Szakdolgozat

Miskolci Egyetem

Közösségi Videómegosztó Szoftver Tervezése és Implementálása

Készítette:

Nagy Máté

Programtervező Informatikus

Témavezető:

Fazekas Levente Áron

Miskolc, 2024

Miskolci Egyetem
Gépészmérnöki és Informatikai Kar
Alkalmazott Informatikai Intézeti Tanszék
Intézményazonosító: FI 87515
Azonosító szám: GEIAK-INZV77/2021-22-1f/BSc

Szakdolgozat Feladat

Nagy Máté (U3ROFS) BSc programtervező informatikus jelölt részére.

A szakdolgozat tárgyköre: Webtechnológiák

A szakdolgozat címe: Közösségi Videómegosztó Szoftver Tervezése és Implementálása

A feladat részletezése:

1. A szakdolgozat célja egy valós idejű videómegosztásra alkalmas webes applikáció tervezése és implementálása.
2. Mutassa be a WebSocket és egyéb webes, valós idejű kommunikációra alkalmas API-kat!
3. Tervezzen meg és implementáljon egy szoftvert, amely képes az alábbi funkciók elvégzésére:
   (a) Felhasználós nyilvántartása
   (b) Videómegosztásra alkalmas szobák létrehozása, nyilvántartása
   (c) Egyidejű vetítés azonos szobában tartózkodó felhasználók számára
   (d) Felhasználók közötti chat
4. Mutassa be az esetleges továbbfejlesztési lehetőségeket!

Tervezésvezető: Fazekas Levente Áron (tanszéki mérnök)

A feladat kiadásának ideje: 2023.09.01

A feladat beadásának határideje: 2024.05.12 Ezt majd pontosan ki kell tölteni nyomtatás előtt

. . . . . .
tanszékvezető

1.

A szakdolgozat feladat módosítása szükséges (módosítás külön lapon)
nem szükséges

Miskolc, ...
tervezésvezető

2. A feladat kidolgozását ellenőriztem:

tervezésvezető

(1) .....
(2) .....
(3) .....
(4) .....

dátum, tervezésvezető aláírása

3. A szakdolgozat beadható
nem adható be

Miskolc, ...
konzulens tervezésvezető

4. A szakdolgozat ..... szövegoldalt
..... db ábrát
..... db CD mellékletet
..... egyéb mellékletet tartalmaz.

5.

A szakdolgozat bírálatra bocsátható
nem bocsátható

A bíráló neve, címe: .....

Miskolc, ...
tanszékvezető

6. A szakdolgozat osztályzata

a témavezető javaslata: .....
a bíráló javaslata: .....
a Záróvizsga Bizottság döntése: .....

Miskolc, ...
a Záróvizsga Bizottság Elnöke

Eredetiségi Nyilatkozat

Alulírott **Nagy Máté**; Neptun-kód: `U3ROFS` a Miskolci Egyetem Gépészmérnöki és Informatikai Karának végzős Programtervező Informatikus szakos hallgatója ezennel büntetőjogi és fegyelmi felelősségem tudatában nyilatkozom és aláírásommal igazolom, hogy *Közösségi Videómegosztó Szoftver Tervezése és Implementálása* című szakdolgozatom saját, önálló munkám; az abban hivatkozott szakirodalom felhasználása a forráskezelés szabályai szerint történt.

Tudomásul veszem, hogy szakdolgozat esetén plágiumnak számít:

- szószerinti idézet közlése idézőjel és hivatkozás megjelölése nélkül;
- tartalmi idézet hivatkozás megjelölése nélkül;
- más publikált gondolatainak saját gondolatként való feltüntetése.

Alulírott kijelentem, hogy a plágium fogalmát megismertem, és tudomásul veszem, hogy plágium esetén szakdolgozatom visszautasításra kerül.

Miskolc, ...év ..... .hó ..... .nap

.....

Hallgató

Tartalomjegyzék

1. fejezet

Bevezetés

A digitális korban élünk, ahol a technológia gyorsan fejlődik és az embereknek egyre több lehetőségük van az online interakcióra. Azonban a közösségi médiával és az egyéb digitális platformokkal eltöltött idő gyakran személytelen és egyoldalú.

A "WatchWithFriends" névre keresztelt alkalmazásom arra hivatott, hogy ezen változtasson. Az alkalmazás lehetőséget nyújt arra, hogy a felhasználók együtt nézzenek YouTube-videókat, miközben élő chaten kommunikálnak egymással. Az alkalmazásban létrehozott "szobák" révén a felhasználók egyszerűen csatlakozhatnak, és megoszthatják egymással kedvenc videóikat egy közös lejátszási listán keresztül. Az alkalmazás nem csupán egy online videómegosztó platformot egészít ki, hanem egy új közösségi élményt is kínál. Ezzel az eszközzel a barátok és családok, akár távoli földrészeken is, összehozhatók egy közös élmény kapcsán. A "WatchWithFriends" az együtt töltött idő minőségét kívánja javítani, lehetővé téve, hogy az emberek valós időben osszák meg reakcióikat, érzéseiket egy-egy videóval kapcsolatban.

Továbbá a chat funkcióval a felhasználók azonnal megvitathatják a látottakat, megoszthatják tippeiket vagy akár következő videó választási ötleteiket is.

A szoftver nem csak szórakoztató, de oktatási célokra is felhasználható. Tanárok és diákok könnyedén nézhetnek együtt oktatási anyagokat, és azonnal megbeszélhetik azok tartalmát. Ezen kívül a vállalati prezentációknál is hasznos lehet a közös videómegtekintés funkció.

A "WatchWithFriends" egy olyan platform, ami a társasági interakciókat nem csak kiterjeszti, de egy új dimenzióba is emeli. Az alkalmazás tehát nem csak egy technológiai újítás, hanem egy közösségi eszköz is, amely összeköti az embereket függetlenül fizikai távolságuktól.

2. fejezet

Tervezés

Ebben a fejezetben a "WatchWithFriends" alkalmazás tervezési folyamatát mutatom be. A fejezet első részében a felhasználói felület tervezését ismertetem, majd a szerver és a kliens közötti kommunikáció megvalósítását.

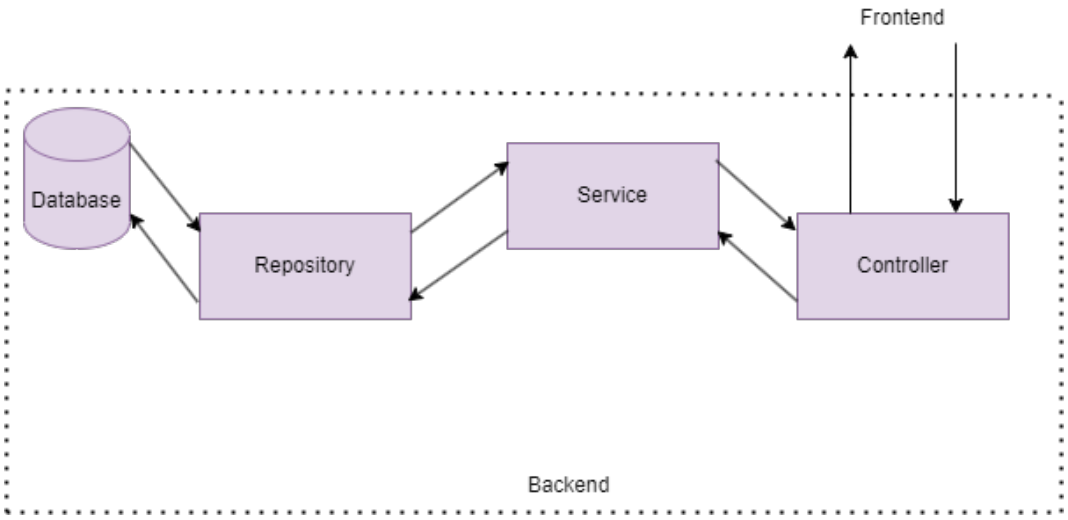
Felhasználói felület tervezése

FELHASZNÁLÓI FELÜLET!!!

Szerver oldali logika megtervezése

Architektúra

Az alkalmazásnál MVC (Model-View-Controller) architektúrára építettem, illetve bővítettem a Repository és a Service rétegekkel. Az MVC architektúra egy architektúrális minta, amely a felhasználói felületet (View), az alkalmazás logikáját (Controller) és az adatokat (Model) három különálló részre osztja. A Repository réteg a Model réteghez tartozik, a Service réteg pedig a Controller réteghez. Azért bontottam tovább a rétegeket, mert így jobban elkülönülnek a felelősségi körök, és a kód is átláthatóbb lesz, illetve könnyebben bővíthető, és tesztelhető lesz.



2.1. ábra. Backend Architektúra

Ismertetem a rétegek főbb tulajdonságait:

- **Adatbázis**: Az alkalmazásom adattára. Itt tárolódnak a statikus és dinamikus adatok.
- **Repositoryk**: Az adatbázis és az alkalmazás logikája közötti köztes réteg. CRUD műveletek, komplex lekérdezések.
- **Servicek**: Az üzleti logika helye. Itt történnek a komplex számítások, validációk és egyéb logikai műveletek.
- **Controllerek**: Az API végpontokat kezelő réteg. Fogadják a kliens kéréseit, és válaszokat generálnak.

Szoftvertervezési elvek (tervezés)

A szoftvertervezési elvek alapvető útmutatások a kiváló kód fejlesztéséhez. Ezek az iránymutatások növelik a kód minőségét, gyorsítják a fejlesztési folyamatot és minimalizálják a hibalehetőségeket. Az ilyen elvek segítenek abban, hogy a kód könnyen újrafelhasználható és karbantartható legyen.

A kód olvashatóságát és tesztelhetőségét is javítják, ami hosszú távon időt és erőforrást spórol. A rugalmasság és a bővíthetőség is nő, tehát a kód könnyen adaptálható új funkciók vagy változások esetén. Továbbá, ezek az elvek arra is ösztönöznek, hogy egyszerű és nem redundáns kódot írj, ami csökkenti a komplexitást és növeli a kohéziót. Az elvek célja a kód moduláris felépítése és az alacsonyabb összekapcsoltság, ami hozzájárul a jobb szervezettséghez és könnyebb karbantartáshoz.

Összességében, a szoftvertervezési elvek olyan iránymutatások, amelyek célja a hatékony, karbantartható és minőségi kód létrehozása. Én a SOLID elveket használtam a szoftvertervezési elvek közül, mert ezeket a leggyakrabban használják, és a legtöbb fejlesztő ismeri őket.

Ismertetem a SOLID elvek főbb tulajdonságait:

- **Single Responsibility Principle (SRP)**: Egy osztály vagy modul csak egy felelősséggel rendelkezzen.
- **Open-Closed Principle (OCP)**: Nyitva áll, de nem módosítható. A kód legyen könnyen bővíthető, de ne kelljen módosítani a meglévő kódot.
- **Liskov Substitution Principle (LSP)**: A leszármazott osztályok legyenek helyettesíthetők az ősosztályokkal.
- **Interface Segregation Principle (ISP)**: A kliensek ne függjenek olyan interfészektől, amelyeket nem használnak.
- **Dependency Inversion Principle (DIP)**: A függőségek irányát fordítsuk meg, és a magasabb szintű modulok ne függjenek a részletektől.

3. fejezet

Technólógia

3.1 Felhasználói felület

Az alkalmazás felhasználói felületét a React könyvtár segítségével valósítottam meg. A React egy nyílt forráskódú JavaScript könyvtár, amelyet a Facebook fejlesztett ki. A könyvtár célja, hogy a felhasználói felület fejlesztését egyszerűbbé tegye. A React egy komponens alapú könyvtár, amely lehetővé teszi a fejlesztők számára, hogy újra felhasználható komponenseket hozzanak létre, amelyeket összeállítva komplex felhasználói felületeket hozhatnak létre.

Komponensek A React komponensek egyfajta sablonok, amelyek a felhasználói felület egy részét írják le. A komponensek egy egyszerű JavaScript objektumok, amelyeknek van egy render metódusa, amely visszaadja a komponens felhasználói felületét.

Komponensek főbb tulajdonságai:

- Paraméterek (props)
- Gyerekei (children)
- Állapotai (state)
- Életciklus metódusai (lifecycle)
- Eseménykezelői (event)
- Stílusai (style)
- Referenciák (ref)
- Kontextusai (context)
- Típusai (type)
- Kulcsai (key)

React

Sokat gondolkodtam, hogy React-ot vagy Angulart használjak az alkalmazás fejlesztéséhez. A React mellett döntöttem, mert a React egy könnyű, gyors és hatékony könyvtár. Viszont sokat Angularoztam is, és a kettő között nem sok különbséget találtam.

Külömbség a React és az Angular között:

- **Egyszerűség és Tanulási Görbe**
  - React: Relatíve egyszerűbb és könnyebben tanulható.
  - Angular: Több beépített funkció, hosszabb tanulási idő.
- **Flexibilitás és Testreszabhatóság**
  - React: Nagyon rugalmas, lehetőség más könyvtárak és eszközök beillesztésére.
  - Angular: Teljes megoldás, kevesebb szabadság de nincs szükség külső könyvtárakra.
- **Teljesítmény**
  - React: Virtuális DOM, gyakran gyorsabb nagy listák esetén.
  - Angular: Real-time Two-Way Data Binding, néha lassabb.
- **Közösség és Ökoszisztéma**
  - React: Nagyobb közösség, több harmadik féltől származó könyvtár.
  - Angular: Kevesebb, de minőségibb beépített eszköz.
- **Fejlesztői tapasztalat**
  - React: JSX és modern JavaScript funkciók.
  - Angular: TypeScript alapú, előny vagy hátrány a preferenciától függően.

Redux vagy Context API

Felmerült bennem a kérdés, hogy a Redux vagy a Context API-t használjam az alkalmazás állapotának kezelésére. A Redux egy állapotkezelő könyvtár, amely lehetővé teszi az alkalmazás állapotának tárolását egy központi helyen. A Context API egy React API, amely lehetővé teszi az alkalmazás állapotának tárolását és megosztását a komponensek között. Nem láttam értelmét, hogy egy ilyen kis alkalmazásnál Redux-ot használjak, ezért a Context API-t választottam.

Ismertetem a Context API főbb tulajdonságait:

- **Beépített Megoldás:** A Context API része a React alapkönyvtárnak, nincs szükség külső függőségekre.
- **Egyszerűség:** A Context API egyszerűbb és könnyen megérthető, kevesebb boilerplate kóddal.
- **Nincs Middleware:** Nem támogat middleware-eket alapból, így aszinkron állapotfrissítéseket manuálisan kell kezelni.

- **Nem Optimalizált Nagy Alkalmazásokhoz:** Nagy alkalmazásokban hatékonytalan lehet, mert minden Context változásra újrarendereli az összes fogyasztót, hacsak nem optimalizáljuk manuálisan.
- **Kevesebb Eszköz:** Kevesebb beépített eszköze van, mint a Redux-nak, de ez nem mindig hátrány, függ a projekt igényeitől.

Példa a Context API állapotkezelésre:

```
// context
const CounterContext = React.createContext();

// state and update
const [count, setCount] = useState(0);

// state refresh
setCount(count + 1);
```

Ismertetem a Redux főbb tulajdonságait:

- **Optimalizáció:** Redux kifejezetten optimalizált a nagyobb alkalmazások számára, és lehetővé teszi az állapot frissítéseinek finom szabályozását.
- **Middleware Támogatás:** Redux lehetőséget ad middleware-ek használatára, amelyek lehetővé teszik az aszinkron állapotfrissítések könnyebb kezelését.
- **Tesztelhetőség:** A Redux architektúrája miatt a tesztelés egyszerűbb, minden action egy független egység, és a reducer funkciók tiszta függvények.
- **Eszközök és Közösség:** Redux-nak nagy közössége van, és sok kiegészítő eszköz, például a Redux DevTools.
- **Boilerplate Kód:** Több boilerplate kód szükséges, hogy elindítsunk egy Redux-alapú állapotkezelést.

Példa a Redux állapotkezelésre:

```
// action
const incrementAction = { type: 'INCREMENT' };

// reducer
function counterReducer(state = 0, action) {
  if (action.type === 'INCREMENT') {
    return state + 1;
  }
  return state;
}

// dispatch
dispatch(incrementAction);
```

TypeScript vagy JavaScript

A TypeScript egy nyílt forráskódú, szigorúan típusos programozási nyelv, amely a JavaScriptre épül, lehetővé teszi a statikus típusok használatát, és minden érvényes JavaScript kód érvényes TypeScript kódnak is számít. Inkább a TypeScript mellett döntöttem, mert a statikus típusok használata növeli a kód minőségét és a fejlesztési sebességet, illetve csökkenti a hiba lehetőségek számát.

3.2 Szerveroldali Logika és API-k

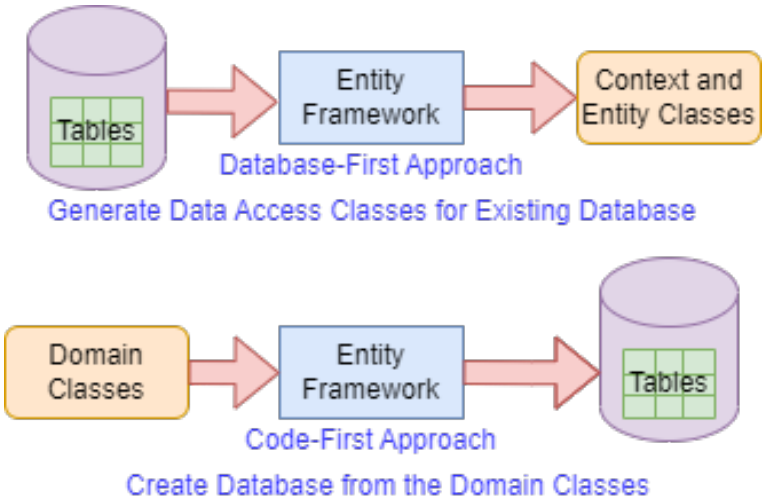
A szerver és a kliens közötti kommunikáció megvalósításához az ASP.NET Core-t használtam. Az ASP.NET Core egy nyílt forráskódú, cross-platform, magas teljesítményű keretrendszer, amelyet a Microsoft fejlesztett ki. Az ASP.NET Core-t a webalkalmazások és a webes API-k fejlesztésére használják. A környezetet a C# programozási nyelvhez tervezték, de támogatja a többi .NET nyelvet is. Én a megvalósítás során a C# nyelvet használtam. Ebben a fejezetben bemutatom a szerveroldali logikát és az API-kat.

Adatbázis

Az alkalmazásnál MS SQL adatbázist használtam, mert a Microsoft SQL Server-t használom a személyes projekteimhez. Az MS SQL (Microsoft SQL Server) egy relációs adatbázis-kezelő rendszer (RDBMS) a Microsofttól. Jól skálázható, és számos fejlett funkcióval rendelkezik, mint például tárolt eljárások, triggerek és nézetek . Gyakran használják vállalati szintű alkalmazásokban, és támogatja a SQL nyelvet az adatok lekérdezésére és manipulálására. Támogatja az ACID tulajdonságokat és a tranzakciós integritást. Az ER modell-t használtam az adatbázis tervezéséhez. Ezt majd a Backend fejlesztési folyamatánál részletezem.

Entity Framework Core

Az Entity Framework Core (EF Core) egy objektum-relációs leképzési (ORM) keretrendszer a Microsofttól, amely .NET Core és .NET 5+ alkalmazások számára készült. Lehetővé teszi a fejlesztők számára, hogy magas szintű, objektumorientált API-n keresztül dolgozzanak adatbázisokkal, anélkül hogy közvetlen SQL lekérdezéseket kellene írniuk. Támogatja a code-first és a database-first megközelítéseket, így rugalmasságot biztosít az adatmodell és az adatbázisséma kialakításában. Az EF Core lehetővé teszi a LINQ (Language Integrated Query) használatát, ami természetes módon illeszkedik a C# és más .NET nyelvekhez.

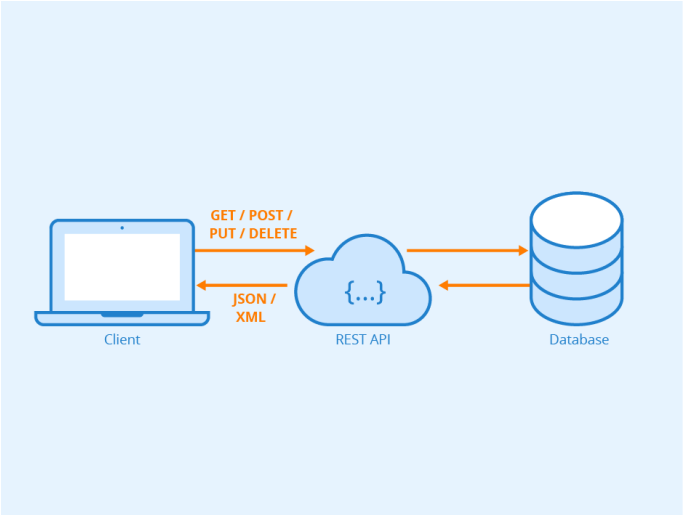


3.1. ábra. Entity Framework Core

Skálázható és teljesítmény-optimalizált, ezért alkalmas mind kis, mind nagyobb vállalati projektekben. Az EF Core támogatja a többszörös adatbázis-motorokat, beleértve az SQL Server, PostgreSQL, SQLite és másokat is. Az adatmigrációk egyszerű kezelése és automatikus generálása az egyik előnye, ami könnyebbé teszi az adatbázisséma változásainak kezelését. A keretrendszer beépített támogatással rendelkezik a tranzakciókezelésre, így az ACID tulajdonságok biztosítottak. Az EF Core rugalmas konfigurációs lehetőségekkel rendelkezik, és jól integrálódik más .NET Core szolgáltatásokkal, mint például a Dependency Injection. Folyamatosan fejlődik és aktívan karbantartott, így a legújabb .NET technológiákhoz és az adatbázis-technológiákhoz is gyorsan alkalmazkodik.

API

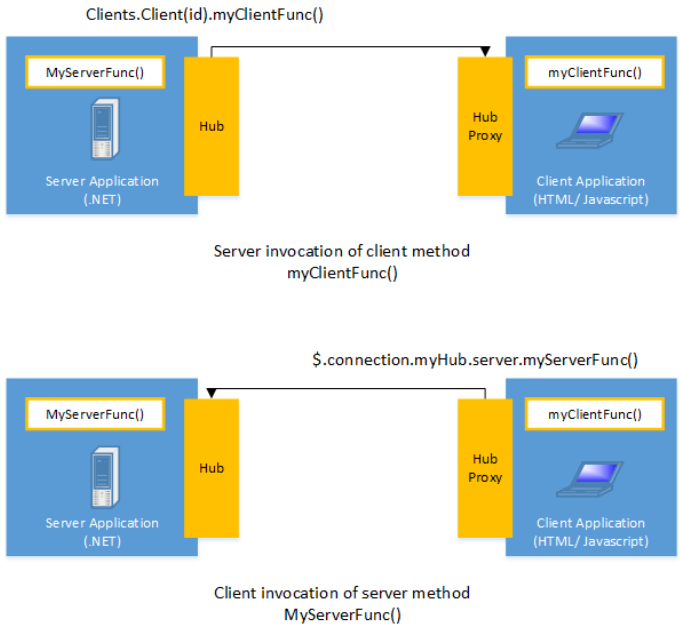
Az alkalmazásnál RESTful API-t használtam, mert a RESTful API-kat a leggyakrabban használják, és a legtöbb fejlesztő ismeri őket. A REST (Representational State Transfer) egy architektúrális stílus, amelyet a webes alkalmazásokhoz használnak. A RESTful API-kat a REST architektúra alapelvei alapján tervezik. A RESTful API-kat a HTTP protokollra építik, és a HTTP metódusokat használják a kérések kezelésére. Ennek megvalósításához a *ControllerBase* osztályt használtam a Controllerekben. Ami teljesen implementálja a RESTful API-kat, és a HTTP metódusokat.



3.2. ábra. RESTful API

A videók lejátszása során különös figyelmet fordítottam az állapotkezelésre. Ennek érdekében a WebSocket protokollt alkalmaztam. A WebSocket egy kétirányú kapcsolatot tesz lehetővé a böngésző és a szerver között. Bár a HTTP protokollra épül, mégis képes valós idejű kommunikációra. A kommunikáció HTTP portokon keresztül zajlik, ami rugalmasságot ad az alkalmazásnak. SignalR-t használtam a WebSocket protokoll megvalósításához.

A SignalR egy Microsoft által fejlesztett aszinkron könyvtár, amely valós idejű webes alkalmazások építését támogatja. Lehetővé teszi a szerver és a kliensek közötti kétirányú kommunikációt, gyakran WebSockets használatával.



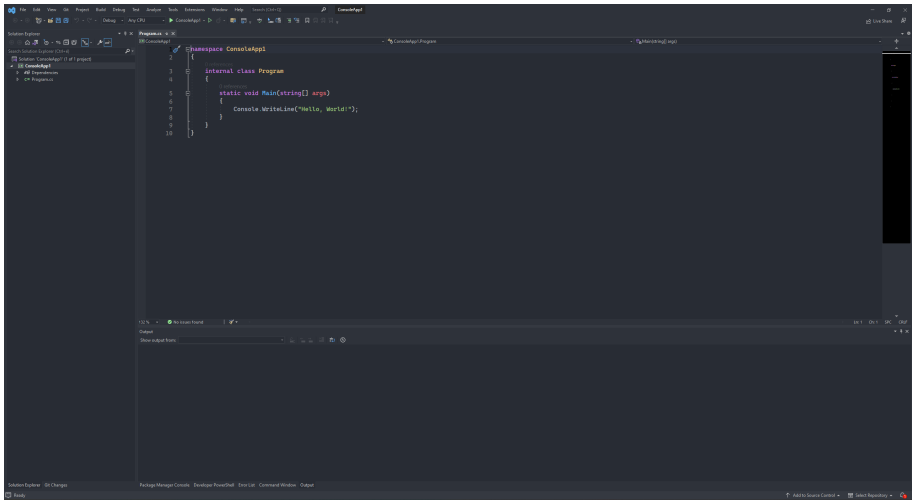
3.3. ábra. SignalR
[1]

[0]Forrás: `https://www.interserver.net/`
[1]Forrás: `https://learn.microsoft.com/`

Ez eltér a hagyományos HTTP "kérés-válasz" modelltől, mivel a szerver aktívan küldhet üzeneteket a klienseknek. Gyakori alkalmazási területei közé tartoznak a chat szolgáltatások, online játékok és élő adatfrissítések. A SignalR automatikusan választja ki a legmegfelelőbb kommunikációs mechanizmust a szerver és a kliens között, például long polling, ha WebSockets nem érhető el. A könyvtár könnyen integrálható más .NET technológiákkal és keretrendszerekkel, például ASP.NET Core-al. A SignalR segítségével könnyen implementálhatók komplex forgatókönyvek, például csoportos üzenetküldés vagy kapcsolatok kezelése. Az állapotkezelés és a kiváló skálázhatóság további előnyei a SignalR használatának. Mivel a SignalR API az ASP.NET Core része, az infrastruktúrát és a biztonsági mechanizmusokat is könnyen ki lehet terjeszteni rá. Összességében a SignalR egy erőteljes eszköz a valós idejű webes alkalmazások fejlesztéséhez.

Fejlesztői környezet

Én a Visual Studio 2022-t használtam a fejlesztéshez, mert ez az egyik legnépszerűbb IDE a .NET fejlesztők körében. A Visual Studio 2022 egy integrált fejlesztői környezet (Integrated Development Environment, IDE), amelyet a Microsoft fejlesztett ki. Ez a platform különösen népszerű a Windows alapú alkalmazások fejlesztésénél, de támogatja számos programozási nyelvet és technológiát, így a fejlesztők széles spektruma számára kínál megoldásokat. Az IDE magába foglalja a kódszerkesztést, a debuggolást, a verziókezelést és más, a fejlesztési ciklusban fontos eszközöket is.



3.4. ábra. Visual Studio 2022

Míg a Visual Studio elsősorban a Windows operációs rendszerre lett tervezve, a Microsoft kiadott egy könnyebb, keresztplatformos változatot is, a Visual Studio Code-ot. Ez utóbbi támogatja a Linux és macOS operációs rendszereket is, így a fejlesztők választhatnak az operációs rendszerüknek legmegfelelőbb változat közül.

Mindkét platform lehetővé teszi az együttműködést és a csapatmunkát, és számos bővítményt és sablont kínál a gyors és hatékony fejlesztés érdekében.

Ábrák jegyzéke

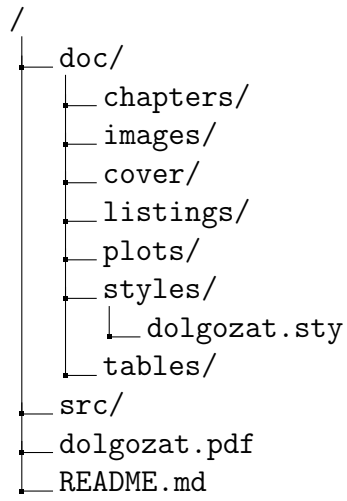
Táblázatok jegyzéke

Programkódok jegyzéke

Irodalomjegyzék

CD Használati útmutató

A CD melléklet struktúrája az alábbi:



A `doc/` jegyzékben találhatóak azok a `.tex` fájlok, melyek a szakdolgozat elkészítéséhez szükségesek. Ebben a `chapters/` jegyzék tartalmazza az egyes fejezetek forrásait. A `cover/` jegyzékben vannak a borító forrásai. Az `images/` jegyzékben kaptak helyet a dolgozatban használt képek. A `listings/` jegyzékben vannak azok a forráskódok, melyek a dolgozatban is megjelennek. A `plots/` jegyzékben azon ábrák forrásai vannak, melyek LaTeX segítségével vannak kirajzolva. A `styles/` jegyzékben vannak azok az `.sty` fájlok, melyek a dolgozat kinézetét határozzák meg. A `tables/` jegyzékben vannak a táblázatok forrásai.

A `s/` jegyzékben található az alkalmazás forráskódja.

A `dolgozat.pdf` fájl az, ami a szakdolgozatot tartalmazza pdf formátumban. Ez a fájl a LaTeX állományok fordításának végeredménye.

A `README.md` fájl egy útmutató a CD melléklet használatához.