMailConfirmed	tinyint(1)			Nem	Nincs
7	RegisterTime	datetime(6)			Nem	Nincs
8	ClassId 🔑	char(36)	ascii_general_ci		Nem	Nincs
9	ButtonGameClicks	bigint(20)		UNSIGNED	Nem	0

Felhasználók adatait tárolja. A jelszó BCrypt hashelve van eltárolva. A „ButtonGameClicks” mező egy egyszerű „játék” állapotát tárolja, melyben a rendszer azt számolja, hogy a felhasználó hányszor nyomott meg egy gombot.


SQL parancs:

```
CREATE TABLE `users` (
  `Id` char(36) CHARACTER SET ascii COLLATE
ascii_general_ci NOT NULL,
  `Username` varchar(20) NOT NULL,
  `PasswordHash` char(60) NOT NULL,
  `Role` varchar(10) NOT NULL,
  `EMail` varchar(50) NOT NULL,
  `EMailConfirmed` tinyint(1) NOT NULL,
  `RegisterTime` datetime(6) NOT NULL,
  `ClassId` char(36) CHARACTER SET ascii COLLATE
ascii_general_ci NOT NULL,
  `ButtonGameClicks` bigint(20) UNSIGNED NOT NULL
DEFAULT 0
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_general_ci;

ALTER TABLE `users`
  ADD PRIMARY KEY (`Id`),
  ADD UNIQUE KEY `IX_Users_EMail` (`EMail`),
  ADD UNIQUE KEY `IX_Users_Username` (`Username`),
  ADD KEY `IX_Users_ClassId` (`ClassId`);

ALTER TABLE `users`
  ADD CONSTRAINT `FK_Users_Classes_ClassId` FOREIGN KEY
(`ClassId`) REFERENCES `classes` (`Id`) ON DELETE
CASCADE;
```

4.3.2.20A „`__efmigrationshistory`” tábla

#	Név	Típus	Illesztés	Tulajdonságok	Nulla	Alapértelmezett
1	MigrationId	 varchar(150)	utf8mb4_general_ci		Nem	Nincs
2	ProductVersion	varchar(32)	utf8mb4_general_ci		Nem	Nincs

Az Entity Framework Core által generált migrációkat követi, az adatbázisra alkalmazott migrációkat sorolja fel. A rendszer innen tudja, hogy milyen változtatásokra van szükség egy esetleges adatbázisszerkezet-változás esetén. SQL parancs:

```
CREATE TABLE `__efmigrationshistory` (  
  `MigrationId` varchar(150) NOT NULL,  
  `ProductVersion` varchar(32) NOT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4  
COLLATE=utf8mb4_general_ci;  
  
ALTER TABLE `__efmigrationshistory`  
  ADD PRIMARY KEY (`MigrationId`);
```

4.4 Szerepkörök

A program különböző felhasználói szerepeket, jogosultsági szinteket határoz meg.

Az alapvető egység az osztály, minden ezeken belül zajlik, logikailag elkülönítve a többi osztálytól. Az osztályokon belül csoportok vannak, melyhez a felhasználók csatlakozhatnak. A felhasználók ezután hozzáférnek az adott csoportban elérhető összes információhoz. Minden osztálynak van egy alapértelmezett csoportja, melyben az osztály összes tagja része. A csoportokhoz egy-egy szöveges kommunikációs csatorna is tartozik, melyhez csak a tagok férnek hozzá.

A rendszer csak azonosítás után érhető el, az azonosítatlan felhasználók csak a bejelentkezési és regisztrációs felületet érik el. Minden jogosultsági szint esetén megkövetelt egy valós és megerősített e-mail cím megadása.

Az alapvető szerepkör a felhasználó (User), aki tartalmakat hozhat létre azokban a csoportokban, melyeknek tagja, és módosíthatja az általa létrehozott tartalmakat. Az adminisztrátorok (Admin) öröklik a felhasználók összes jogát, de tantárgyakat, csoportokat, és tanórákat is létrehozhatnak, felhasználókat hívhatnak meg az osztályukba, valamint a mások által létrehozott tartalmakat törölhetik és módosíthatják. Az adminisztrátorok csak az osztályon belül tevékenykedhetnek, más osztályokra nincs rálátásuk.

Az osztályok magasabb szintű kezelése a rendszeradminisztrátor (SysAdmin) feladata. Ő nem fér hozzá a felhasználók és adminisztrátorok által használt felülethez, csak a saját adminisztrációs felülethez. Itt tud osztályokat létrehozni és törölni, valamint felhasználóknak adminisztrátori jogot adni, és adminisztrátor meghívó kódokat generálni.

4.4.1 Azonosítatlan felhasználó (vendég)

- Bejelentkezés
- Regisztráció
- Jelszó módosító link kérése e-mailben
- Rendszeradminisztrátori bejelentkezés

4.4.2 Azonosított felhasználó megerősítetlen e-mail címmel

- E-mail-cím megerősítése

4.4.3 Felhasználó

- Belépés csoportokba
- Jelszó módosítása
- Felhasználói statisztika megtekintése (saját és másoké)
- Események, feladatok és tanórák megtekintése azokban a csoportokban, melynek tagja
- Események és feladatok létrehozása azokban a csoportokban, melynek tagja
- Saját események és feladatok szerkesztése
- Szöveges üzenetek küldése az általa elérhető eseményekhez és feladatokhoz, valamint az osztályhoz tartozó csatornába
- Üzenet- és eseménycsatolmányok feltöltése
- Részvétel a gomb játékban

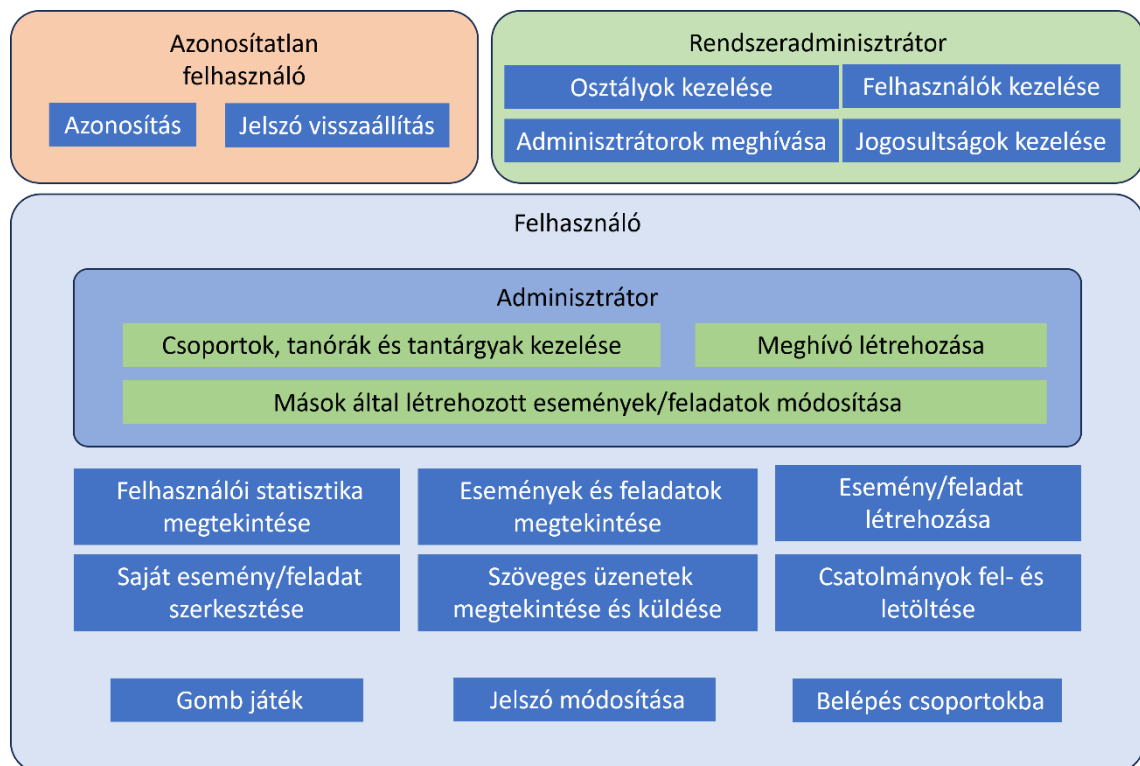
4.4.4 Adminisztrátor

- Belépés csoportokba
- Jelszó módosítása
- Felhasználói statisztika megtekintése (saját és másoké)
- Események, feladatok és tanórák megtekintése azokban a csoportokban, melynek tagja
- Események és feladatok létrehozása azokban a csoportokban, melynek tagja
- Saját és mások által létrehozott események és feladatok szerkesztése
- Szöveges üzenetek küldése az általa elérhető eseményekhez és feladatokhoz, valamint az osztályhoz tartozó csatornába
- Üzenet- és eseménycsatolmányok feltöltése
- Részvétel a gomb játékban
- Csoportok, tanórák és tantárgyak létrehozása
- Felhasználók meghívása

4.4.5 Rendszeradminisztrátor

- Osztály létrehozása
- Osztály törlése
- Adminisztrátorok meghívása osztályokba
- Felhasználók előléptetése adminisztrátorrá, adminisztrátorok lefokozása felhasználóvá
- Felhasználók törlése
- Felhasználók jelszavának visszaállítása

4.4.6 Szerepkör diagram



4.5 Az alkalmazás felépítése

4.5.1 Az alkalmazás részei

Az alkalmazás egy ASP.NET Core MVC backendből (mely a webes frontendet és a mobilalkalmazáshoz használt API-t tartalmazza) és egy Flutter mobil frontendből áll.

4.5.2 A rendszer működése

4.5.2.1 Az alkalmazás indulása

Az alkalmazás induláskor ellenőrzi a megfelelő konfigurációs fájlok jelenlétét. (A konfiguráció helyességét azonban nem ellenőrzi.) Ezután, ha a konfiguráció nem létezik, konfigurációs módban indul el, egyébként normál módban.

4.5.2.1.1 Konfigurációs mód

Konfigurációs módban a rendszer csak a legszükségesebb komponenseket tölti be, melyek adatbázis és egyéb külső függőségek nélkül is működnek. Egy grafikus konfigurációs felületet biztosít, melyen a rendszeradminisztrátor belépési adatait, a tranzakciós e-mailek küldéséhez használt SMTP szerver adatai és a használni kívánt adatbázis adatait kell megadni. Ebben a módban a rendszer a konfiguráció egy részét automatikusan tölti ki, ezek a beállítások csak manuálisan változtathatók (ilyen pl. a JWT-k titkosításához használt kulcs, a fájlok tárolásához használt mappa elérési útvonala, stb). A menüben csak előre lehet lépkedni, a már kitöltött szekciókról a rendszer továbbirányít. A helytelen konfiguráció elkerülése érdekében az adatbázis és az SMTP szerver adatait a rendszer csak akkor engedi menteni, ha azok helyesek. A helyesség egy csatlakozási kísérlet által van ellenőrizve.

A konfiguráció befejezésekor az alkalmazás felszólít az újraindításra, mely után a rendszer normál módban indul.

4.5.2.1.2 Indítás normál módban

Normál módban a rendszer az összes szükséges middleware-t és szolgáltatást betölti. Ilyenkor elindul az e-mail-szolgáltatás, elérhető az adatbázis, működik az autentikáció, és a teljes grafikus felület elérhető. A kezdeti konfigurációs felület ilyenkor nem elérhető. A konfigurációs fájl ilyenkor csak olvasásra van megnyitva, a program nem módosít a tartalmán.

4.5.3 A rendszer részletes leírása

Az ASP.NET Core keretrendszer MVC (Model-View-Controller) architektúráját használja az alkalmazás. Ez lehetővé teszi mind statikus, mind dinamikusan generált weblapok működtetését, valamint rengeteg beépített middleware-rel és szolgáltatással rendelkezik, melyek megkönnyítik a felhasználók azonosítását, a konfiguráció kezelését és olvasását, valamint az egyéb, gyakran használt funkciók megvalósítását. Lehetővé teszi háttérben futó szolgáltatások működtetését is.

4.5.3.1 Konfiguráció

A rendszer a konfiguráció tárolásához egy JSON fájlt használ. (appsettings.json) Ez kezdetben még nem létezik, a konfigurációs fázisban hozza létre a rendszer. A konfigurációs fájlban az alábbi beállítások adhatók meg:

- **Jwt**
 - **Key**: Titkos kulcs (a rendszer automatikusan generál egy 128 karakterből álló hexadecimális kulcsot)
 - **Audience, Issuer**: A többi mezőre nincs szükség, csak komplexebb autorizációs rendszerek esetén
- **FileStorageDir**: Feltöltött fájlok tárolásához használt mappa útvonala
- **ConnectionStrings**
 - **Default**: Az adatbázis eléréséhez használt connection string.
- **SysAdmin**
 - **Email**: A rendszeradminisztrátor e-mail címe.
 - **PasswordHash**: A rendszeradminisztrátor jelszavának Enhanced BCrypt hashe.
- **Smtp**
 - **Server**: Az SMTP szerver hosztneve vagy ip címe
 - **Port**: Az SMTP szerver által használt portszám
 - **Username**: Felhasználónév az SMTP szerveren való bejelentkezéshez.
 - **Password**: Jelszó az SMTP szerveren való bejelentkezéshez.
 - **SenderAddress**: A küldő e-mail címe. Erről a címről kapják a felhasználók az e-maileket. A domainnek meg kell egyeznie az SMTP szerveren regisztrálttal, esetleg a felhasználónév rész is szerver függő lehet. Jellemző érték: noreply@domain.tld
 - **SenderName**: A küldő neve, aliasa. A felhasználók postaládájában ez lesz a küldő neve.

Így néz ki egy konfigurációs fájl:

```
{
  "Jwt": {
    "Key": "<JWT key>",
    "Audience": "<JWT audience>",
    "Issuer": "<JWT issuer>"
  },
  "FileStorageDir": "<fájl tárolási mappa>",
  "ConnectionStrings": {
    "Default": "<adatbázis connection string>"
  },
  "SysAdmin": {
    "Email": "<rendszeradminisztrátor e-mail címe>",
    "PasswordHash": "<rendszeradminisztrátor jelszó hash>"
  },
  "Smtp": {
    "Server": "<smtp szerver hosztneve vagy ip címe>",
    "Port": "<smtp szerver portszáma>",
    "Username": "<smtp szerver felhasználónév>",
    "Password": "<smtp szerver jelszó>",
    "SenderAddress": "<küldő e-mail cím>",
    "SenderName": "<küldő név>"
  }
}
```

4.5.3.2 Biztonság

A felhasználók jelszavait hashelve tárolja a rendszer. Ehhez a BCrypt.Net-Next csomagot használja a rendszer. A módszer lehetővé teszi a munkafaktor növelését, így bizonyos szinten jövőálló megoldásnak tekinthető.

4.5.3.3 A rendszer indulása

A StartupExtensions.cs fájl úgynevezett extension method-okat definiál a szerver működését leíró WebApplicationBuilder objektumhoz. Képes a normál és a konfigurációs módban szükséges szolgáltatások és middleware-ek betöltésére. Először betölti az MVC architektúrához szükséges kontrollereket és nézeteket. (A nézetek a dinamikus HTML generálásért, a kontrollerek a kérések kezeléséért és az adatmanipulációért felelnek.)

```
builder.Services.AddControllersWithViews().ExcludeSpecificControllers(typeof(SetupController));
```

A SetupControllerre nincs szükség, ezért azt egy utólag hozzáadott extension method-del kihagyjuk. Itt megmutatkozik, hogy az ASP.NET Core fejlesztésekor kiemelt figyelmet fordítottak arra, hogy a komplex rendszer működése átlátható legyen, és bármely szinten bele lehessen nyúlni a működésébe. A controller eltávolítását végző kódrészlet például így néz ki:

```
public static void ExcludeSpecificControllers(this ApplicationPartManager partManager, params Type[] controllerTypes)
{
    partManager.FeatureProviders.RemoveAt(0);
    partManager.FeatureProviders.Add(new ExcludeSpecifiedControllersFeatureProvider(controllerTypes));
}
```

Illetve a használt FeatureProvider:

```
private class ExcludeSpecifiedControllersFeatureProvider : ControllerFeatureProvider
{
    2 references
    private readonly Type[] _excludedTypes;

    0 references
    public ExcludeSpecifiedControllersFeatureProvider(Type[] types)
    {
        _excludedTypes = types.Select(x=>x.GetTypeInfo()).ToArray();
    }

    0 references
    protected override bool IsController(TypeInfo typeInfo)
    {
        // [...] itt megnézzük, hogy a vizsgált típus controller-e
        // pl.: benne van-e a nevében a Controller szó, egy controller alapsztály leszármazottja-e, stb.

        //ellenőrizzük, hogy a vizsgált típust ki kell-e hagyni
        if (_excludedTypes.Contains(typeInfo))
        {
            return false;
        }

        return true;
    }
}
```

Hozzáadjuk az OpenAPI Swagger használatához szükséges generátort, szintén szűrve, hogy csak az API kontrollereket fedezze fel.

```
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen(x =>
{
    x.DocumentFilter<BoroHFR.Helpers.SwaggerControllerFilter>();
});
```

A szűrésért az alábbi kódrészlet felel:

```
internal class SwaggerControllerFilter : IDocumentFilter
{
    0 references
    public void Apply(OpenApiDocument swaggerDoc, DocumentFilterContext context)
    {
        foreach (var item in swaggerDoc.Paths.ToArray())
        {
            //eltávolítjuk a /api/ útvonalrészletet nem tartalmazó API metódusokat
            if(!item.Key.ToLower().Contains("/api/"))
            {
                swaggerDoc.Paths.Remove(item.Key);
            }
        }
    }
}
```

A következő lépés az úgynevezett DbContext hozzáadása, melyen keresztül éri el a rendszer az adatbázist. Ehhez EntityFramework Core-t használunk. A DbContext felel a kapcsolat konfigurálásáért, az adatbázismodell létrehozásáért, és az adatokat is rajta keresztül érjük el. (Az adatbázis eléréséről részletesen később).

```
builder.Services.AddDbContext<BoroHfrDbContext>(c =>
{
    string connStr = builder.Configuration.GetConnectionString("Default");
    c.UseMySQL(connStr, ServerVersion.AutoDetect(connStr));
});
```

Itt megmutatkozik az ASP.NET Core konfigurációkezelési erőssége: configuration prividereken keresztül egyesíti a különböző forrásokból származó konfigurációs bejegyzéseket: többek között JSON és INI fájlokat, környezeti változókat, parancssori paramétereket képes kezelni, de saját provider is készíthető, ha például egy adatbázisban szeretnénk konfigurációt tárolni. A BoroHFR egy írni is képes (beállítások mentéséhez) JSON providert implementál, melynek konfigurációs módban, az első indításkor van szerepe. (Normál módban a konfigurációs fájl csak olvasásra van megnyitva.)

Ezután a SignalR szolgáltatás hozzáadása következik, melyet a rendszer a valós idejű kommunikációhoz használ. Ez a háttérben WebSocket technológiát használ.

```
builder.Services.AddSignalR();
```

A további szolgáltatások a kontrollerekben kapnak szerepet, erről bővebben később. Az ASP.NET Core különböző élettartamú szolgáltatásokat képes kezelni:

- Singletonokat, melyekből összesen egy létezik, ezt a példányt használja minden esetben a rendszer
- Tranziens szolgáltatásokat, melyekből akkor készül új példány, ha szükség van rájuk
- Scoped szolgáltatásokat, melyekből minden kérés esetén új példány készül

Lehetőség van saját háttérszolgáltatások definiálására is, ilyen például az itt látható EmailSenderBackgroundService.

```
builder.Services.AddSingleton<EmailService>();
builder.Services.AddHostedService<EmailSenderBackgroundService>();

builder.Services.AddSingleton<ViewRendererService>();

builder.Services.AddSingleton<IHttpContextAccessor, HttpContextAccessor>();
```

A rendszer ezután a fájlátroláshoz használt szolgáltatást adja hozzá a gyűjteményhez, miután ellenőrizte, hogy a célmappa létezik és írható.

```
builder.Services.AddSingleton<FileStorageHandler>();
```

Ezt az autentikációs szolgáltatások inicializálása követi. A belső konfiguráció is nagyon könnyen értelmezhető, de ezt most kihagytam a kódrészletből. A BoroHFR kétféle autentikációs megoldást használ: cookie-kat a böngészők esetén és JWT-eket a REST API esetén.

```
builder.Services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)
    .AddCookie(CookieAuthenticationDefaults.AuthenticationScheme, x =>
    {
        // cookie konfiguráció
    })
    .AddJwtBearer(JwtBearerDefaults.AuthenticationScheme, x =>
    {
        // JWT konfiguráció
    });
```

Természetesen ezután az autorizáció beállítása következik: az ASP.NET támogatja a policy-based (irányelv-alapú) jogosultságellenőrzést. A rendszer három jogosultsági szintet definiál a három típusú felhasználó számára.

```
builder.Services.AddAuthorization(opt =>
{
    opt.AddPolicy("User", p =>
    {
        p.AddAuthenticationSchemes("Bearer", "Cookies");
        p.RequireAuthenticatedUser();
        p.RequireRole("User", "Admin");
    });

    opt.AddPolicy("Admin", p =>
    {
        p.AddAuthenticationSchemes("Bearer", "Cookies");
        p.RequireAuthenticatedUser();
        p.RequireRole("Admin");
    });

    opt.AddPolicy("SysAdmin", p =>
    {
        p.AddAuthenticationSchemes("Cookies");
        p.RequireAuthenticatedUser();
        p.RequireRole("SysAdmin");
    });
});
```

Itt egy alapszintű megoldást láthatunk, sokkal komplexebb szabályok definiálására is van lehetőség.

Ezután a request pipeline felépítése következik: a beérkező kérések az összes használt middleware-en végigmennek, mindegyik elvégzi a saját feladatát, és csak ezután következik a kérés feldolgozása a kontrollerekben. A pipeline felépítéséért az AddNormalPipeline függvény felel.

A request pipeline így néz ki:

1. Kivételkezelő (fejlesztési módban nem)
2. Hsts (fejlesztési módban nem)
3. Swagger és Swagger UI
4. Statikus fájlokat kiszolgáló middleware
5. Végpontválasztó middleware
6. Jogosultság ellenőrző middleware
7. Alapértelmezett útvonal definiáló middleware
8. Szöveges kommunikációhoz használt SignalR hub

Ezután az adatbázis aktuális állapotra hozása történik az adatbázis-migrációk alkalmazásával. Később részletesen.

A szolgáltatások és middleware-ek beállítása ezzel megtörtént, az app készen áll a kérések kiszolgálására.

4.5.3.4 Adatbázis-kapcsolat

A BoroHFR az EntityFramework Core (továbbiakban EF vagy EF Core) nevű ORM-et használja az adatbázis kezeléséhez. Ez jelentősen megkönnyíti a komplex műveleteket és az adatbázis naprakészen tartását. Az EF képes C# osztályokból adatbázis táblákat készíteni, illetve támogatja az ezek közti relációkat is, így az adatbázissal való kapcsolattartáshoz használhatók a magasabb szinten használt modellek. (Képes pl. kapcsolótáblákat automatikusan létrehozni.) A lekérdezésekhez nem szükséges SQL-t írni (bár időnként érdemes felülírni az alapértelmezett viselkedést), a C# nyelv kitűnően használható Linq metódusait használhatjuk, ezt fordítja futási időben SQL-re az EF. (A lekérések teljesítményérzékeny esetekben előre is lefordíthatók, ez a program indításakor történik.) Az EF Core a kiértékelést okozó (pl: ToList(), ToArray(), First(), Single(), stb.) metódusokból aszinkron változatokat is definiál, melyek használatával a kérést végrehajtó szálnak nem kell megvárnia az IO műveletek befejeződését, az idejét hasznosan töltheti, például egy másik lekérdezés feldolgozásával, majd az adatbázis válaszána megérkezése után az egyik szál folytatja a félbehagyott kérés feldolgozását. Az EF az adatbázis-modell változása esetén képes ún. migrációkat készíteni, mely az adatbázis aktuális állapotára építi fel a változtatásokat, és visszavonási lehetőséget is biztosít.

Az EF Core tehát egy rendkívül hasznos eszköz, bár hibás konfiguráció esetén komoly teljesítménycsökkenést okoz. Például hajlamos az összes string mezőt LONGTEXT típusú oszlopban tárolni, ami egy jellemzően 5-10 karakteres szöveg esetén nem szerencsés.

A BoroHFR a Models mappában definiált modellek közül mindet adatbázis táblaként használja. Az osztályokon belül a konfigurációt attribútumok segítségével végezhetjük el. A konfiguráció a legtöbb esetben magától értetődő: A Key attribútum elsődleges kulcsot jelöl, a MaxLength maximális hosszt (az EF ilyenkor kapcsol és VARCHAR(255) mezőt hoz létre), a ForeignKey pedig idegen kulcsot definiál.

Alább a File osztály látható, mely a feltöltött fájlok metaadatait tárolja:

```
public class File
{
    [Key]
    15 references
    public FileId Id { get; set; }
    [MaxLength(255)]
    5 references
    public required string DownloadName { get; set; }
    [MaxLength(255)]
    2 references
    public required string ContentType { get; set; }
    4 references
    public long Size { get; set; }

    2 references
    public UserId OwnerId { get; set; }
    [ForeignKey(nameof(OwnerId))]
    6 references
    public required User Owner { get; set; }

    5 references
    public ClassId ClassId { get; set; }
    [ForeignKey(nameof(ClassId))]
    4 references
    public Class Class { get; set; } = null!;
}
```

A modellek ún. strongly-typed (erősen típusos) azonosítókat használnak, melyek célja az azonos típusú elsődleges kulcsot használó táblák összekeverési lehetőségének megakadályozása. A kulcsokat egy-egy Guid-ot tároló readonly struct valósítja meg, a kontrollerekben való használatot a StronglyTypedIdTypeConverter generikus konverter osztály teszi lehetővé, ezen kívül pedig felülírásra került a ToString metódus az egyszerű és intuitív használathoz (ne kelljen mindig a kulcs value mezőjére meghívni). Azért esett Guid-ra a választás, mert gyakran a kéréseket kezelő controller metódusok paraméterként kapják a modellek elsődleges kulcsát, és el szeretnénk volna kerülni az egész számok által szolgáltatott plusz információ elrejtését, valamint megnehezíteni a kulcsok próbálgatással való megkeresését.

Az EF használatakor az adatbázishoz egy, a DbContext osztályt öröklő osztályon keresztül lehet elérni. Ez jelen esetben a BoroHfrDbContext névre hallgat. A DbContext legfontosabb részét a DbSet típusú tulajdonságok képezik. Ezek a gyűjtemények szimbolizálják az adatbázis tábláit, rajtuk futtathatunk Linq lekérdezéseket.

Az Users táblához tartozó DbSet deklarációja:

```
public virtual DbSet<User> Users { get; set; } = null!;
```

Ez később például az alábbi módon használható:

```
var user = await _dbContext.Users.FirstOrDefaultAsync(x=>x.Username == model.Username);
```

Itt a _dbContext változó egy szolgáltatásgyűjteményből lekért BoroHfrDbContext típusú objektum. Megfigyelhetjük az EF Core aszinkron metódusainak használati módját is.

A lekérdezés az alábbi SQL kódot generálja:

```
SELECT `u`.`Id`, `u`.`ButtonGameClicks`, `u`.`ClassId`,  
`u`.`EMail`, `u`.`EMailConfirmed`, `u`.`PasswordHash`,  
`u`.`RegisterTime`, `u`.`Username`  
FROM `Users` AS `u`  
WHERE `u`.`Username` = @__model_Username_0  
LIMIT 1
```

Az OnModelCreating metódus lehetőséget biztosít az osztályokban nemkívánatos vagy az attribútumokkal nem beállítható konfiguráció elvégzésére. Many-to-many kapcsolótábla beállítása (az EF automatikusan létrehozza, viszont az alapértelmezett elnevezési módszerei hagynak némi kívánni valót maguk után, így ezt érdemes felülírni):

```
modelBuilder.Entity<User>().HasMany(x => x.Groups).WithMany(x => x.Members)  
    .UsingEntity(x=>x.ToTable("GroupMemberships"));
```

Erősen típusos elsődleges kulcs konvertálása az EF számára érthető Guid formátumba:

```
modelBuilder.Entity<User>().Property(x => x.Id)  
    .HasConversion(id => id.value, guid => new UserId(guid))  
    .ValueGeneratedOnAdd();
```

Opcionális relációhoz használt erősen típusos idegen kulcs típuskonverziója:

```
modelBuilder.Entity<Group>().Property(x => x.SubjectId)  
    .HasConversion<Guid?>(id => !id.HasValue ? null : id.Value.value, guid => !guid.HasValue ? null : new SubjectId(guid.Value));
```

A BoroHFR az adatbázis szerkezetének naprakészen tartásához migrációkat használ. Az adatbázis tárolja a saját verzióját, így bármikor könnyedén frissíthető. Migrációt létrehozni a 'dotnet ef migrations add <név>' paranccsal, a legutóbb hozzáadottat törölni a 'dotnet ef migrations remove' paranccsal, az adatbázist frissíteni pedig a 'dotnet ef database update' paranccsal lehet. (A parancsok használatához szükség van az EF Core CLI eszközére, mely a 'dotnet tool install -g dotnet-ef' paranccsal telepíthető. Az eszközt érdemes globálisan telepíteni, erre szolgál a -g kapcsoló.)

A program induláskor alkalmazza az adatbázisra az új migrációkat, így nem szükséges manuálisan frissíteni az adatbázist. Ezért az alábbi kódrészlet felel:

```
using (IServiceScope scope = app.Services.CreateScope())  
{  
    var db = (BoroHfrDbContext)scope.ServiceProvider.GetRequiredService(typeof(BoroHfrDbContext));  
    db.Database.Migrate();  
}
```

A művelethez egy DbContext objektumra van szükség, amely egy service scope-on keresztül elérhető. Az indítási folyamat leírásánál láthattuk, hogy a program egy szolgáltatás gyűjteményt épít, melyből később lekérhetőek azok. Itt pontosan

ez történik, viszont a szolgáltatások élettartama miatt szükség van a ServiceScope létrehozására. Ezután a gyűjteményből lekérhető az adatbáziskezelő szolgáltatás, melyen meghívjuk a Migrate() metódust. A using blokk végén a scope-on meghívásra kerül a Dispose() metódus, mely felszabadítja a megfelelő élettartamú szolgáltatásokat.

4.5.3.5 Az MVC architektúra

A rendszer MVC (Model-View-Controller) architektúrát használ. Ez azt jelenti, hogy a kérések feldolgozásáért (kontroller) és a HTML generálásáért (nézet) a program különböző részei felelnek. Ez lehetővé teszi komplex rendszerek átlátható módon való fejlesztését, valamint egyszerűvé teszi az API kérések kiszolgálását: egyszerűen egy olyan metódust kell definiálnunk, mely nem nézetet (HTML-t), hanem például JSON formátumú információt ad vissza. Az ilyen API metódusokat érdemes külön kontrollerbe szervezni.

A kontrollerek és a nézetek is használják az adatbázishoz is használt modelleket, innen az M a rövidítésben.

Ezen kívül egy fő része van a rendszer által használt architektúrának: az úgynevezett ViewModel-ek (nézetmodell). Ezek feladata az, hogy kapcsolatot teremtsenek a nézetek és kontrollerek, a felhasználó számára megjelenített felület és az adatbázis által használt modellek közt. (Például más típusú vagy kevesebb adatot jelenítünk meg vagy kérünk be a felhasználótól, mint ami a modellben szerepel.) A ViewModel-ek használata emlékeztet az MVVM (Model-View-ViewModel) architektúrára, viszont a működésért itt a kontroller felel.

4.5.3.6 Modellek és nézetmodellek

A rendszer által használt modelleket a Models mappa tartalmazza. Két kakukktojás (ICalendarEvent interfész és Lesson osztály) kivételével ezek adatbázis táblákat valósítanak meg. Az adatbázis szekcióban látható egy ilyen modell. Lényegében csak az adott entitás (pl. felhasználó) adatait tartalmazzák, függvényeik jellemzően nincsenek.

Ez alól kivételt képez az InviteTokens osztály, mely definiál egy tokeneket előállító statikus függvényt:

```
public static string GenerateToken()
{
    const string validChars = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    return RandomNumberGenerator.GetString(validChars, 10);
}
```

A RandomNumberGenerator osztály kriptográfiához is használható random értékeket állít elő, itt egy 10 karakter hosszú alfanumerikus kód előállítására használjuk.

Az osztályok közti relációk azonosak az adatbázisban lévőkkel.

A nézetmodellek a ViewModels mappában találhatók, kontroller szerint rendezve. Nevük az alábbi sémát követi: [Nézet]ViewModel. Általában a hozzájuk tartozó nézettel együtt nyernek értelmet.

Ez például a felhasználó információit megjelenítő nézethez tartozó nézetmodell:

```
public class UserDataViewModel
{
    4 references
    public string Username { get; set; }
    4 references
    public string StorageUsed { get; set; }
    4 references
    public UserRole Role { get; set; }
    4 references
    public int EventsCreated { get; set; }
    4 references
    public int HomeworksDone { get; set; }
    4 references
    public int EventsAttended { get; set; }
    4 references
    public DateOnly RegDate { get; set; }
}
```

Az osztály tulajdonságai nem mind részei az adatbázisban tárolt modellnek, ezek külön lekérdezések eredményei. A lekérdezések a kontrollerben történnek, majd a nézet paraméterként kapja a nézetmodellt.

```
[HttpGet(template: "userdata")]
0 references
public async Task<IActionResult> UserData()
{
    var user = await _dbContext.Users.FirstAsync(x=>x.Id == this.GetUserId());
    var storage = new ByteSize(await _dbContext.Files.Where(x=>x.Owner == user).SumAsync(x=>x.Size));
    var model = new UserDataViewModel()
    {
        Username = user.Username,
        Role = user.Role,
        RegDate = DateOnly.FromDateTime(user.RegisterTime.Date),
        EventsAttended = await _dbContext.Events
            .Where(x=>x.Type == EventType.Event && x.AssociatedUsers.Contains(user))
            .CountAsync(),
        HomeworksDone = await _dbContext.Events
            .Where(x=>x.Type == EventType.Task && x.AssociatedUsers.Contains(user))
            .CountAsync(),
        EventsCreated = await _dbContext.Events.Where(x => x.Creator == user).CountAsync(),
        StorageUsed = storage.Humanize()
    };
    return View(model);
}
```

Az adatok egy listában vannak megjelenítve:

```
Információ:
<ul>
    <li>Regisztráció dátuma: @Model.RegDate</li>
    <li>Rang: @(Model.Role == UserRole.User ? "Felhasználó" : "Adminisztrátor")</li>
    <li>Létrehozott események: @Model.EventsCreated</li>
    <li>Elkészített feladatok: @Model.HomeworksDone</li>
    <li>Események, melyeken részt vett: @Model.EventsAttended</li>
    <li>Felhasznált tárhely: @Model.StorageUsed</li>
</ul>
```

4.5.3.7 Nézetek

A nézetek HTML kód generálásáért felelnek. Általában kapnak egy nézetmodellt a kontrollerből, és az ebben található adatokat jelenítik meg. A nézetek a Views mappában találhatóak, kontrollerek szerint mappákba rendezve. A Shared mappa kontrollerek közt megosztott elemeket tartalmaz. A nézetek .cshtml fájlok, melyek a C# és a HTML ötvözetek. Használhatóak benne a szokásos vezérlési szerkezetek (if, while, for, foreach), ezeken belül is definiálhatunk HTML elemeket, valamint a kapott nézetmodell tulajdonságait és a helyi változókat megjeleníthetjük a weblapon szöveges formában. A változók elérésében a '@' karakter segít. Példa:

Használatba vesszük a nézetmodellt tartalmazó névteret és a nézet modelljeként deklaráljuk:

```
@using BoroHFR.ViewModels.Groups
@model IndexViewModel
```

A HTML kódon belül C# kódrészleteket használunk:

```
<table class="table table-dark">
  <thead>
    <tr> ...
  </tr>
</thead>
<tbody>
  @foreach (var group in Model.Groups)
  {
    <tr>
      <td>@group.Name</td>
      <td>@group.Teacher</td>
      <td> ...
    </td>
  </tr>
  }
</tbody>
</table>
```

A nézetekben is elvégezhetnénk a kontroller által végzett munkát, hiszen támogatja például az @inject direktívát, mellyel szolgáltatásokat kérhetünk le, de ez nem javasolt, mivel pont a felelősségek és feladatok különválasztása teszi áttekinthetővé a kódbázist.

Bizonyos cshtml fájlok úgynevezett layout-ok (elrendezés), melyek azt a célt szolgálják, hogy az oldal fejlécét és az egy kontrolleren belül többször használt keretet csak egyszer kelljen megírni. Ilyen layout például a Views/Shared/_Layout.cshtml, melyet több kontroller is használ. Ebben alapvetően a navigációs sávot, az oldal alján található footert, az oldal fejlécét és a szükséges css és js fájlokat betöltő tageket találjuk. A layoutot használó nézet oda kerül beszúrásra, ahol a layout meghívja a RenderBody() függvényt. A layoutok úgynevezett szekciókat is definiálhatnak. Ezek arra valók, hogy a nézetekben szereplő szkript és stílus elemek, valamint azon HTML elemek, melyeket a layoutban szeretnénk elhelyezni, a helyükre kerüljenek. Például a script tagek a body végére, a style tagek a fejlécbe, stb. Ehhez a layoutban meg

kell hívni a `RenderSection()` metódust, mely paraméterként kapja az adott szekció nevét, és azt, hogy a nézeteknek kötelezően definiálniuk kell-e a szekciót.

Layout:

```
@await RenderSectionAsync("Scripts", required: false)
```

Nézet:

```
@section Scripts{
    <script>
        $("#searchform").submit(x => search(x));

        async function search(sm) {
            sm.preventDefault();
            loadSearchResults();
        }

        // [...]
    </script>
}
```

Létezik egy harmadik fajta nézet fájl is, az úgynevezett `PartialView` (részleges nézet). Ez annyit jelent, hogy az adott nézethez nem tartozik layout, hanem csak az oldal egy részletét küldi vissza a rendszer. Ezt tipikusan az oldal dinamikusan változó részeihez használjuk, például egy keresés eredményeit így tölthetjük be, ha nem szeretnénk az egész oldalt újratölteni.

Definiáljuk az eredményeket tartalmazó div-et:

```
<div id="resultscontainer" class="d-flex flex-column">
    <p class="text-center">Még nincs itt semmi. Keress rá egy tantárgyra!</p>
</div>
```

Szkript segítségével betöltjük a keresés eredményét a div-be:

```
async function loadSearchResults() {
    $("#resultscontainer").html('<spinner>');
    var text = await postData('@Url.Action("SubjectSearch")', { query: $("#subjsearch").val() });
    $("#resultscontainer").html(text);
}
```

Látható, hogy a `.cshtml` fájlokban akár a szkriptet is generálhatjuk dinamikusan. Az `Url.Action()` függvény helyén futásidőben a megadott action (kontroller függvény) elérési útvonala jelenik meg.

A Views mappában található még a `_ViewImports.cshtml` fájl is, mely a globális (összes nézetre vonatkozó) `using` direktívákat tartalmazza.

4.5.3.8 *Kontrollerek*

A kontrollerek a `Controllers` mappában találhatók. A `Controller` vagy a `ControllerBase` osztály leszármazottai. (Attól függően, hogy API kontrollerről van-e szó, vagyis szükség van-e nézetekre.)

A kontrollerek constructor injection segítségével kapják meg az általuk használt szolgáltatásokat a szolgáltatás gyűjteményből. Például:

```
private readonly BoroHfrDbContext dbContext;  
  
0 references  
public HomeController(BoroHfrDbContext dbContext)  
{  
    this.dbContext = dbContext;  
}
```

A kontrollerek általában elérési út szerint is elkülönülnek egymástól. Ez a Route attribútum segítségével könnyen megvalósítható, hiszen a controlleren belüli összes végpont öröklí a megadott útvonalat. Az attribútum használatakor szükség van a routing middleware-re:

```
app.UseRouting();
```

Több Route attribútum is jelen lehet egy controlleren, ekkor minden megadott útvonalra érkező kérést a controller fog kezelni.

Az Authorize attribútum a jogosultságok ellenőrzésére szolgál: csak a megadott policy-nak (irányelv) vagy autentikációs sémának való megfelelés esetén érhető el az adott controller. Nem csak controllerre, hanem a benne található függvényekre is alkalmazható, ilyenkor ez felülírja az osztály szinten megadott szabályt. Ha azt szeretnénk, hogy egy controller vagy végpont azonosítatlan felhasználókat is kiszolgáljon, ezt az AllowAnonymous attribútummal jelezhetjük. Több Authorize attribútum jelenléte esetén a keretrendszer AND kapcsolatot hoz létre: a felhasználó minden jelzett szabálynak meg kell, hogy feleljen. Az autorizációs attribútumok csak akkor vannak hatással a program működésére, ha induláskor betöltésre került és konfigurálva lett a megfelelő middleware:

```
builder.Services.AddAuthorization(/* [...] */);  
app.UseAuthorization();
```

Az Authorize és a Route attribútum használatára példa:

```
[Route(template: "settings")]  
[Authorize(policy: "User")]  
1 reference  
public class SettingsController : Controller
```

Itt a „settings” a controller gyökeri útvonala (vagyis a controlleren belül definiált végpontok a /settings/[...] útvonalon lesznek elérhetőek.), a „User” pedig egy, az indításkor definiált autorizációs irányelv neve.

A kontrollerek fő elemei az úgynevezett Action-ök (művelet), melyek speciális függvények. Ezek állnak a kérések kezelési láncának végén: egy előfeldolgozott és ellenőrzött, kifejezetten a részükre szóló kérést kell feldolgozniuk. Visszatérési értékük többféle lehet: valamilyen HTTP válaszkód (esetleg plusz tartalom társaságában), nézet, részleges nézet, vagy valamilyen serializálható objektum. Az action-ök lehetnek aszinkron függvények is, ami az erőforráshasználat optimalizációját segíti elő azokban az esetekben, amikor IO műveletekre van szükség. Ebben az esetben a legtöbb ilyen függvény aszinkron, mivel az adatbázisműveletek esetén érdemes kihasználni az aszinkron feldolgozás

lehetőségét. Az elnevezési konvenció ilyenkor megkövetelné, hogy a függvények nevében szerepeljen az „Async” szó, viszont ezeket a függvényeket – a unit tesztek kivéve – kizárólag a keretrendszer hívja meg, így ettől eltekinthetünk, ami kissé megkönnyíti a kontrollerek átláthatóságát és az action-ök keresését. (Nem kell azon gondolkodni, hogy az adott action vajon aszinkron-e?)

Az action-ökhöz attribútumok segítségével HTTP metódusok rendelhetők (GET, POST, DELETE, stb.), meghatározható a kontrolleren belüli útvonaluk, és a query paramétereik (az alapvető típusok esetén típussal együtt) is. A query-ben nem szereplő paramétereket a keretrendszer a kérés testéből (body) nyeri ki. Az action-ök akár szolgáltatást is kaphatnak paraméterként, ez a constructor injection-höz hasonlóan működik.

Alább egy GET metódus található a [kontroller]/joingroup/[id] útvonalon, mely a felhasználót hozzáadja a megadott csoporthoz, aszinkron módon. A metódus BadRequest (HTTP 400) vagy Ok (HTTP 200) eredménnyel tér vissza. Ezt a keretrendszer fogja a megfelelő formátumban visszaküldeni válaszként.

```
[HttpGet(template: "joingroup/{id:guid}")]
0 references
public async Task<IActionResult> JoinGroup(GroupId id)
{
    var user = await GetCurrentUserAsync();
    var group = await _dbContext.Groups
        .Where(x=>x.Subject!.Class.Id == user.ClassId && x.Id == id)
        .SingleOrDefaultAsync();
    if (group is null)
    {
        return BadRequest(error: "User cannot join that group.");
    }
    group.Members.Add(user);
    await _dbContext.SaveChangesAsync();
    return Ok();
}
```

Ha egy action nézetet ad vissza, a keretrendszer először a Views/[Kontroller]/[Action].cshtml útvonalon keresi azt. Ha nem találja, megnézi a Views/Shared mappában is. Ha a használni kívánt nézet neve különbözik az action nevéől azt is át kell adni paraméterként.

```
return View();
```

```
return View(model);
```

```
return View(viewName: "CreateEvent", VM);
```

Részleges nézetek esetén:

```
return PartialView(viewName: "EventInfoPartial", VM);
```

Egy másik gyakran használt visszatérési érték a RedirectToAction(). Ez egy átirányítás eredményt küld a kliensnek, a cél action útvonalával.

```
return RedirectToAction(actionName: "Index");
```

4.5.3.9 Validáció és űrlapok

Az ASP.NET Core MVC szerveroldali validációja és űrlaptámogatása kitűnő.

Ezt a BoroHFR ki is használja.

Űrlapok létrehozásakor általában egy nézetmodellel dolgozunk, melynek mezői az űrlap bemeneti mezőinek tartalmát tárolják. A Html segédfüggvények segítségével könnyedén hozhatunk létre input mezőket (a példában textbox), hozzájuk tartozó címkéket, valamint validációs üzeneteket is megjeleníthetünk. A generált HTML elemek tulajdonságait egy anonim objektumon keresztül állíthatjuk be, a C#-ban foglalt kulcsszavakat a '@' karakterrel kezdve. A megfelelő azonosítók és nevek beállításra kerülnek rajtuk, de ezek is felülírhatóak.

```
@using (Html.BeginForm(FormMethod.Post, new
{
    @class = "px-5 coolform",
}))
{
    <h1>Bejelentkezés</h1>
    <div class="form-group py-1">
        @Html.LabelFor(x => x.Username, "Felhasználónév:", new { @class = "form-label"})
        @Html.TextBoxFor(x => x.Username, null,
            new { @class="form-control bg-dark text-light", @required=""})
        @Html.ValidationMessageFor(x => x.Username, "", new
        {
            @class = "text-danger"
        })
        @if (Model is not null && Model.BadUsername)
        {
            <span class="text-danger">Érvénytelen felhasználónév.</span>
        }
    </div>

    <!-- többi mező -->

    <button class="btn btn-primary" type="submit">Bejelentkezés</button>
}
```

Ezután nincs más dolgunk, mint a nézetmodellben megadni a validációs szabályokat, valamint a kontrollerben ellenőrizni a paraméterként kapott nézetmodell érvényességét.

Nézetmodell:

```
public class LoginViewModel
{
    [Required(ErrorMessage = "Add meg a felhasználóneved!")]
    [MinLength(3, ErrorMessage = "A felhasználónevek minimum 3 karakter hosszúak.")]
    [MaxLength(20, ErrorMessage = "A felhasználónevek maximum 20 karakter hosszúak.")]
    [DataType(DataType.Text)]
    5 references
    public string Username { get; set; }
    [Required(ErrorMessage = "Add meg a jelszavad!")]
    [MinLength(8, ErrorMessage = "A jelszavak minimum 8 karakterből állnak.")]
    [DataType(DataType.Password)]
    5 references
    public string Password { get; set; }
    1 reference
    public bool RemainLoggedIn { get; set; }
    3 references
    public bool BadUsername { get; set; }
    3 references
    public bool BadPassword { get; set; }
}
```

Kontroller:

```
[HttpPost("login/{returnUrl?}")]
5 references
public async Task<IActionResult> Login(LoginViewModel model, string? returnUrl)
{
    if (!ModelState.IsValid)
    {
        return View(model);
    }

    // azonosítás ...
}
```

A ModelState objektum tárolja a kérés törzséből kiolvasott modell validációs információit. Ha az IsValid tulajdonság igazat ad vissza, nincs hiba, különben visszaküldjük a nézetet a kapott modellel. A nézetben pedig a `Html.ValidationMessageFor()` helyén megjelenik a megfelelő hibaüzenet. Ezzel elkerültük, hogy hibás adatot tároljunk, valamint a felhasználó számára is megjelenítettünk egy hibaüzenetet. A validációs szabályokat csak egy helyen (a nézetmodellben vagy a modellben) kell megadni, így változás esetén ezek módosítása is egyszerű.

4.5.3.10 Autentikáció

Az indításról szóló részben már volt szó az autentikációról. A konfiguráció ennyiből áll:

```
builder.Services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)
    .AddCookie(CookieAuthenticationDefaults.AuthenticationScheme, x =>
    {
        x.LoginPath = "/authentication/login";
        x.AccessDeniedPath = "/authentication/login";
        x.Cookie.Name = "ChocolateBiscuit";
        x.SlidingExpiration = true;
        x.Cookie.IsEssential = true;
        x.Cookie.SameSite = SameSiteMode.Strict;
    })
    .AddJwtBearer(JwtBearerDefaults.AuthenticationScheme, x =>
    {
        x.RequireHttpsMetadata = true;
        x.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuerSigningKey = true,
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidateLifetime = true,
            ValidIssuer = builder.Configuration[key: "Jwt:Issuer"],
            ValidAudience = builder.Configuration[key: "Jwt:Audience"],
            IssuerSigningKey = new SymmetricSecurityKey(System.Text.Encoding.UTF8.GetBytes(builder.Configuration[key: "Jwt:Key"]!))
        };
    });
```

Betöltjük és konfiguráljuk a cookie-kat és JWT-ket kezelő szolgáltatásokat.

```
builder.Services.AddAuthorization(opt =>
{
    opt.AddPolicy(name: "User", p =>
    {
        p.AddAuthenticationSchemes(schemes: "Bearer", "Cookies");
        p.RequireAuthenticatedUser();
        p.RequireRole(roles: "User", "Admin");
    });

    opt.AddPolicy(name: "Admin", p =>
    {
        p.AddAuthenticationSchemes(schemes: "Bearer", "Cookies");
        p.RequireAuthenticatedUser();
        p.RequireRole(roles: "Admin");
    });

    opt.AddPolicy(name: "SysAdmin", p =>
    {
        p.AddAuthenticationSchemes(schemes: "Cookies");
        p.RequireAuthenticatedUser();
        p.RequireRole(roles: "SysAdmin");
    });
});
```

Létrehozzuk az jogosultságellenőrző irányelveket.

Ezután a felhasználó azonosítása következik. Ezt cookie-k esetén az AuthenticationController végzi. (JWT tokenet az ApiAuthenticationController-n keresztül lehet szerezni.) Regisztráció során eltároljuk a felhasználó bejelentkezési adatait az adatbázisban. Bejelentkezéskor ellenőrizzük a felhasználó nevét és jelszavát, és ha nem egyezik az adatbázisban tárolt adatokkal, hibaüzenetet jelenítünk meg.

```

var user = await _dbContext.Users.FirstOrDefaultAsync(x=>x.Username == model.Username);
if (user is null)
{
    model.BadUsername = true;
    return View(model);
}

if (!BCrypt.Net.BCrypt.EnhancedVerify(model.Password, user.PasswordHash))
{
    model.BadPassword = true;
    return View(model);
}

```

Miután azonosítottuk a felhasználót, itt az ideje, hogy megkapja a további műveletekhez használható cookie-t. Ehhez létrehozuk a felhasználó adatait tartalmazó principalt a `GetAuthenticationData` függvény segítségével, majd `SignIn` választ küldünk vissza a kliensnek. A böngésző ezt eltárolja, és minden további kéréshez csatolja.

```

var (principal, properties) = GetAuthenticationData(user, model.RemainLoggedIn);

if (returnUrl is null)
{
    properties.RedirectUri = Url.Action(action: "Index", controller: "Home");
}

return SignIn(principal, properties,
    Microsoft.AspNetCore.Authentication.Cookies.CookieAuthenticationDefaults.AuthenticationScheme);

private (ClaimsPrincipal principal, AuthenticationProperties properties) GetAuthenticationData(User u, bool persistentLogin)
{
    var claims = new List<Claim>()
    {
        new Claim(ClaimTypes.Name, u.Username),
        new Claim(ClaimTypes.Email, u.Email),
        new Claim(ClaimTypes.Role, u.Role.ToString()),
        new Claim(type: "Id", u.Id.value.ToString())
    };
    var identity = new ClaimsIdentity(claims, authenticationType: "Cookies");
    var principal = new ClaimsPrincipal(identity);
    var authSettings = new AuthenticationProperties
    {
        IsPersistent = persistentLogin,
        ExpiresUtc = persistentLogin ? DateTimeOffset.UtcNow.AddMonths(months: 1) : DateTimeOffset.UtcNow.AddHours(hours: 1),
        AllowRefresh = true
    };
    return (principal, authSettings);
}

```

A felhasználóhoz 4 információt tárolunk a cookie-ban: a felhasználónevét, az e-mail címét, a szerepkörét és az azonosítóját. Ezek az információk később a kontrollerekben és a nézetekben a `User.Claims` tulajdonságon keresztül elérhetők. A szerepkörnek megadott értéket a jogosultságellenőrző middleware is használja.

4.5.3.11 Szöveges kommunikáció, chat

A BoroHFR SignalR segítségével valósít meg valós idejű kommunikációt. Ehhez szükség van egy hub-ra, ami a kapcsolatokat és a függvényhívásokat kezeli. (A Hub osztály leszármazottja.) Ez leginkább egy kontrollerre hasonlít. Az útvonalát nem Route attribútummal, hanem indításkor adjuk meg:

```
app.MapHub<BoroHFR.Hubs.ChatHub>("/hubs/chat");
```

Itt is használhatjuk az `Authorize` attribútumot jogosultság ellenőrzéshez.

```

[Authorize("User")]
2 references
class ChatHub : Hub

```

A hubon belüli publikus metódusok elérhetőek a kliens oldalról. A rendszer által használt ChatHub csak három ilyen függvényt definiál: SubscribeToConversation (feliratkozás egy csatorna frissítéseire), UnsubscribeFromConversation (leiratkozás egy csatorna frissítéseiről), SendMessage (üzenet küldése).

A SubscribeToConversation felveszi a hívó kapcsolat objektumát a csatornához tartozó csoportba. A csoportrendszer a SignalR egyik szolgáltatása, a kapcsolódó kliensek logikailag csoportokba rendezhetők, majd a csoportokon egyszerre hajthatunk végre műveleteket.

Csoporthoz való hozzáadás:

```
await Groups.AddToGroupAsync(Context.ConnectionId, conversationId.ToString());
```

A Context.ConnectionId a SignalR által generált, kapcsolatonként egyedi azonosító, a conversationId a célcsatorna azonosítója.

Függvényhívás a csoport összes tagján:

```
await Clients.Group(conversationId.ToString())
    .SendAsync("ReceiveMessage", conversationId /* további paraméterek */);
return msg.Id;
```

A SendAsync() a kliensoldalon meghívja a megadott nevű függvényt, viszont nem várja meg a kliens választ a visszatérési értékkel. Ha arra is szükség lenne, az InvokeAsync() függvényt használhatnánk.

A kliens oldalon JS kód fut, ott csak a ReceiveMessage függvény van definiálva:

```
this.connection = new signalR.HubConnectionBuilder()
    .withUrl("/hubs/chat")
    .configureLogging(signalR.LogLevel.Information)
    .withAutomaticReconnect()
    .build();

this.connection.on("ReceiveMessage", this.ReceiveMessage);
```

A ReceiveMessage megkapja a szerver által küldött paramétereket, és azok alapján hozza létre az üzenetet megtestesítő HTML elemet.

Üzenetküldéskor a kliens meghívja a szerver SendMessage függvényét, majd megvárja, amíg a szerver meghívja az ő ReceiveMessage függvényét. A megoldás így nem hibátűrő, nem lehet újrapróbálni egy küldést, viszont az ablakban megjelenő üzenetek biztosan tárolásra kerültek az adatbázisban.

A csatolmányok kezeléséhez a kliens a fájl API-t használja a fájlok feltöltéséhez, eltárolja a feltöltött fájlok azonosítóját, majd az üzenet küldésekor a kapott azonosítókat is elküldi.

Létrehozzuk a feltöltendő fájlt tartalmazó FormData objektumot:

```
var formData = new FormData();
formData.append('file', file);
```

A kliens a fájlok feltöltéséhez XMLHttpRequest-et használ:

```
var xhr = new XMLHttpRequest();
```

A kapott azonosítókat rejtett inputokban helyezzük el:

```

// ha sikeres volt a feltöltés, eltároljuk az üzenethez tartozó formban
// a csatolmány azonosítóját
xhr.addEventListener('load', function (event) {
    var files = JSON.parse(event.target.responseText);
    for (var i = 0; i < files.length; i++) {
        var field = document.createElement("input");
        field.setAttribute("type", "hidden");
        field.setAttribute("name", "attachment");
        field.value = files[i];
        $("#chat-box-" + channelId + " .publisher .attachment-guids").append(field);
    }
    //visszaállítjuk a fájl input értékét
    $("#chat-box-" + channelId + " input[type=file]")[0].value = '';
});

//elindítjuk a kérést, elküldjük a fájlt
xhr.open('POST', '/api/file', true);
xhr.send(formData);

```

A szervernek ellenőriznie kell a megadott azonosítók érvényességét:

```

var dbAttachments = await _dbContext.Files
    .Where(x=>x.ClassId == user.ClassId && attachmentIds.Contains(x.Id))
    .ToListAsync();

```

Ezután a már ellenőrzött fájlokat hozzárendeljük az üzenethez, majd a módosításokat elmentjük az adatbázisba:

```

msg.Attachments = dbAttachments;
await _dbContext.SaveChangesAsync();

```

4.5.3.12 E-mail rendszer

Az e-mail rendszer 2 részből áll. Egy párhuzamosan elérhető sorból és egy háttérszolgáltatásból. Az ezeket megvalósító fájlok a Services mappában találhatóak.

Az EmailService valósítja meg a sort, és singleton élettartammal van regisztrálva a szolgáltatás gyűjteményben. →


```

public class EmailService
{
    private ConcurrentQueue<Email> SendQueue = new();

    2 references
    public bool Any => SendQueue.Count > 0;

    public event Action? EmailAdded;

    3 references
    public virtual void Enqueue(Email email)
    {
        SendQueue.Enqueue(email);
        EmailAdded?.Invoke();
    }

    1 reference
    public Email Dequeue()
    {
        Email? email;
        var result = SendQueue.TryDequeue(out email);
        if (result)
            return email!;

        throw new InvalidOperationException();
    }

    1 reference
    public Email[] Snapshot()
    {
        return SendQueue.ToArray();
    }
}

```

Az EmailAdded esemény szerepe az, hogy „felébressze” a háttérszolgáltatást, ami ezután addig küldi az e-maileket, amíg a sor nem lesz üres.

A küldő háttérszolgáltatás implementációja az „EmailSenderBackgroundService.cs” fájlban található. A MailKit csomagot használja az SMTP kapcsolathoz. Kezeli a várakozási sor EmailAdded eseményét.

A háttérszolgáltatás hozzáadása a szolgáltatásgyűjteményhez háttérszolgáltatásként:

```

builder.Services.AddHostedService<EmailSenderBackgroundService>();

```

A szolgáltatás constructor injection segítségével kapja meg a szükséges szolgáltatásokat, valamint magát a ServiceProvider-t is, mivel az adatbázisműveletekhez használt DbContext élettartama nem teszi lehetővé a közvetlen használatot. (A háttérszolgáltatás singletonnak számít.). Az SMTP kapcsolathoz használt információt a konfigurációból olvassa ki.


```

public class EmailSenderBackgroundService : IHostedService
{
    private readonly MailboxAddress _senderAddress;
    private readonly ILogger<EmailSenderBackgroundService> _logger;
    private readonly EmailService _emailQueue;
    private readonly DnsEndPoint _serverEndpoint;
    private readonly NetworkCredential _credential;
    private readonly SmtpClient _client;
    private readonly IServiceProvider _serviceProvider;

    private CancellationTokenSource? _tokenSource;

    private Email? _currentEmail;

    0 references
    public EmailSenderBackgroundService(IConfiguration config,
        ILogger<EmailSenderBackgroundService> logger, EmailService queue,
        IServiceProvider serviceProvider)
    {
        _logger = logger;
        IConfigurationSection smtp = config.GetSection(key: "Smtp");
        _serverEndpoint = new(smtp[key: "Server"]!, int.Parse(smtp[key: "Port"]!));
        _credential = new NetworkCredential(smtp[key: "Username"], smtp[key: "Password"]);
        _senderAddress = new(smtp[key: "SenderName"], smtp[key: "SenderAddress"]);
        _emailQueue = queue;
        _client = new();
        _client.Disconnected += ClientDisconnected;
        _serviceProvider = serviceProvider;
    }
}

```

Indításkor a szolgáltatás betölti a küldési sorba az adatbázisban tárolt elemeket. Erre azért van szükség, mert ha a rendszer leállításkor még vannak elemek a sorban, a program ezeket elmenti az adatbázisba.

A küldési sor helyreállítása után regisztrálja az EmailAdded eseménykezelőt, mely elindítja az e-mail-küldő metódust egy háttérzálon. Ennek az az értelme, hogy így az eseménykezelő azonnal visszatérhet.

```

public async Task StartAsync(CancellationToken cancellationToken)
{
    using (var scope = _serviceProvider.CreateScope())
    {
        var context = scope.ServiceProvider.GetRequiredService<BoroHfrDbContext>();
        var queue = await context.EmailQueue.ToArrayAsync();
        foreach (var item in queue)
        {
            cancellationToken.ThrowIfCancellationRequested();
            _emailQueue.Enqueue(item);
            context.EmailQueue.Remove(item);
        }
        await context.SaveChangesAsync(cancellationToken);
        if (queue.Any())
        {
            OnEmailAdded();
        }
    }
    _emailQueue.EmailAdded += OnEmailAdded;
}

```

```

public async Task StopAsync(CancellationToken cancellationToken)
{
    _emailQueue.EmailAdded -= OnEmailAdded;
    var snapshot = _emailQueue.Snapshot();
    using (var scope = _serviceProvider.CreateScope())
    {
        var context = scope.ServiceProvider.GetRequiredService<BoroHfrDbContext>();
        context.EmailQueue.AddRange(snapshot);
        if (_currentEmail is not null)
            context.EmailQueue.Add(_currentEmail);
        await context.SaveChangesAsync(cancellationToken);
    }
    _client.Dispose();
}

```

```

private void OnEmailAdded()
{
    if (!_client.IsConnected)
    {
        Task.Run(SendEmailsInQueue);
    }
}

```

Ha nem sikerül csatlakozni a kiszolgálóhoz, a folyamat hibaüzenetet logol, és kilép a metódusból. Egy e-mailt 5-ször próbál meg elküldeni a rendszer, aztán hibaüzenetet logol és folytatja a következő e-mail küldésével.

```

for (int i = 0; i < 5; i++)
{
    //megpróbáljuk elküldeni az e-mailt
    try...
}
// ha a _currentEmail nem null, nem sikerült elküldeni
if (_currentEmail is not null)
{
    _logger.LogError(message: $"Unable to send email: {_currentEmail.Subject}");
    _currentEmail = null;
}

```

E-mail küldés akkor történik, ha egy felhasználónak meg kell erősítenie az e-mail-címét vagy az elfelejtett jelszó funkciót használja. Az e-mailek HTML formátumúak, és dinamikus generálják őket a Views/Email mappában található nézetek. Mivel az e-mail generálása nem nézet eredménybe történik, a Razor.Templating.Core csomagot használjuk. A küldéshez extension method-okat hoztunk létre.

Ez a függvény például létrehoz egy jelszóvisszaállító tokent az adott felhasználó számára, majd elküldi neki e-mailben.

```
public static async Task SendPasswordResetEmailAsync(this ControllerBase controller,
    BoroHfrDbContext dbContext, EmailService emailService, User user)
{
    var token = PasswordResetToken.Create(user);
    dbContext.PasswordResetTokens.Add(token);
    await dbContext.SaveChangesAsync();
    var confirmUri = new Uri(
        $"{controller.Request.Scheme}://" +
        $"{controller.Request.Host}" +
        $"{controller.Url.Action(action: "NewPassword", controller: "Authentication",
            new { token = token.Token.value })}",
        UriKind.Absolute);
    string msg = await RazorTemplateEngine.RenderPartialAsync(viewName: "/Views/Email/ResetPassword.cshtml",
        viewModel: new ResetPasswordViewModel() { Username = user.Username, ResetUrl = confirmUri });
    emailService.Enqueue(email: new Email(subject: "BHR - Jelszó visszaállítása", msg, user.Email));
}
```

Az elküldött e-mail így néz ki:

Szia, jmrttn!

Az alábbi linken módosíthatod a jelszavad:

[Módosítás](#)

A link 1 óra után érvénytelenné válik.

Ez egy automatikusan küldött üzenet, ne válaszolj rá! Az elérhetőségek megtalálhatók a weboldalon.

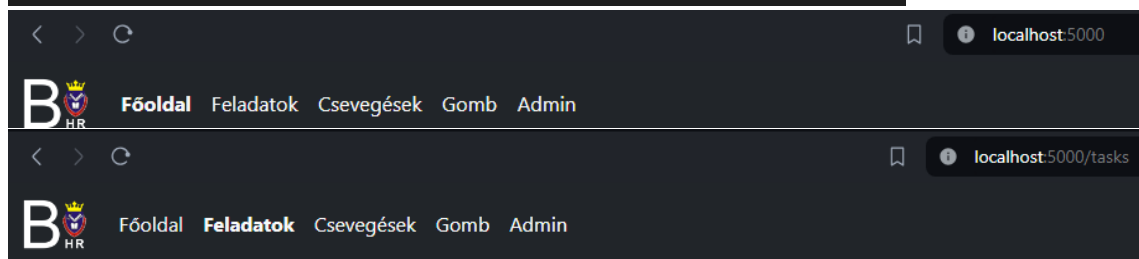
Boronkay Házi Feladat Rendszer - borohfr.jmrttn.dev

4.5.3.13A „TagHelpers” mappa

A mappában található IsActiveTagHelper (<https://bencull.com/blog/is-active-route-tag-helper-asp-net-mvc-core>) célja, hogy a jelenlegi útvonalnak megfelelő navigációs link ki legyen emelve a többi közül. A TagHelper segítségével így elég a megfelelő HTML elemre elhelyezni az „is-active-route” attribútumot, valamint a _ViewImports.cshtml fájlban használatba venni a segédosztályt:

```
<li class="nav-item">
    <a class="nav-link text-light" is-active-route asp-controller="Tasks"
        asp-action="Index">Feladatok</a>
</li>
```

```
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```



4.5.3.14A „wwwroot” mappa

A wwwroot mappában statikus tartalmak találhatók: képek, ikonok, CSS és JS fájlok. Ezeket a rendszer a statikus fájlokat kiszolgáló middleware-en keresztül

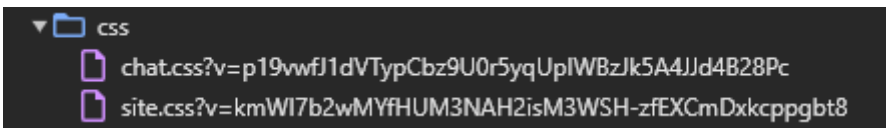
teszi elérhetővé. Ehhez szükség van a program indításakor a következő beállításra:

```
app.UseStaticFiles();
```

A használat a következő módon történik:

```
<link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
<link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />
<link rel="stylesheet" href="~/css/chat.css" asp-append-version="true" />
```

Az „asp-append-version” attribútum arra való, hogy a böngésző ne a gyorsítótárjában eltárolt verziót használja a fájlokból – ha van újabb verzió. Ez működés közben úgy néz ki, hogy a böngésző a fájlnev végére elhelyezi az adott fájlverzióhoz kapcsolható azonosítót. Az azonosító megváltozik, ha a fájl módosítás történik.



4.5.4 Mobilalkalmazás leírása

4.5.4.1 Alkalmazás felépítése

Flutterben az egész alkalmazás felhasználói felülete widgetekből épül fel. Ezek az elemek alkotják az alkalmazás képernyőit és az azokon megjelenített tartalmat.

Minden oldal és azon megjelenített felület widget. Az oldalakat, valamint a többször felhasznált widgeteket külön fájlokban tároljuk.

Például:



Két fő widgetet különböztetünk meg: Stateful widget és a Stateless widget. Ezek között a fő különbség az, hogy a Stateful widget képes tárolni és kezelni az állapotát, míg a Stateless widget nem.

A Stateful widgetek akkor hasznosak, ha az adott widget állapota változik az alkalmazás futása során, és ez a változás befolyásolja a megjelenítést.

4.5.4.2 Példa Stateless widgetre

```
class MyTextField extends StatelessWidget {  
  final bool pw;  
  final TextEditingController ctrl;  
  const MyTextField({super.key, required this.pw, required this.ctrl,});  
  
  @override  
  Widget build(BuildContext context) {  
    return Padding( // Padding ...  
  }  
}
```

Ebben a kódrészletben a bejelentkező oldal egy szövegbeviteli mezőjét láthatjuk.

4.5.4.3 Példa Stateful widgetre

```
class LoginPage extends StatefulWidget {  
  const LoginPage({super.key});  
  
  @override  
  State<LoginPage> createState() => _LoginPageState();  
}  
  
class _LoginPageState extends State<LoginPage> { ...
```

Ebben a kódrészletben a bejelentkező oldalt láthatjuk

Az alkalmazás induláskor a main.dart fájlt nyitja meg, majd futtatja abban a main metódust. Ez létrehozza a MyApp widgetet, amely beállítja kezdőlapnak a bejelentkezési oldalt.

```
void main() {  
  runApp(MyApp());  
}  
  
class MyApp extends StatelessWidget {  
  const MyApp({super.key});  
  
  @override  
  Widget build(BuildContext context) {  
    return MaterialApp(  
      debugShowCheckedModeBanner: false,  
      home: LoginPage(),  
    ); // MaterialApp  
  }  
}
```

4.5.4.4 Szerver-kapcsolat

A mobilalkalmazás-szerver kapcsolat fontossága számos előnyt és funkcionalitást jelent mind a felhasználók, mind pedig a fejlesztők számára. Ha az alkalmazás kapcsolódik a szerverhez, az ott létrehozott/törölt adatok nem

csak lokálisan, hanem az alkalmazás újraindítása után az adatbázisban is megmaradnak.

A mobilalkalmazás HTTP kérésekkel kommunikál a szerverrel, ezáltal az adatbázissal. Ehhez a http 1.2.0 dart csomagot használjuk.

```
http: ^1.2.0      import 'package:http/http.dart' as http;
```

A szerver ip-címét/domainjét egy globális szöveg mezőben tároljuk.

```
String httpLink = "https://borohfr.jmrtn.dev/";
```

A szervernek http kéréseket küldünk és ezeket szerveroldalon fogadjuk.

4.5.4.5 Bejelentkezés példa

Bejelentkezésnél a mobilalkalmazás a http kérés body-jában elküldi a beírt adatokat. A kérés válaszként egy kódot, egy body-t és egy headert ad vissza az alábbiak szerint:

- 200-as kód: Ilyenkor a bejelentkezés sikeres, a beírt adatok szerepelnek a felhasználók között. A headerben kapott felhasználói tokent eltároljuk, valamint a felhasználót beléptetjük. Később ezt a tokent használva tudunk a belépett felhasználóhoz tartozó kéréseket küldeni.
- 400-as kód: Ezt a kódot akkor kapjuk, ha a felhasználó valamelyik mezőt nem töltötte ki, ilyenkor ezt jelezzük a felhasználónak.
- 401-es kód: Ilyenkor a bejelentkezés sikertelen (pl. helytelen belépési adatok), a body-ban kapott választ írjuk ki a felhasználónak.

```
void signInUserIn()
{
  http.post(Uri.parse(globals.httpLink+'api/auth/login'),headers: {"Content-Type": "application/json"},
  body:jsonEncode({"username": usernamecontroller.text, "password": passwordcontroller.text, "persistent": true} ))
  .then((value) => {setState(() {
    globals.token = value.headers[HttpHeaders.authorizationHeader].toString();
    statuscode = value.statusCode;
    response = value.body;
  })),
  if(statuscode == 200)
  {
    Navigator.push(context, MaterialPageRoute(builder: (context) => const TaskPage())),
    resettext(),
  }
  else if(statuscode == 401)
  {
    loginerror(response)
  }
  else if(statuscode == 400)
  {
    loginerror("Nem töltötte ki az összes mezőt!")
  }
  else
  {
    loginerror("A rendszerben hiba lépett fel!")
  }
});
}
```

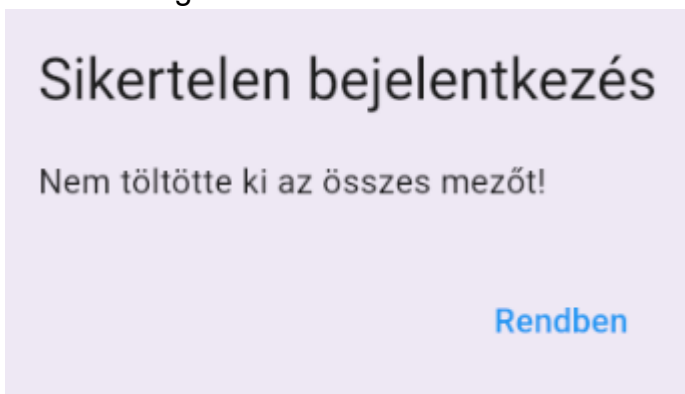
A felhasználó helytelen belépésekor egy felugró ablakban közöljük a hibát:

```
void loginerror(String toWrite)
{
  showDialog(
    context: context,
    builder: (context)
    {
      return LoginAlert(text: toWrite);
    }
  );
}
```

```
class LoginAlert extends StatelessWidget {
  final text;
  const LoginAlert({super.key, required this.text});

  @override
  Widget build(BuildContext context) {
    return AlertDialog(
      title: const Text("Sikertelen bejelentkezés"),
      content: Text(text),
      actions: [
        TextButton(
          style: TextButton.styleFrom(foregroundColor: Colors.blue),
          onPressed: (){
            Navigator.pop(context);
          },
          child: const Text("Rendben")), // TextButton
      ],
    ); // AlertDialog
  }
}
```

Példa a felugró ablakra:



4.5.4.6 Példák további http kérésekre

```
void fetchGroups()
{
    http.get(Uri.parse(globals.httpLink+'/api/groups'),
    headers: {"Content-Type": "application/json", "Authorization": globals.token})
    .then((value) => {setState(() {
        groups = jsonDecode(value.body).cast<Map<String, dynamic>>();
        actualGroup = groups[0];
        actualchatGroup = groups[0];
    }),}
    );
}
```

Itt a "globals.token" változóban eltárolt azonosítójú felhasználó csoportjait kérjük le, amelyet eltárolunk a groups nevű változóban.

```
void deleteTask(int index)
{
    http.get(Uri.parse(globals.httpLink+'/api/tasks/'+tiletasks[index]["id"].toString()+'/delete'),
    headers: {"Content-Type": "application/json", "Authorization": globals.token})
    .then((value) => {setState(() {
        fetchTasks();
    }),}
    );
}
```

A "deleteTask" metódus egy feladat törlését végzi el. A http kérésben átadjuk a törlendő feladat azonosítóját, amely ezáltal kitörli a megadott azonosítójú feladatot az adatbázisból.

4.6 Tesztelés

A szoftvertesztelés a szoftverminőség-biztosítás és így a szoftverfejlesztés részét képezi. A tesztelés egy rendszer vagy program kontrollált körülmények melletti futtatása, és az eredmények kiértékelése. A hagyományos megközelítés szerint a tesztelés célja az, hogy a fejlesztés során létrejövő hibákat minél korábban felfedezze, és ezzel csökkentse azok kijavításának költségeit.

A biztonság egy másik fontos terület, ami miatt a szoftverek tesztelésének lehetőségét nem szabad figyelmen kívül hagyni a fejlesztés során. Egy érzékeny felhasználói adatokkal gazdálkodó szoftver sebezhetősége szélsőséges esetben azt eredményezheti, hogy a felhasználók adatlopás áldozatává válhatnak.

Az időben elkezdett szoftvertesztelés gyorsabbá és olcsóbbá teszi a fejlesztési folyamatokat, a termék használata során végig fenntartja az ügyfélélegedettséget, vagyis a precízen tesztelt termék versenyelőnyt jelent a piacon, valamint üzemeltetése során az ügyfélpanaszok számát is csökkenti.

4.6.1 Backend tesztelés

A backend tesztelése rendkívül fontos, hiszen ez felelős a teljes rendszer működéséért, a validációért és a jogosultságkezelésért. A REST API esetén és a frontendhez tartozó kontrollerek esetén is fontos a végpontokat tesztelni,

lehetőleg az összes lehetséges inputtal, így biztosítva a helyes működést, valamint a tesztelés során megtalált hibákat még a szoftver kiadása előtt felfedezhetjük. Változtatások esetén pedig ellenőrizhetjük, hogy nem okoztunk-e hibát.

4.6.2 API tesztelés példa

Api teszteléshez használhatjuk például a projektben használt SwaggerUI felületet.

Vizsgáljuk meg most a bejelentkezéshez (JWT token megszerzéséhez) használt végpontot.

POST `/api/auth/login`

Parameters

No parameters

Request body

Example Value | Schema

```
{
  "username": "string",
  "password": "string",
  "persistent": true
}
```

Láthatjuk, hogy ez egy HTTP POST metódust használó, a szükséges adatokat JSON formátumban váró végpont.

4.6.2.1 1. *teszteset: helyes adatokkal*


Az első tesztesetben ellenőrizzük, hogy a rendszer helyes bemenetre helyes választ küld-e.

A Swagger „Try it out” funkcióját használva elküldjük az alábbi, helyes adatokat tartalmazó (létezik felhasználó ezekkel a bejelentkezési adatokkal) JSON karakterláncot:

application/json

Az alábbi választ kapjuk:

Code	Details
------	---------

 Download

```
authorization: Bearer  
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZWl1cy54bWVud2F1bm9yZy93cy8yMDAxLzA1LTkzW50AHR5LS2N5YmlydC9uYW11IjoiYXBpdGZvdCIsImh0dHA6Ly9yZ2h1bWVfLnhtbHNvYXub3JnL3dzLzIwMDUwMDUvaHRlb3RpdHkvY2xhaW1zL2V1YW11YXRkcmlzcy616Inr1c3RAVM9yb2hmci5qbXJ0bi5kZXylClCjodHRwOi8vc2NoZWl1cy5tYStahNyB3NvZnQuY29tL3dzLzIwMDgvdMDYvaHRlb3RpdHkvY2xhaW1zL3ZvbGU0IjV2V2YyIiwiaWF0IjoxZjczSG1wNS11MDkgOGLTQxY2Y1TM5my1NDhhMzMxd2cwZGUiLC1eHaiojE3MT1SNjAyMzV9.6P18Rog4YInag2iFzmw_zWrO9wQEfm9ni0o6_8o8NdK  
content-type: text/plain; charset=utf-8  
date: Wed, 13 Mar 2024 22:17:14 GMT  
server: Kestrel  
transfer-encoding: chunked
```

A kapott JWT-t dekódolva (a szerver által használt kulcs birtokában) az alábbi információt olvashatjuk ki:

HEADER: ALGORITHM & TOKEN TYPE
<pre>{ "alg": "HS256", "typ": "JWT" }</pre>
PAYLOAD: DATA
<pre>{ "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/ name": "apitest", "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/ emailaddress": "test@borohfr.jmrtn.dev", "http://schemas.microsoft.com/ws/2008/06/identity/claim s/role": "User", "Id": "1f79db05-b084-41ca-a393-f48b331d70de", "exp": 1712960235 }</pre> <div>Sat Apr 13 2024 00:17:15 GMT+0200 (közép-európai nyári idő)</div>

Az azonosítás tehát sikeres.

„persistent” : false beállítás mellett is sikeres a kérés, ilyenkor hamarabb jár le a token érvényessége:

```
{
  "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name": "apitest",
  "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress": "test@borohfr.jmrtn.dev",
  "http://schemas.microsoft.com/ws/2008/06/identity/claims/role": "User",
  "Id": "1f79db05-b084-41ca-a393-f48b331d70de",
  "exp": 1710371908
}
```

The screenshot displays a REST client interface with a light blue header bar. The top right corner shows the selected format as "application/json".

The main area is divided into two sections:

- Request:** Shows a JSON body with the following fields:

```
{  "username": "apitest",  "password": "apitest1234"}
```
- Response:** Displays the status code "200" and the label "Server response". Below this, there are tabs for "Code" and "Details". The "Code" tab is active, showing the raw response body:

```
Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy54bWxzczBmLm9yZy93cy8yMDAxLzA1L2lkZW50aXR5L2NsYWltcy9uYm1lIjoieXBpdGVzdCIsImh0dHA6Ly9yZ2hlbmFzLnhtbHNvYXAub3JnL3dzLzIwMDUvMDUvaWR1bnRpdHkvY2xhaW1zL2VtYWlsYWlkcmVzcyI6InRlc3RAYm9yb2hmci5qbXQ0bi5kZXilCjodHRwOi8vc2NoZW1hcy5taW9yb3NvZ29tL3dzLzIwMDUvMDUvaWR1bnRpdHkvY2xhaW1zL3JvbGUoIjVcZ2VyIiwiaSwQioiIjVjc2ZGIwNS1iMDg0LTQxYTZETmM5My1mNDhiMTAzNTd9.WTtm_LdzbZ2GMmfvgtxTwAlaTtzi2K64Qy-1ER8U
```

To the right of the response body is a "Download" button.

Below the response body, the "Response headers" section is visible, listing several headers:

- authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy54bWxzczBmLm9yZy93cy8yMDAxLzA1L2lkZW50aXR5L2NsYWltcy9uYm1lIjoieXBpdGVzdCIsImh0dHA6Ly9yZ2hlbmFzLnhtbHNvYXAub3JnL3dzLzIwMDUvMDUvaWR1bnRpdHkvY2xhaW1zL2VtYWlsYWlkcmVzcyI6InRlc3RAYm9yb2hmci5qbXQ0bi5kZXilCjodHRwOi8vc2NoZW1hcy5taW9yb3NvZ29tL3dzLzIwMDUvMDUvaWR1bnRpdHkvY2xhaW1zL3JvbGUoIjVcZ2VyIiwiaSwQioiIjVjc2ZGIwNS1iMDg0LTQxYTZETmM5My1mNDhiMTAzNTd9.WTtm_LdzbZ2GMmfvgtxTwAlaTtzi2K64Qy-1ER8U
- content-type: text/plain; charset=utf-8
- date: Wed, 13 Mar 2024 22:25:57 GMT
- server: Kestrel
- transfer-encoding: chunked

4.6.2.2 2. tesztet: helytelen adatokkal

Az első esetben hibás jelszóval teszteljük a végpontot:

Request body

application/json

```
{
  "username": "apitest",
  "password": "rosszjelszo",
  "persistent": false
}
```

Server response

Code

Details


401

Undocumented

Error: Unauthorized

Response body

A megadott adatokkal nem található felhasználó.

 Download

Response headers

content-type: text/plain; charset=utf-8
date: Wed, 13 Mar 2024 22:20:09 GMT
server: Kestrel
transfer-encoding: chunked

Aztán hibás felhasználónévvel:

Request body

application/json

```
{
  "username": "nincsilyenfelhasznalo",
  "password": "rosszjelszo",
  "persistent": false
}
```

Server response

Code

Details


401

Undocumented

Error: Unauthorized

Response body

A megadott adatokkal nem található felhasználó.

 Download

Response headers

content-type: text/plain; charset=utf-8
date: Wed, 13 Mar 2024 22:21:53 GMT
server: Kestrel
transfer-encoding: chunked

4.6.2.3 3. tesztet: hiányos adatokkal

Először nem adunk meg jelszót:

Request body application/json


```
{  "username": "apitest"}
```

Server response

Code	Details
400 <i>Undocumented</i>	Error: Bad Request

Response body

```
{  "type": "https://tools.ietf.org/html/rfc9110#section-15.5.1",  "title": "One or more validation errors occurred.",  "status": 400,  "errors": {    "$": [      "JSON deserialization for type 'BoroHFR.Controllers.Api.LoginData' was missing required properties, including the following: password"    ],    "data": [      "The data field is required."    ]  },  "traceId": "00-3c89346e37c4ab5caffd7624d1895715-ee87b878c37e7e54-00"}
```

 Download

Response headers

```
content-type: application/problem+json; charset=utf-8
date: Wed, 13 Mar 2024 22:29:28 GMT
server: Kestrel
transfer-encoding: chunked
```

A második esetben nem adunk meg felhasználónevet:

Request body application/json

```
{  "password": "apitest1234"}
```

Server response

Code	Details
400 <i>Undocumented</i>	Error: Bad Request

Response body

```
{
  "type": "https://tools.ietf.org/html/rfc9110#section-15.5.1",
  "title": "One or more validation errors occurred.",
  "status": 400,
  "errors": {
    "$": [
      "JSON deserialization for type 'BoroHFR.Controllers.Api.LoginData' was missing required properties, including the following: username"
    ],
    "data": [
      "The data field is required."
    ]
  },
  "traceId": "00-8c3ff00cc900579a7b621bdc9e77ca3-e2198ec352d4addb-00"
}
```

Response headers

```
content-type: application/problem+json; charset=utf-8
date: Wed, 13 Mar 2024 22:33:08 GMT
server: Kestrel
transfer-encoding: chunked
```

Ezzel leellenőriztük a végpont validációját is: a felhasználónév és a jelszó megadása kötelező.

4.6.2.4 4. tesztet: megerősítetlen e-mail cím

Request body application/json

```
{
  "username": "apitest",
  "password": "apitest1234"
}
```

Server response

Code	Details
401 <i>Undocumented</i>	Error: Unauthorized

Response body

```
Erősítsd meg az e-mail fiókod!
```

Response headers

```
content-type: text/plain; charset=utf-8
date: Thu, 14 Mar 2024 15:59:14 GMT
server: Kestrel
transfer-encoding: chunked
```

Látható, hogy a szerver nem engedi, hogy a felhasználó belépjen, amíg nem erősítette meg az e-mail címét.

4.6.3 Kontroller tesztelés

A web frontendhez tartozó kontrollerek esetén is nagyon fontos a tesztelés. 2 kontrollerhez (AuthenticationController és SettingsController) készültek unit tesztek. A unit tesztek lényege, hogy a program bizonyos egységeinek

működését egy elkülönített tesztkörnyezetben, programatikusan ellenőrizzük, adott feltételek alapján. Például itt is elvárhatjuk helyes bejelentkezési adatok esetén a sikeres bejelentkezést, hibásak esetén a hiba jelzését. A unit tesztekben a szükséges szolgáltatások szimulálásához a Moq csomagot használtuk. A tesztek a BoroHFR.Tests projektben találhatóak, és az eredeti projekt struktúráját követik. A teszt projekt az XUnit keretrendszert használja.

4.6.4 Kontroller tesztelés példa

A példában az AuthenticationController Login metódusának unit tesztjét mutatjuk be.

Először létrehozzuk és konfiguráljuk a kontrollerhez használt kamu szolgáltatásokat, valamint magát a kontrollert.

```
public AuthenticationController()
{
    var dbContextMock = new Mock<BoroHfrDbContext>();
    dbContextMock.Setup();

    var context = new Mock<HttpContext>();
    var request = new Mock<HttpRequest>();
    request.Setup(x => x.Scheme).Returns(value: "https");
    request.Setup(x => x.Host).Returns(HostString.FromUriComponent("localhost:5000"));
    context
        .Setup(c => c.Request)
        .Returns(request.Object);

    var emailServiceMock = new Mock<EmailService>();
    emailServiceMock.Setup(x => x.Enqueue(It.IsAny<Email>()));

    var loggerMock = new Mock<ILogger<BoroHFR.Controllers.AuthenticationController>>();

    var configMock = new Mock<IConfiguration>();

    controller = new BoroHFR.Controllers.AuthenticationController(dbContextMock.Object,
        emailServiceMock.Object, loggerMock.Object, configMock.Object);
    controller.ControllerContext = new ControllerContext() { HttpContext = context.Object };
    controller.Url = new UrlHelper();
}
```

(Ez az AuthenticationController osztály a BoroHFR.Tests projektben, külön mappában található, nem azonos a kontrollerrel.)

A **dbContextMock.Setup();** extension method létrehozza és tesztadatokkal tölti fel a kamu DbContext DbSet tulajdonságait, mely így már használható a kontroller által.

```
public static void Setup(this Mock<BoroHfrDbContext> mock)
{
    mock.Setup(x => x.Users).ReturnsDbSet(ExampleUsers);
    mock.Setup(x => x.Classes).ReturnsDbSet(ExampleClasses);
    mock.Setup(x => x.EmailConfirmationTokens).ReturnsDbSet(ExampleEmailConfirmationTokens);
    mock.Setup(x => x.InviteTokens).ReturnsDbSet(ExampleInviteTokens);
    mock.Setup(x => x.PasswordResetTokens).ReturnsDbSet(ExamplePasswordResetTokens);
    mock.Setup(x => x.Groups).ReturnsDbSet(ExampleGroups);
    mock.Setup(x => x.Subjects).ReturnsDbSet(ExampleSubjects);
    mock.Setup(x => x.Files).ReturnsDbSet(Array.Empty<Models.File>());
    mock.Setup(x => x.Events).ReturnsDbSet(ExampleEvents);
}
```


Az UriHelper osztály implementálja az IUriHelper interfészt, egyetlen hívható függvénye:

```
public string? Action(UrlActionContext actionContext)
{
    return "/";
}
```

Ennek célja pusztán a rendszer valós szolgáltatások nélkül való működésre bírása.

4.6.4.1 1. teszteset: nézet

```
[Fact]
| 0 references
public void Login_Get_View()
{
    var result = controller.Login();
    Assert.IsType<ViewResult>(result);
}
```

Ellenőrizzük, hogy a kontroller ViewResult típust ad-e vissza a bejelentkező felület lekérésekor.

4.6.4.2 2. teszteset: belépés helyes adatokkal

```
[Fact]
| 0 references
public async Task Login_Success()
{
    var result = await controller
        .Login(model: new LoginViewModel()
            { Username = "test", Password = "apitest1234" },
            returnUrl: null);
    Assert.IsType<SignInResult>(result);
}
```

Ellenőrizzük, hogy helyes bejelentkezési adatok megadásakor SignInResult eredményt kapunk-e.

4.6.4.3 3. teszteset: validációs hiba

```
[Fact]
| 0 references
public async Task Login_ValidationError()
{
    controller.ModelState.AddModelError(key: "a", errorMessage: "b");
    var result = await controller.Login(
        model: new LoginViewModel() { },
        returnUrl: null);
    Assert.IsType<ViewResult>(result);
    var view = (result as ViewResult)!;
    Assert.IsType<LoginViewModel>(view.Model);
    var model = (view.Model as LoginViewModel)!;
    Assert.False(model.BadUsername);
    Assert.False(model.BadPassword);
}
```

Ellenőrizzük, hogy szimulált validációs hiba esetén (a tesztek futtatásakor nem elérhető a validációs szolgáltatás, ezért a ModelState-hez hozzáadunk egy kamu

hibát) a kontroller visszairányít-e a bejelentkező nézetre, valamint nem kapunk hibás jelszóra vagy felhasználónévre utaló jelzést. Nem adunk meg jelszót és felhasználónevet, a hibás felhasználónévre vagy jelszóra utaló hibajelzés viszont nem a helyes működés: a kontrollernek már a bejelentkezési adatok ellenőrzése esetén fel kell ismernie a hibát.

4.6.4.4 4. teszteset: hibás jelszó

```
[Fact]
0 references
public async Task Login_BadPassword_Failure()
{
    var result = await controller.Login(
        model: new LoginViewModel() {
            Username = "test",
            Password = "testpassword123" },
        returnUrl: null);
    Assert.IsType<ViewResult>(result);
    var view = (result as ViewResult)!;
    Assert.Null(view.ViewName);
    Assert.IsType<LoginViewModel>(view.Model);
    var model = (view.Model as LoginViewModel)!;
    Assert.True(model.BadPassword);
}
```

Ellenőrizzük, hogy hibás jelszó esetén a kontroller a bejelentkezési felületre irányít-e, és jelzi-e a jelszó hibás voltát.

4.6.4.5 5. teszteset: hibás felhasználónév

```
[Fact]
0 references
public async Task Login_BadUsername_Failure()
{
    var result = await controller.Login(
        model: new LoginViewModel() {
            Username = "bad_username",
            Password = "apitest1234" },
        returnUrl: null);
    Assert.IsType<ViewResult>(result);
    var view = (result as ViewResult)!;
    Assert.Null(view.ViewName);
    Assert.IsType<LoginViewModel>(view.Model);
    var model = (view.Model as LoginViewModel)!;
    Assert.True(model.BadUsername);
}
```

Ellenőrizzük, hogy hibás felhasználónév esetén a kontroller a bejelentkezési felületre irányít-e, és jelzi-e a felhasználónév hibás voltát.

4.6.4.6 6. teszteset: megerősítetlen e-mail cím

```
[Fact]
| 0 references
public async Task Login_EmailNotVerified_View()
{
    var result = await controller.Login(
        model: new LoginViewModel() {
            Username = "test2",
            Password = "apitest1234" },
        returnUrl: null);
    Assert.IsType<ViewResult>(result);
    var view = (result as ViewResult)!;
    Assert.Equal(expected: "EmailConfirmationNotification", view.ViewName);
    Assert.IsType<EmailConfirmationNotificationViewModel>(view.Model);
}
```

Ellenőrizzük, hogy ha a bejelentkezési adatok ugyan helyesek, de a felhasználó nem erősítette meg az e-mail címét, a kontroller a cím megerősítését kérő nézetet adja vissza eredményként.

A tesztek lefuttatva az összes sikeres:

✓ Login_BadPassword_Failure	148 ms
✓ Login_BadUsername_Failure	1 ms
✓ Login_EmailNotVerified_View	149 ms
✓ Login_Get_View	1 ms
✓ Login_Success	149 ms
✓ Login_ValidationError	1 ms
✓ Logout_Success	1 ms

4.6.5 Frontend tesztelés

A tesztelés az egyik kulcsfontosságú része egy szoftver/alkalmazás fejlesztésének, hiszen ezen folyamat által derülhetnek ki hibák, problémák, esetleg hiányosságok.

A tesztelés folyamán végigveszünk minden esetet, mellyel egy alrendszer vagy funkció szembenézhet, a legalapvetőbb helyzetektől a legkomplikáltabbakig.

Tesztelőként főleg a rendszert vizsgáljuk, miképpen reagál a különböző bemenetekre.

A megfelelő és hatékony teszteléshez ismerni kell a rendszer működését, hiszen hiba esetén tudnunk kell, hol akadt el a teszteset, hogy máris javítani tudjuk. Ehhez viszont a teszteseteket helyes sorrendben kell végignézni, hiszen így fogjuk tudni, hogy a rendszerben egy folyamat meddig működik jól, és ha elakad, nem kell keresni, hogy hol a probléma, azonnal rátalálunk.

Frontend tesztelésnél viszont elég a fekete dobozos módszer, ahol a tesztelő nem ismeri a rendszert, csak annyit tud, hogy milyen bemenetekre hogyan kellene viselkednie. Ekkor a tesztelő csak megnézi az összes esetet, majd leírja, mely esetek voltak hibásak, a fejlesztőknek ezzel kell kezdeniük valamit.

A következő példában átnézzük, hogyan kell tesztelni egy bejelentkezői felületet.

4.6.6 Frontend tesztelés példa

A bejelentkező felületen lévő mezők helytelen kitöltését jelzi a rendszer validációs üzenetekkel. Tesztelés szempontjából érdemes először kipróbálni, hogy az üresen hagyott mezőkre mit reagál a rendszer.

Two screenshots of a login form. The left screenshot shows the 'Felhasználónév:' field with a blue border and a tooltip message: 'Kérjük, töltsse ki ezt a mezőt.' The right screenshot shows the 'Jelszó:' field with a blue border and a tooltip message: 'Kérjük, töltsse ki ezt a mezőt.'

Ha minden ki lett töltve, akkor érdemes a rendszer adatellenőrzésének helyességét vizsgálni. Lehet-e, hogy valamilyen kivételt nem kezel a bejelentkezésnél?

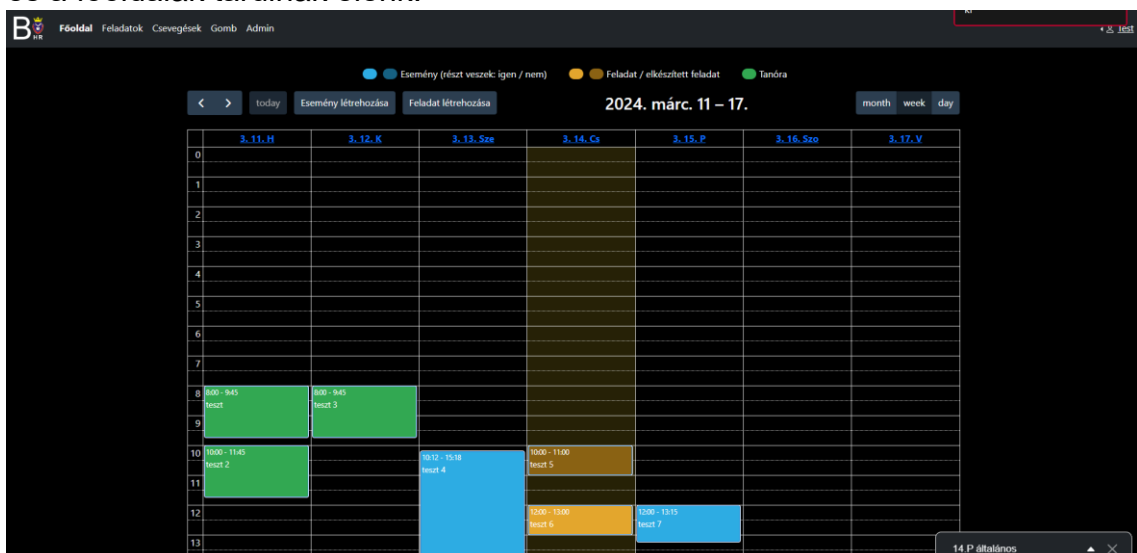
Three screenshots of the login form showing validation errors. The first shows a red error message: 'A jelszavak minimum 8 karakterből állnak.' The second shows a red error message: 'Hibás jelszó.' The third shows a red error message: 'Érvénytelen felhasználónév.'

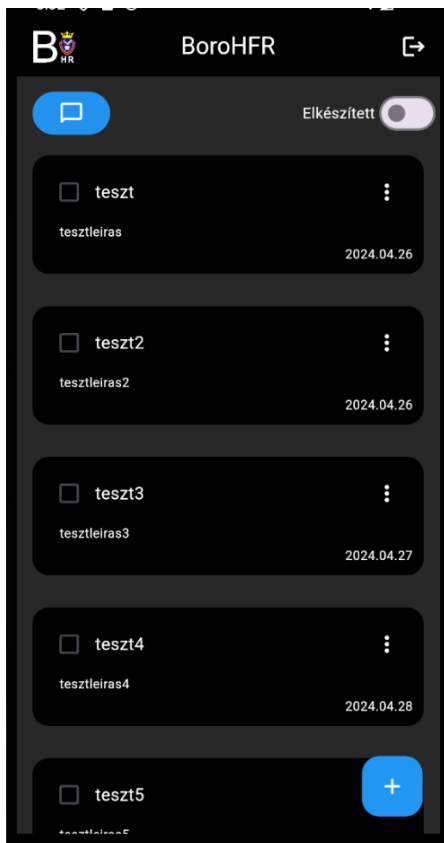
Tesztelés után arra lehetünk figyelmesek, hogy ha hibás vagy kevesebb, mint 8 karakterből álló jelszót adunk meg, vagy nem ismert felhasználónévvel találkozunk, a rendszer újratölti a nézetet a beírt felhasználónévvel, és a hibás mezőnél lévő validációs üzenettel.

A mobilalkalmazásban a mezők hibás kitöltését a rendszer szintén felugró figyelmeztető üzenetekkel jelzi.

<p>Felhasználónév:</p> <input type="text"/>	<p>Sikertelen bejelentkezés</p> <p>Nem töltötte ki az összes mezőt!</p> <p>Rendben</p>
<p>Felhasználónév:</p> <input type="text" value="test"/>	<p>Sikertelen bejelentkezés</p> <p>Nem töltötte ki az összes mezőt!</p> <p>Rendben</p>
<p>Felhasználónév:</p> <input type="text" value="test"/>	<p>Sikertelen bejelentkezés</p> <p>A megadott adatokkal nem található felhasználó.</p> <p>Rendben</p>

Helyes bemeneti paraméterek esetén a bejelentkeztetés sikeresen végbe megy, és a főoldalak tárulnak elénk.





A bejelentkezési felület így makulátlannak bizonyult, kifogásolhatatlanul működik. Fontos azt is ellenőrizni, hogy a rendszer nem enged azonosítatlan felhasználókat belépni a fő kezelőfelületre.

Name	Method	Status	Type
borohfr.jmrtn.dev	GET	302	document / Redirect
login?ReturnUrl=%2F	GET	302	document / Redirect
login?ReturnUrl=%2F	GET	200	document

Láthatjuk, hogy a főoldalról az azonosítatlan felhasználó átirányításra kerül a bejelentkezési felületre.

4.7 Továbbfejlesztési lehetőségek

4.7.1 Felhasználói profil

A felhasználókhöz kiterjedtebb profil tartozhatna: beállíthatnának képet, leírást magukról, részletesebb és más jellegű statisztikát lehetne gyűjteni hozzájuk, amit például mérőszámok és jelvények megszerzésére lehetne használni.

A közösségen belüli aktivitást akár külön rangokkal is lehetne jutalmazni, mely vizuális megkülönböztetést, vagy akár plusz funkciókat is jelenthetne.

Érdemes lenne lehetővé tenni az adminisztrátorok számára a felhasználók felfüggesztését vagy kitiltását, mely nem lenne azonos a rendszeradminisztrátor számára már elérhető törlési lehetőséggel. A felhasználó ilyenkor értesítést kapna, megszakadnának az aktív munkamenetei, viszont az általa létrehozott tartalmak nem kerülnének törlésre és később visszaállítható lenne a fiókja.

4.7.2 Fájlmenedzsment

A feltöltött fájlok esetében logikus fejlesztési lépés lehet, hogy a felhasználó áttekinthesse és törölhesse a feltöltött fájljait. Adminisztrátor szinten szintén érdemes lenne egy hasonló eszközt beépíteni a rendszerbe.

A tárhely korlátozható lehetne osztály és felhasználói szinten is.

4.7.3 Hirdetőtábla

Létre lehetne hozni egy külön felületet, ahol nem kifejezetten tanulmányi jellegű és csoportokhoz köthető, hanem általánosabb, közérdekű információ kerülhetne megjelenítésre. Akár közérdekű hirdetések elhelyezésére is lehetne lehetőség (pl. valaki ismer egy jó magántanárt, akciós valami hasznos dolog stb.)

4.7.4 Értesítések

A rendszert ki lehetne egészíteni különböző értesítésekkel, ebbe beleértve mobilon a push értesítéseket, valamint az e-mail értesítéseket. Az értesítések a közelgő feladatokra és eseményekre való emlékeztetőül szolgálna, természetesen csak a felhasználó által engedélyezett esetekben.

Az adminisztrátorok fontos esetekben küldhetnének értesítéseket és e-maileket a felhasználóknak.

4.7.5 Chat

A kommunikációs rendszert érdemes lenne szerkesztési és törlési lehetőséggel kiegészíteni, lehetővé tenni a fontos üzenetek kitűzését, valamint egy egyszerűbb formázási lehetőség is hasznos lenne (pl. Markdown).

A küldött fájlok esetén hasznos lenne előnézet megvalósítása.

Szintén hasznos lehetne a privát és a közvetlen üzenetküldés lehetőségét megvalósítani, melyhez barátrendszer is társulhatna.

4.7.6 Naptár

A főoldal naptár nézetéből az események és feladatok exportálása megvalósítható lenne, ami lehetővé tenné, hogy a felhasználók az általuk preferált naptár alkalmazásban láthassák a fontos információkat.

4.7.7 Csoportok

A csoportokban hasznos lehetne a csoportszintű adminisztráció bevezetése: lennének csoportszintű adminisztrátorok is, melyek csak az adott csoportot felügyelnék. Ebben az esetben bevezethetők lennének a privát csoportok, melyekbe meghívásos vagy jelentkezéses módon lehetne belépni. Ha minden csoportnak lennének adminisztrátorai, be lehetne vezetni a felhasználók számára elérhető műveletek korlátozását. (pl. a felhasználó ne tudjon eseményt létrehozni a csoportban, csak az adminisztrátor)

4.7.8 Privát bejegyzések

A felhasználóknak lehetőségük lenne csak általuk kezelt és megtekinthető feladatok, események és tanórák létrehozására. ha van barátrendszer, akár velük is meg lehetne osztani bizonyosakat.

4.7.9 Intézmény szintű használat

A rendszer viszonylag egyszerűen átalakítható lenne intézményi adminisztrációs rendszerré, aminek nincs külső szolgáltató felé függősége, nyílt és szabadon használható, valamint kevés erőforrást igényel. Nyilván nem az elektronikus naplót váltaná le, hanem egy egységes felületet biztosítana a tanárok számára, akik adminisztrálhatnák a saját csoportjaikat (csak ők hozhatnának létre feladatokat, eseményeket).

Ez szembemegy az eredeti elképzeléssel, de igény esetén ilyen célra is alkalmas lenne a rendszer.

4.7.10 Rendszeradminisztrátor biztonság

A rendszeradminisztrátor bejelentkezése a megadott e-mail címhez lehetne kötve. Belépéskor kapna egy biztonsági kódot e-mailben, és csak a kód megadása után engedné be a rendszer. (Kétfaktoros autentikáció)

4.7.11 Mobilalkalmazás naptár

A főoldalon lehetne egy navigációs sáv, amelyről a felhasználó szabadon tudna váltani egy másik oldalra, ahol a feladatokat naptárban elhelyezve tudjuk megtekinteni, mint a webalkalmazásban.

4.7.12 Biometrikus azonosítás

A mobiltelefonos alkalmazásban bejelentkezéskor az adatok megadása helyett/mellett lehetne a telefon saját biometrikus azonosítórendszerét használni (például ujjlenyomat, arcfelismerés).

4.7.13 Webalkalmazásban elérhető funkciók

A telefonos alkalmazásban elérhető lenne az összes további olyan funkció, amely a webalkalmazásban elérhető, viszont a mobilappban nem. Ilyen például az admin felület, vagy a csoportok felvétele.

5. Felhasználói dokumentáció

5.1 A program rövid ismertetése

A BoroHFR egy web- és mobilalkalmazás, mellyel kényelmesen és hatékonyan lehet számontartani eseményeinket és feladatainkat. A számos online naptártól jelentősen eltér, hisz célközönsége a tanulók és hallgatók. Az alkalmazás idézőjeles célja felváltani a papíralapú órarendet és leckeüzeteket egy webalkalmazásra, amely így bárhol elérhető elektronikai eszközökön. A webalkalmazásban létrehozhat bármely személy eseményt vagy házi feladatot, melyet a rendszer megjelenít társainak is valós időben. Továbbá minden feladathoz, eseményhez és csoporthoz tartozik csevegési felület, amelyben akár fájlokat is fel lehet tölteni.

5.2 A szerver futtatása (helyben)

Használat előtt érdemes lehet lefuttatni a teszteket, melyek a BoroHFR.Tests mappában találhatók, itt kell kiadni a 'dotnet test' parancsot. Ehhez .NET 8 SDK szükséges.

5.2.1 Docker (ajánlott)

A projekt tartalmaz egy Dockerfile és egy compose.yaml nevű fájlt. A compose.yaml fájl tartalmazza a magát a projektet, illetve egy Alpine Linux alapú, MariaDB-t futtató image-et, valamint 2 docker volume-ot az adatbázis állapotának tárolásához és a szerver konfigurációját valamint a feltöltött fájlokat tartalmazó mappához. Haladó felhasználók használhatják külön is a Dockerfile-t, ami akkor hasznos, ha már van egy létező MariaDB vagy MySQL adatbázisszerver, és azt szeretné használni a beüzemelő.

A konfigurációs fájlok lehetővé teszik a függőségek automatikus letöltését, az alkalmazás lefordítását és futtatását Dockerben. Ehhez szükséges, hogy a Docker telepítve legyen a céleszközön.

Első lépés a forráskód letöltése, melyhez a Github tárolót ajánljuk.

Ezután következhet a konfiguráció elvégzése a compose.yaml fájlban. (nem szükséges a működéshez) Itt főként a portszám beállítása lehet fontos, ami alapértelmezetten 5005.

```
server:
  build:
    context: .
    target: final
  ports:
    - 5005:8080
  depends_on:
    db:
      condition: service_healthy
  restart: always
  networks:
    - borohfr
  volumes:
    - storage:/app/storage
  environment:
    - ASPNETCORE_ENVIRONMENT=Docker
```

A bal oldalon lévő szám a külső, míg a kettőspont jobb oldalán lévő a belső, a program által használt portot jelenti. Az ASP.NET Core alapértelmezett portja 8080 Docker konténerben. A környezeti változók segítségével módosítható az ASP.NET Core konfigurációja. Ehhez segítség ezen a weboldalon található: <https://learn.microsoft.com/en-us/aspnet/core/fundamentals/host/web-host?view=aspnetcore-8.0#host-configuration-values>.

Ezután a 'docker compose build' parancs kiadása szükséges.

A futtatáshoz a 'docker compose up' parancs adható ki. A kettő kombinálható: 'docker compose up --build', viszont felesleges minden futtatáskor lefordítani a teljes projektet.

Systemd-t használó Linux disztribúció esetén érdemes létrehozni egy service fájlt, például:

```
[Unit]
Description=BoroHFR frontend and API
After=network.target

[Service]
Type=oneshot
RemainAfterExit=yes
User=root
#a docker kezelése miatt root
WorkingDirectory=<BoroHFR root mappa>
ExecStart=/usr/bin/docker compose up -d
ExecStop=/usr/bin/docker compose down
#a docker elérési útvonala különbözhet

[Install]
WantedBy=multi-user.target
```

5.2.2 Standalone (forráskódból)

A fordításhoz .NET 8 SDK-ra van szükség. <https://dot.net>

Ebben a módban a beüzemelőre van bízva az adatbázis konfigurálása. A forráskód letöltése után a fordításhoz a 'dotnet publish -c Release -o publish' parancs használható, ami a lehető leghatékonyabb kimenetet adja. Ezután a kimeneti mappa tartalma bárhová áthelyezhető, és a 'dotnet BoroHFR.dll' paranccsal futtatható a program.

5.2.3 Standalone

A program futtatásához ASP.NET Core Runtime 8 környezetre van szükség. <https://dot.net>

Az adatbázis konfigurálása ebben az esetben is a beüzemelő feladata.

A lefordított, futtatható állomány letöltése után a program a 'dotnet BoroHFR.dll' paranccsal futtatható.

5.3 HTTPS konfigurálása

A rendszer úgy van megtervezve, hogy egy reverse proxy (pl. Nginx vagy Apache) mögött fut, ezért kizárólag HTTP protokollt használ.

Minta konfiguráció Nginx használata esetén:

```
upstream borohfr {
    zone upstreams 64K;
    server 127.0.0.1:5005 max_fails=1 fail_timeout=2s;
    keepalive 2;
}

server {
    server_name <domainnév>;

    location / {
        proxy_pass http://borohfr;
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-For $remote_addr;
        proxy_set_header X-Forwarded-Proto $scheme;

        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
    }
}
```

Ezután történhet a HTTPS beállítása. Ehhez a Certbot nevű segédeszközt ajánljuk, amely képes tanúsítványokat generálni, valamint Nginx (+ pár másik szoftver) használata esetén képes a konfigurációs fájlokat is módosítani a tanúsítványok használatához. A Certbot a tanúsítványokat automatikusan megújítja, mielőtt azok lejárnának. <https://certbot.eff.org/>

Egy kész konfigurációs fájl:

```
upstream borohfr {
    zone upstreams 64K;
    server 127.0.0.1:5005 max_fails=1 fail_timeout=2s;
    keepalive 2;
}

server {
    server_name borohfr.jmrt.n.dev;

    location / {
        proxy_pass http://borohfr;
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-For $remote_addr;
        proxy_set_header X-Forwarded-Proto $scheme;

        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
    }

    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/borohfr.jmrt.n.dev/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/borohfr.jmrt.n.dev/privkey.pem; # managed by Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot
}

server {
    if ($host = borohfr.jmrt.n.dev) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

    listen 80;
    server_name borohfr.jmrt.n.dev;
    return 404; # managed by Certbot
}
```

5.3.1 Az alkalmazás konfigurálása az első indításkor

Az alkalmazás grafikus konfigurációs felülettel rendelkezik. Első indításkor a szerver konfigurációs módban indul el.

BoroHFR kezdeti beállítások

A következőkben a szerver kezdeti beállításait kell megadni. Egy MySQL/MariaDB adatbázisra és egy SMTP kiszolgálóra lesz szükség.

Következő

5.3.1.1 Adatbázis

Adatbázis adatai

MySQL vagy MariaDB adatbázis használható. A megadott felhasználónak módosító jogokkal is rendelkeznie kell.

Szerver:

Port:

Felhasználónév:

Jelszó:

Adatbázis:

Mentés

Egy megfelelő MariaDB vagy MySQL adatbázisszerverre van szükség, melyen a megadott felhasználó rendelkezik táblaszerkezet-módosító jogosultsággal a megadott nevű adatbázisban. Erre azért van szükség, mert a program önállóan építi fel a szükséges adatbázisszerkezetet.

A mentés gomb megnyomásakor a szerver ellenőrzi, hogy tud-e csatlakozni a megadott adatokkal. Amennyiben nem, úgy nem enged továbblépni.

5.3.1.2 Rendszeradminisztrátor adatai

Rendszeradminisztrátor adatai:

E-mail cím:

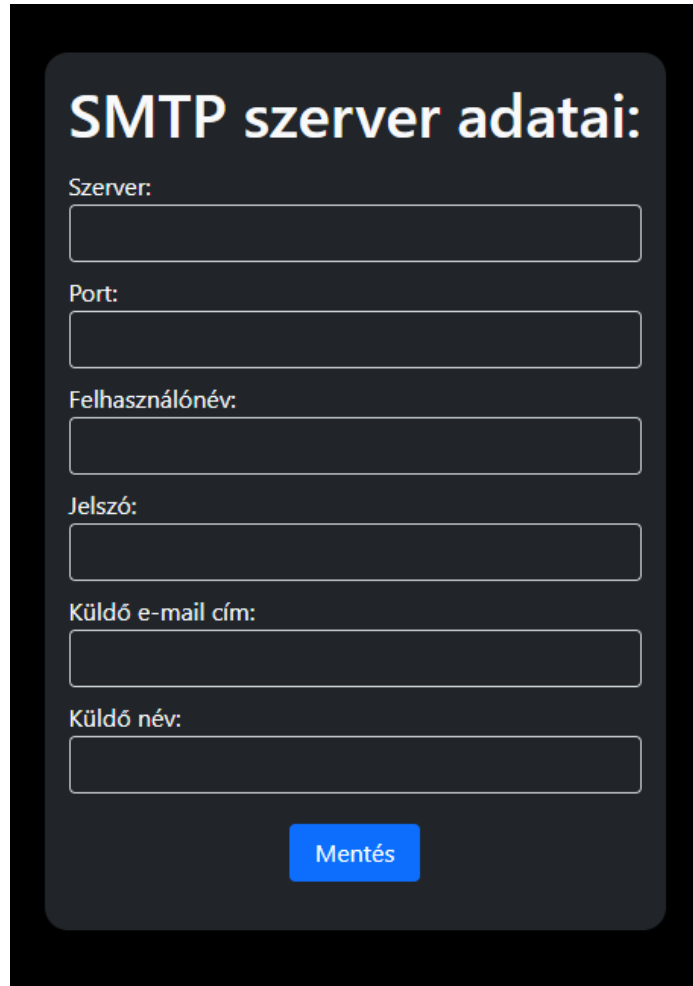
Jelszó:

Jelszó megerősítése:

Mentés

Egy darab rendszeradminisztrátor lehet a rendszerben. A rendszeradminisztrátor jelszava legalább 12 karakter kell, hogy legyen. A megadott e-mail cím nincs ellenőrizve, viszont ezzel kell belépnie a rendszeradminisztrátornak. A rendszeradminisztrátor felülete a /sysadmin útvonalon érhető el.

5.3.1.3 SMTP szerver beállításai

The image shows a dark-themed web form titled "SMTP szerver adatai:". Below the title are six input fields, each with a label to its left: "Szerver:", "Port:", "Felhasználónév:", "Jelszó:", "Küldő e-mail cím:", and "Küldő név:". At the bottom of the form is a blue button labeled "Mentés".

SMTP szerver adatai:

Szerver:

Port:

Felhasználónév:

Jelszó:

Küldő e-mail cím:

Küldő név:

Mentés

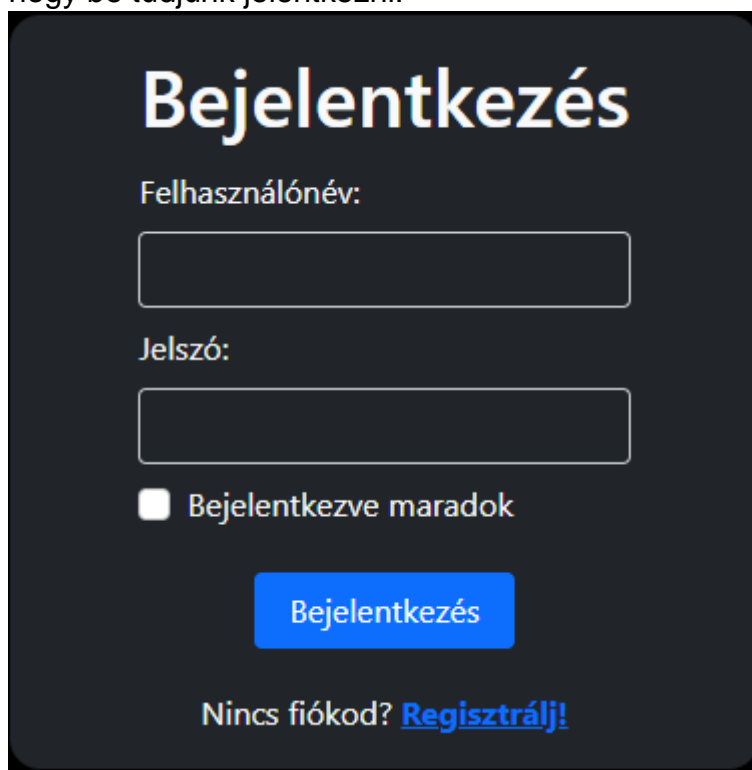
Az alkalmazás használatához szükség van egy SMTP szerverre. A MailerSend szolgáltatása például ingyenes és megbízható. <https://www.mailersend.com/> Egy alternatíva lehet például az Amazon SES. <https://aws.amazon.com/ses/> A rendszer csak akkor enged továbblépni, ha sikeresen tudott csatlakozni az SMTP szerverhez.

A rendszer ezután felszólít a program újraindítására. Ezután a program normál módban indul el, és használatba vehető. További konfiguráció a [programgyökér]/storage/appsettings.json fájlban lehetséges (pl.: tárolás helye, log szintek stb.).


5.4 Az alkalmazás használata felhasználóknak

5.4.1 Bejelentkezés

A webalkalmazást megnyitva azonnal a bejelentkező felületre kerülünk. Itt szükséges megadni egy létező felhasználónevet, és az ahhoz tartozó jelszót, hogy be tudjunk jelentkezni.

A sötét téma bejelentkezési felület. A cím "Bejelentkezés" nagy, fehér, sans-serif betűkben. Alatta a "Felhasználónév:" címke egy fehér keretű input mezőhöz tartozik. Ezután a "Jelszó:" címke egy hasonló input mezőhöz tartozik. Alatta egy checkbox áll a "Bejelentkezve maradok" szöveggel. Középen egy kék "Bejelentkezés" gomb található. Az alján a "Nincs fiókod? [Regisztrálj!](#)" szöveg látható, ahol a link kék és aláhúzott.

A mobilalkalmazást megnyitva szintén a bejelentkező felületen találjuk magunkat. Itt ugyanúgy meg kell adnunk a létező, érvényes adatokat, hogy be tudjunk lépni.



Felhasználónév:

Jelszó:

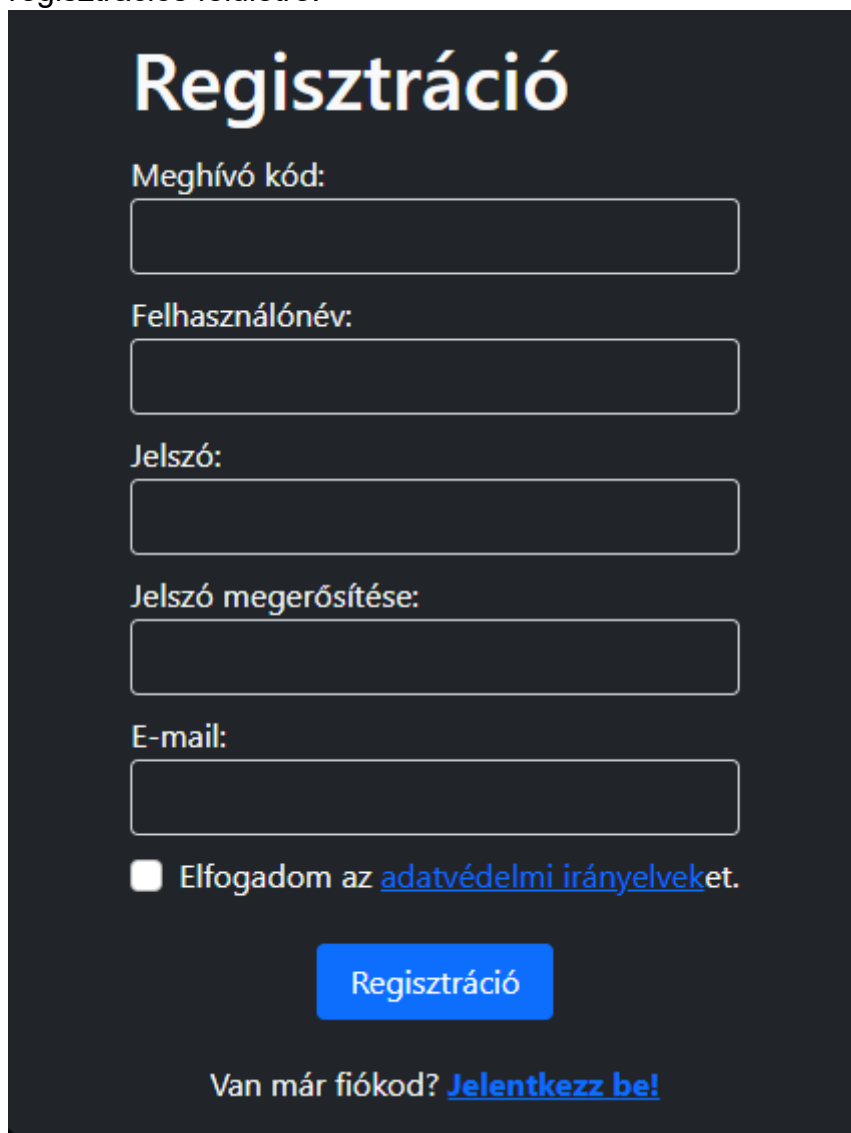
Bejelentkezés

A „Bejelentkezés” gombra kattintva a rendszer ellenőrzi, megfelelőek-e a megadott adatok, és átirányítja a felhasználót a főoldalra.

5.4.2 Regisztráció

Ha a felhasználó nem rendelkezik fiókkal, akkor regisztrálnia kell. Mobilalkalmazás használata esetén a felhasználónak már rendelkeznie kell fiókkal, Amennyiben nem rendelkezik ilyennel, azt a webes alkalmazáson teheti meg az alábbiak szerint.

Megteheti, ha rákattint a „Regisztrálj!” hiperhivatkozásra, így átkerül a regisztrációs felületre.

The image shows a registration form on a dark background. At the top, the word "Regisztráció" is written in a large, bold, white font. Below it, there are five input fields, each preceded by a label in a smaller white font: "Meghívó kód:", "Felhasználónév:", "Jelszó:", "Jelszó megerősítése:", and "E-mail:". Each label is followed by a white-outlined rectangular input box. Below the "E-mail:" field, there is a small white square checkbox followed by the text "Elfogadom az [adatvédelmi irányelveket.](#)" in white. At the bottom center, there is a blue rectangular button with the white text "Regisztráció". Below the button, there is a line of white text: "Van már fiókod? [Jelentkezz be!](#)".

A regisztrációhoz szükséges egy érvényes meghívó kód, egy szabad felhasználónév, egy jelszó, mely legalább 8 karakterből áll és egy érvényes e-mail cím. Az adatvédelmi irányelveket el lehet olvasni a kékkel aláhúzott hiperhivatkozásra való rákattintással.

Miután a regisztráló kitöltötte helyesen a mezőket és elfogadta az irányelveket, a „Regisztráció” gombra kattintva a rendszer elküld egy e-mailt a megadott e-mail címre, és várja, hogy a regisztráló megerősítse e-mail címét a levélben kapott megerősítő linkre kattintva.

Az e-mail cím megerősítése után a bejelentkező felületre kerül a felhasználó, ahol a már létező fiókjába be is léphet.

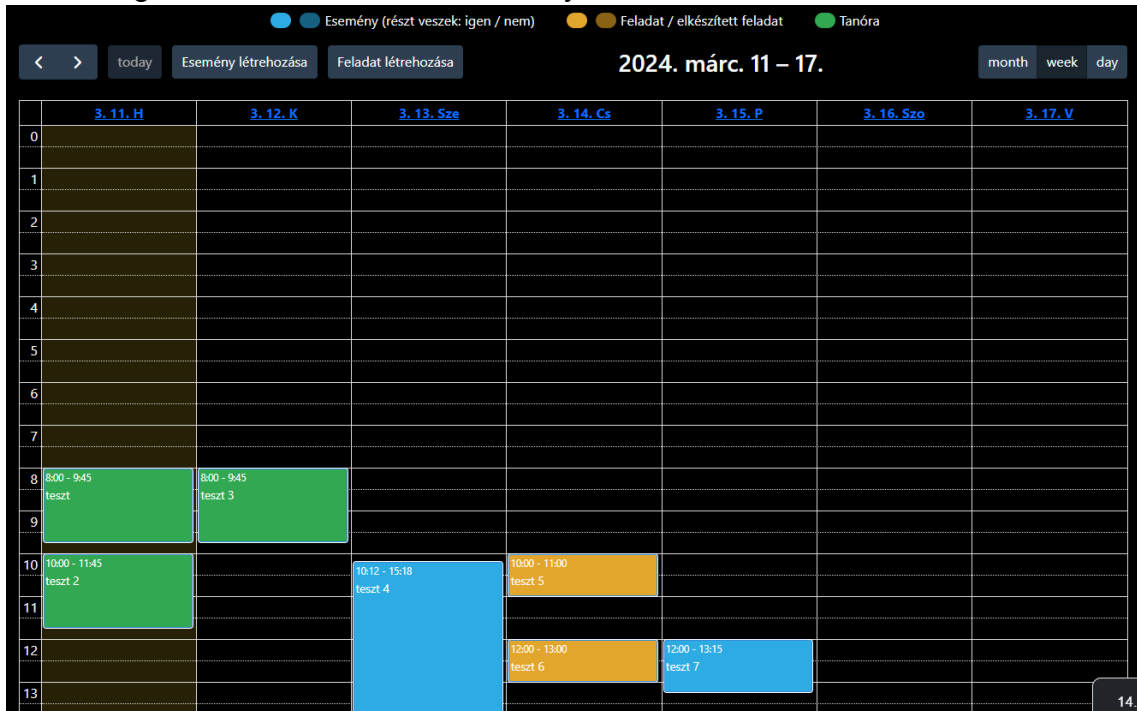
5.4.3 Webalkalmazás felület

5.4.3.1 Navigációs sáv



5.4.3.2 Főoldal

Bejelentkezés után főoldal tárul elénk, ahol a felhasználó az aktuális hétre vonatkozó információkat fogja látni: osztályon belüli csoportok eseményei, tanórái és feladatai. Lehet váltani hónapot, hét és nap nézetbe, hogy különböző szemszögből lehessen látni az eseményeket, feladatokat.



A „today” gombbal lehet az aktuális napot/hetet/hónapot megjeleníteni. „Esemény létrehozása”/”Feladat létrehozása” gomb szolgál esemény/feladat hozzáadására, mely lenyomása után átvisz egy külön oldalra, ahol az új esemény/feladat paramétereit lehet beállítani a létrehozás előtt.

5.4.3.3 Feladatok

A navigációs sáv „Feladatok” menüpontjára kattintva egy újabb felületre visz át a rendszer, ahol még a nem késznek jelölt feladatokat lehet megtalálni.

Az „Új feladat” gombbal lehet feladatot létrehozni.

Három tevékenység érhető el egy feladathoz:

1. „Csevegés”: Lenyomásakor megnyitja a feladathoz tartozó csevegést.
2. „Szerkesztés”: Átírányít a feladat paramétereit kimutató oldalra, ahol azokat szerkeszteni lehet, majd menteni, esetleg törölni.
3. „Kész”: Ezzel a gombbal lehet egy feladatot késznek jelölni, amint a gomb lenyomásra kerül a feladat eltűnik erről az oldalról, de a főoldalon még megtalálható lesz.

5.4.3.4 Csevegések

Itt találhatóak azok a csoportok, melyeknek a felhasználó a tagja. A táblázat „Műveletek” oszlopában a „Csevegés” gombra kattintva az adott csoporthoz tartozó csevegés fog megnyílni.

B Főoldal Feladatok Csevegések Gomb Admin + Test			
Csoportok			
Név	Oktató	Műveletek	
alap	John Doe	Csevegés	

5.4.3.5 Admin

Ez az oldal kizárólag olyan felhasználóknak jelenik meg, akik bármely osztályban adminisztrátor szerepet töltenek be: ők kezelik a tárgyakat, a csoportokat és az órákat. Az adminisztrátor feladata meghívni a felhasználókat az osztályba. A regisztrációkor a felhasználóhoz automatikusan hozzá lesz rendelve ekkor az osztályuk is, amiből természetesen egy lehet. Osztályadminisztrátori szerepet több felhasználó is betölthet osztályonként. Az adminisztrátor létrehozhat tárgyakat, csoportokat és órákat, melyeket az osztály többi része is lát.

B

HR

Főoldal

Feladatok

Csevegések

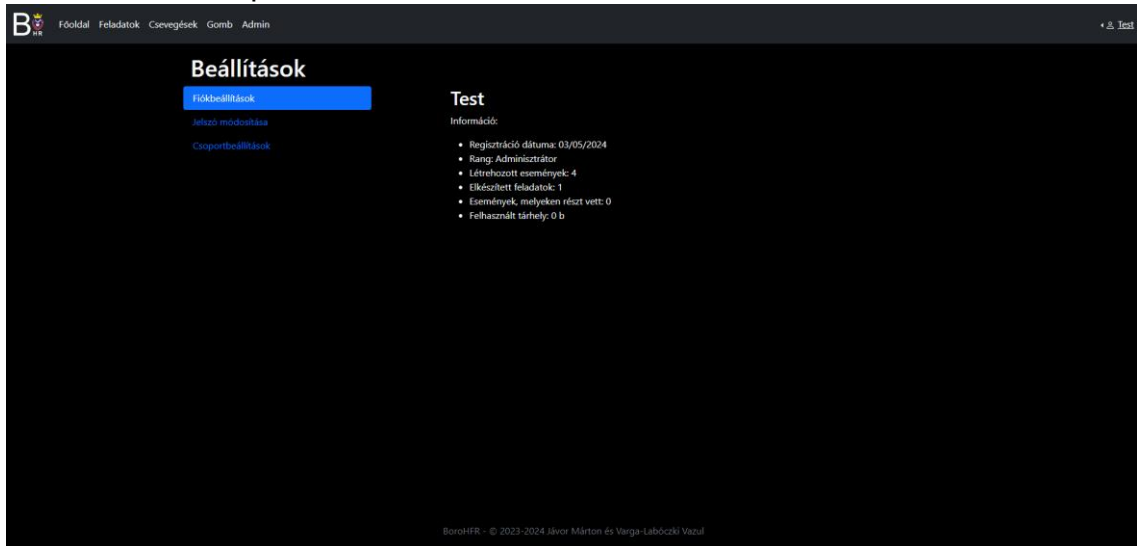
Gomb

Admin

5.4.3.6 Beállítások

A navigációs sáv jobb szélén található a felhasználónévvel szövegezett legördülő menü, melyben található a „Kijelentkezés” gomb, amellyel ki lehet jelentkezni a fiókból.

A menü másik pontja a „Beállítások”, melyre kattintva a rendszer átirányít a beállításokat végző oldalra. Itt a felhasználó a saját található statisztikáit találja a „Fiókbeállítások” pont alatt.



Ezen az oldalon végezhető a jelszó módosítás is: a felhasználó a felületen megadja a jelenlegi jelszavát, majd az újat kétszer, és a mentés gomb lenyomása után megváltozik a jelszó, ha helyesen töltötte ki a mezőket.

A „Csoportbeállítások” alatt lehet csatlakozni a csoportokba, illetve kilépni azokból. Van keresésre is lehetőség a csoportok között, ha esetleg nem találná a felhasználó a keresett csoportot.



Beállítások

[Fiókbeállítások](#)[Jelszó módosítása](#)[Csoportbeállítások](#)

Jelenlegi jelszó:

Új jelszó:

Új jelszó ismét:

Mentés

Beállítások

[Fiókbeállítások](#)[Jelszó módosítása](#)[Csoportbeállítások](#)

Csoportjaid:

Matematika 2 - alap (John Doe) [Kilépés](#)

Csoportok keresése

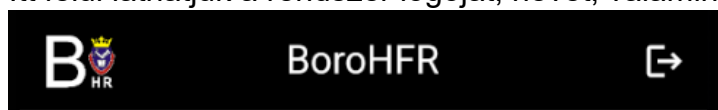
[Keresés](#)

Még nincs itt semmi. Keress rá egy tantárgyra!

5.4.4 Mobilalkalmazás felület

5.4.4.1 Főoldal

Ha bejelentkeztünk a mobilalkalmazásba, a főoldal tárul elénk. Itt felül láthatjuk a rendszer logóját, nevét, valamint a kijelentkezés gombot.



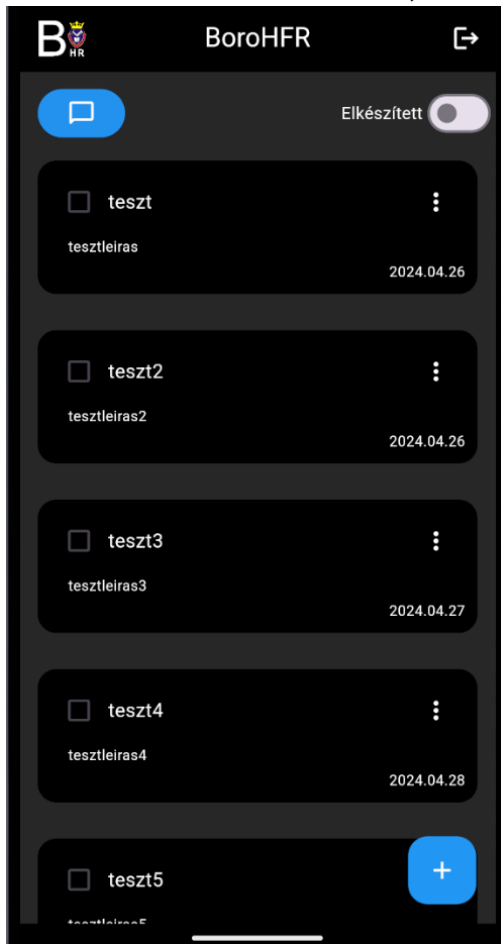
Kijelentkezéskor az alkalmazás megerősítést kér, hogy véletlenül ne jelentkezzünk ki az oldalról.

Kijelentkezés

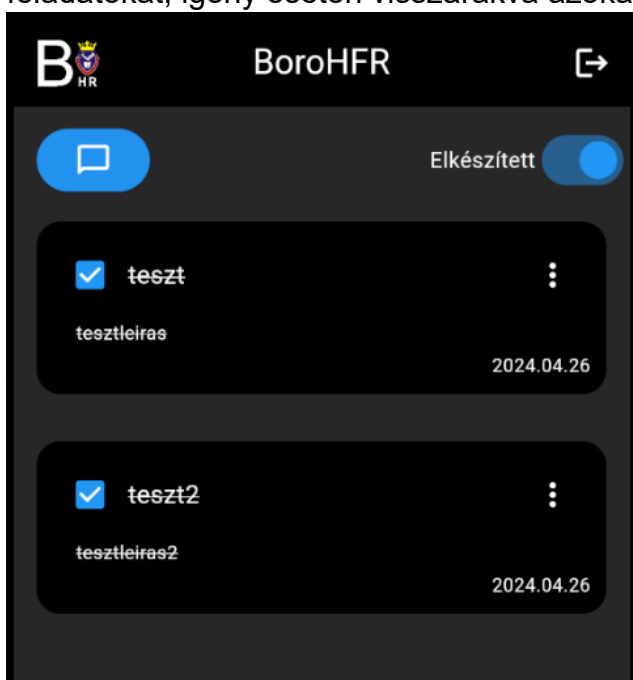
Biztosan kijelentkezel?

[Mégse](#)[Kijelentkezés](#)

A főoldal fő részét az elkészítendő feladatok alkotják. Ezen a részen láthatjuk az elkészítendő feladatokat, egy görgethető oldalon, felsorolva.

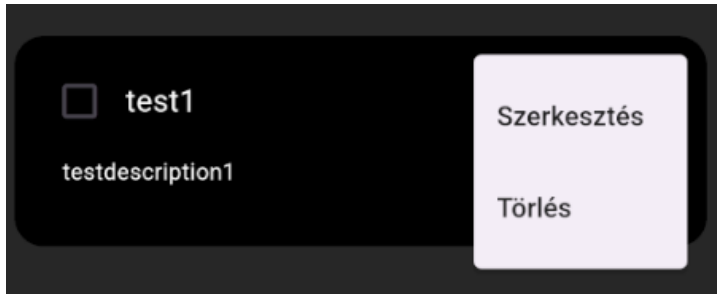


Amennyiben elkészülünk egy feladattal, a cím mellett lévő gombra kattintva a feladat átkerül az elkészítettek közé. Ha esetleg meggondoljuk magunkat, az "Elkészített" kétállású gombra kattintva megtekinthetjük az elkészített feladatokat, igény esetén visszarakva azokat a még nem kész feladatok közé.

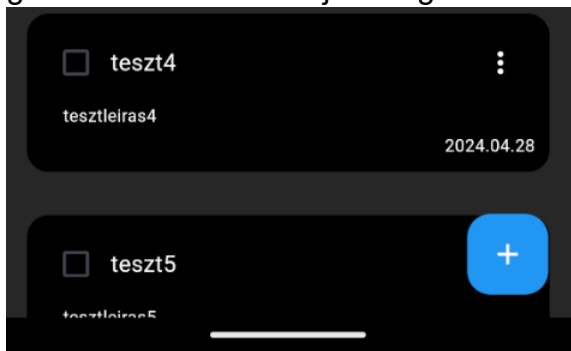


Az egyes feladatoknál a 3 pöttyre kattintva 2 menüpontból választhatunk:

- Szerkesztés: Ebben a menüpontban tudjuk a feladatot szerkeszteni, ha annak paraméterei megváltoztak volna.
- Törlés: Tudjuk törölni a már elkészített, vagy esetleg tévesen létrehozott feladatokat.



Ha új feladatot szeretnénk hozzáadni, akkor azt a jobb alul található kék '+' gombra kattintva tehetjük meg.



Ilyenkor egy olyan felület tárul elénk, melyben meg tudjuk adni a létrehozandó feladat adatait.

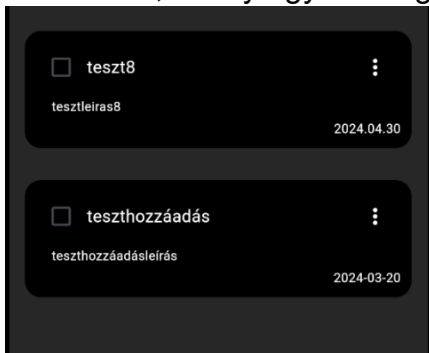
The screenshot shows the BoroHFR mobile application interface. At the top, there's a header with the BoroHFR logo and a toggle switch labeled 'Elkészített'. Below this, a modal window titled 'Feladat létrehozása' (Task Creation) is displayed. The modal contains three input fields: 'Cím:' (Title), 'Határidő:' (Deadline), and 'Leírás:' (Description). The 'Határidő:' field is pre-filled with the date '2024-03-20' and the time '15:54'. At the bottom of the modal, there are two buttons: 'Mégse' (Cancel) and 'Hozzáadás' (Add). The background shows a list of tasks, including 'teszt', 'teszt4', and 'teszt5', each with a checkbox and a three-dot menu icon.

A határidő kiválasztásához segítséget is kapunk.

This screenshot shows a date picker interface. At the top, it displays 'Wed, Mar 20' with an edit icon. Below this, there's a calendar for 'March 2024'. The calendar grid shows days from Sunday to Saturday. The date '20' is highlighted with a purple circle. At the bottom, there are 'Cancel' and 'OK' buttons.

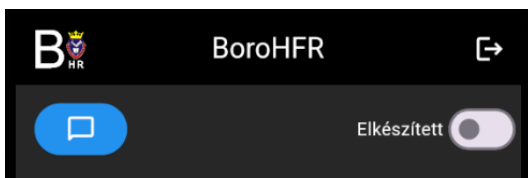
This screenshot shows a time picker interface. At the top, it says 'Enter time'. Below this, there are two large input fields for 'Hour' and 'Minute'. The 'Hour' field is set to '13' and the 'Minute' field is set to '56'. At the bottom, there are three buttons: a clock icon, 'Cancel', and 'OK'.

Ha beírtuk a megfelelő paramétereket, a hozzáadás gombbal tudjuk hozzáadni a feladatot, amely egyből meg is jelenik a képernyőn.

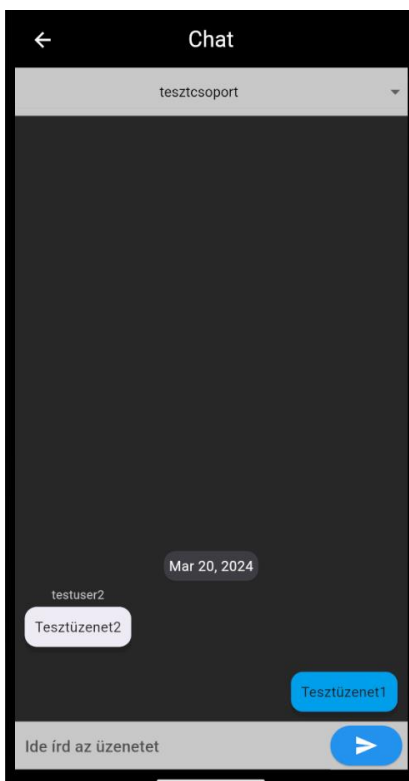


5.4.4.2 Chat

A mobilalkalmazáshoz (mint a webalkalmazáshoz) tartozik egy beszélgető rendszer. Ezt a főoldal bal felső sarkában található kék chat gombbal tudjuk elérni.



A gombot megnyomva megnyílik a chat.



A felső lenyíló menüre kattintva kiválaszthatjuk, hogy melyik csoport csevegését szeretnénk látni. A saját üzeneteink kéken, másoké fehéren jelennek meg. Az üzenetek napok szerint csoportosítva vannak. Az alsó szövegbeviteli mezőbe írva tudunk üzenetet küldeni.

6. Irodalomjegyzék

6.1 Könyvek

- Reiter István: C# programozás lépésről lépésre (Jedlik Oktatási Stúdió, Budapest, 2012, ISBN: 9786155012174)
- Eric Vogel: Beginning Entity Framework Core 5: From Novice to Professional (Apress, 2021, ISBN: 9781484268810)

6.2 Online cikkek

- <https://hu.wikipedia.org/wiki/Szoftvertesztel%C3%A9s>
- <https://testerlab.io/blog/4-ok-hogy-mi%C3%A9rt-fontos-a-szoftvertesztel%C3%A9s/>
- <https://www.learnentityframeworkcore.com/>
- <https://learn.microsoft.com/en-us/ef/core/>
- <https://learn.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-8.0>
- <https://bencull.com/blog/is-active-route-tag-helper-asp-net-mvc-core>

6.3 Felhasznált anyagok, eszközök

- <https://github.com/devlooped/moq>
- <https://github.com/xunit/xunit>
- <https://github.com/BcryptNet/bcrypt.net>
- <https://github.com/Humanizr/Humanizer>
- <https://github.com/jstedfast/MailKit>
- <https://github.com/mysql-net/MySqlConnector>
- <https://github.com/PomeloFoundation/Pomelo.EntityFrameworkCore.MySql>
- <https://github.com/soundaranbu/Razor.Templating.Core>
- <https://nginx.org/en/docs/>
- <https://certbot.eff.org/>
- <https://github.com/dotnet/aspnetcore>
- <https://fullcalendar.io/>
- <https://pub.dev/packages/http>
- https://pub.dev/packages/grouped_list
- <https://pub.dev/packages/intl>
- https://pub.dev/packages/signaler_netcore

7. Mellékletek

7.1 Online elérhetőség

<https://borohfr.jmrtn.dev>

7.2 Forráskód

<https://github.com/marton200472/BoroHFR-Server>

https://github.com/cromwell1028/bhr_app