|  |  |  |
| --- | --- | --- |
| oe_cimer_szines_print_res | **NEUMANN JÁNOS**  **INFORMATIKAI KAR** | NIK_cimer.jpg |

**SZAKDOLGOZAT**

|  |  |  |
| --- | --- | --- |
| **OE-NIK**  **2020** | Hallgató neve:  Hallgató törzskönyvi száma: | **Bárány Viktória**  **T/005038/FI12904/N** |

**Óbudai Egyetem**

**Neumann János Informatikai Kar**

**Szoftvertervezés és -fejlesztés Intézet**

**SZAKDOLGOZAT**

**FELADATLAP**

Hallgató neve: **Bárány Viktória**

Törzskönyvi száma: T/005038/FI12904/N

Neptun kódja: JEMMTI

A dolgozat címe:

**Gráfadatbázison alapuló tanulócsoport ajánló alkalmazás**

**Graph based study group recommendation application**

Intézményi konzulens: Simon-Nagy Gabriella

Külső konzulens:

Beadási határidő: 2019. december 15.

A záróvizsga tárgyai: Számítógép architektúrák

Szoftverrendszerek fejlesztése

specializáció

**A feladat**

Tervezzen és valósítson meg egy olyan alkalmazást, amely a hallgatók számára nyújt lehetőséget arra, hogy a tanuló társaikat ideálisan választhassák ki, illetve a csapatmunkákhoz kialakítandó csapatokat optimálisan szervezhessék meg. Ismertesse a gráfadatbázisok főbb tulajdonságait. Tekintse át a Neptunból kinyerhető adatok szerkezetét és kapcsolatait, a gráfadatbázisba való importálás lehetőségeit. Mérje fel a potenciális felhasználók igényeit. Tervezzen és valósítson meg egy olyan, egyszerűen használható és áttekinthető grafikus felülettel rendelkező mintaalkalmazást, amely lehetővé teszi tanulótársak, csapattársak vagy korrepetitorok keresését a felhasználó által összeválogatott szűrőfeltételek alapján, valamint a felhasználók közötti kapcsolatok elemzésével.

**A dolgozatnak tartalmaznia kell**:

* a feladat pontos ismertetését,
* a NoSQL és speciálisan a gráfadatbázisok ismertetését,
* a feladatban meghatározott célú hasonló rendszerek bemutatását,
* a feladat megoldásához alkalmazható technológiák tömör bemutatását,
* a választott megoldások indoklását,
* a tervezés és megvalósítás részletes leírását,
* a tesztelés tervét és eredményeit,
* a továbbfejlesztési lehetőségek számba vételét.

Ph.

..........……………….

Dr. Vámossy Zoltán

intézetigazgató

A szakdolgozat elévülésének határideje: **2021. december 15.**

(OE TVSz 55.§ szerint)

A dolgozatot beadásra alkalmasnak tartom:

|  |  |
| --- | --- |
| ……………….. | ..……………………. |
| külső konzulens | intézményi konzulens |

**Tartalomjegyzék**

[1 Bevezetés 6](#_Toc26462000)

[1.1 A probléma fontossága, felvezetése 6](#_Toc26462001)

[1.2 Az alkalmazásról 6](#_Toc26462002)

[1.3 Tervezett funkciók 7](#_Toc26462003)

[1.4 Alkalmazandó technológiák 8](#_Toc26462004)

[1.5 Célok meghatározása 8](#_Toc26462005)

[2 Irodalomkutatás 9](#_Toc26462006)

[2.1 Adatbázisok 9](#_Toc26462007)

[2.2 Szerveroldali technológiák 17](#_Toc26462008)

[2.3 Felhasználói felület 20](#_Toc26462009)

[2.4 Hasonló rendszerek bemutatása 23](#_Toc26462010)

[2.5 A megoldási módszer kiválasztása, a választás indoklása 25](#_Toc26462011)

[3 A részletes specifikáció leírása 26](#_Toc26462012)

[4 Tervezés 29](#_Toc26462013)

[4.1 Rendszerterv 29](#_Toc26462014)

[4.2 Adatforrások – Neptun Export módjai 30](#_Toc26462015)

[4.3 Az adatmodell tervezése 32](#_Toc26462016)

[4.4 Szerveroldal tervezése 33](#_Toc26462017)

[4.5 Kliens tervezése 33](#_Toc26462018)

[4.6 Az alkalmazás folyamatai 34](#_Toc26462019)

[4.7 Fejlesztési terv 34](#_Toc26462020)

[5 A megvalósítás leírása 35](#_Toc26462021)

[6 Tesztelés 37](#_Toc26462022)

[7 Az eredmények bemutatása, értékelése, hasonló rendszerek eredményeivel összevetése 38](#_Toc26462023)

[8 A megvalósítás elemzése, alkalmazásának és továbbfejlesztési lehetőségeinek számbavétele 39](#_Toc26462024)

[9 A szakdolgozat tartalmi összefoglalója magyarul és angol nyelven 40](#_Toc26462025)

[10 Irodalomjegyzék 41](#_Toc26462026)

# Bevezetés

Az egyetemen a bolognai rendszer bevezetésének következtében a tankörök régi formában való működése gyakorlatilag megszűnt. Régen a tankör tagjai együtt haladtak a tanulmányaik végzése során, hasonlóan a középfokú oktatásban lévő osztályközösségekhez. Ma a hallgatók ideális esetben az első féléves tárgyaikat teljesítik ilyen körülmények közt, egy fix tankörrel. A további félévek során önállóan állíthatja össze mindenki az órarendjét, így a tankörök sok esetben már nem maradnak együtt. Sajnos nagyarányú a lemorzsolódás, az egyetemet elhagyó, passzív félévre kényszerült hallgatók illetve a törzs-tárgyból megbukott hallgatók lemaradnak a társaikhoz képest, új emberek közé kerülnek. A mintatanterv szerint haladók szintén új közösségben találják magukat, a felsőbb évfolyamokról évet ismétlő társaik csatlakoznak hozzájuk. A kötelezően választható tárgyak megjelenésével ez a folyamat kiteljesedik, a hallgatói közösségek az esetek többségében szétszóródnak.

## A probléma fontossága, felvezetése

A tanulás csapatban sokkal hatékonyabb, könnyebb egy tárgyat teljesíteni, ha van lehetőség közös tanulásra, tudásátadásra. A közös tanulás hatékonyabb, minden hallgatónak más képességei vannak, egymástól is sokat tanulhatunk. Mindenkinek más a tanulási stratégiája, más szempontból közelítünk meg egy-egy problémakört, így tanulságos lehet a közös munka. Sok tárgy esetében követelmény is az, hogy a csoporttársainkkal együttműködve oldjunk meg feladatokat, készítsünk közös féléves beadandót. Azok a csapatok, melyek tagjai nem tudnak összhangba kerülni, ugyanolyan energiabefektetéssel rosszabb eredményeket produkálnak, mint társaik.

Az oktatók vagy véletlen sorsolással osztják ki, vagy pedig önálló hallgatói szerveződés során jönnek létre a csapatok. Ritka eset az, mikor egy kurzuson nincs félresikerült beadandó, féléves feladat a rossz kiosztás miatt. Igaz, nem minden konfliktus rossz, hiszen az ilyen feladatok arra is felkészítenek, hogy a való életben is helyt tudjunk állni egy munkahelyen ahol nagy valószínűséggel majd csapatban fogunk dolgozni. Nem mindig a csapatkiosztáson múlik egy-egy kudarc, de úgy hiszem lényeges javulást hozhat, ha valamilyen eszköz ott lenne a hallgatók és az oktatók kezében, mely segítene abban, hogy jól funkcionáló csapatokban dolgozhassanak, tanulhassanak.

## Az alkalmazásról

A fentebb megfogalmazott problémára próbál választ adni az elkészítendő alkalmazásom: ebben a hallgatókat, és az általuk hallgatott tárgyakat, a közöttük lévő kapcsolatokat tároljuk, és a letárolt adatokat alapul véve különböző lekérdezésekkel tanuló-kör, korrepetitor ajánlásokat teszünk számára. Az alkalmazás a hallgatók számára lehetőséget nyújtana arra, hogy a tanuló társaikat ideálisan választhassák ki, illetve a csapatmunkákhoz kialakítandó csapatokat könnyebben szervezhessék meg. A felhasználók számára egy – a letárolt adatokon alapuló - webes applikációt tervezek megvalósítani, hogy a számukra meghatározó információkat könnyen elérjék, és könnyen értelmezhető formában tekinthessék meg.

## Tervezett funkciók

Felhasználókezelés, User- autentikáció

Tanulópartner csoporttárs kereső:

* Előrehaladás figyelembevételével (tanulmányi átlag, tanterv, stb…)
* Szakirány, szakmai érdeklődés figyelembevételével

Csapat-összeállító:

* Csapattársak javaslata, adott órára

Korrepetitor kereső:

* Olyan hallgató(k) ajánlása, akik már teljesítették az adott tárgyat, jó érdemjeggyel, és/vagy vannak közös óráik

A jövőbeni felhasználók igényeinek elemzése során a következő elvárások fogalmazhatók meg az elkészítendő alkalmazással szemben:

Fontos, hogy az alkalmazás tartalmazzon felhasználó-autentikációt, mivel az adatok, melyeket az alkalmazás az ajánlásokhoz használ, nagyrészt személyes, szenzitív adatok, így azok nem lehetnek publikusak mindenki számára. A felhasználói felületnek egyértelműnek és áttekinthetőnek kell lennie. A nyitóoldal az autentikációt fogja tartalmazni mely a bejelentkezés után az alkalmazás főöldalára irányít át. Továbbá a nyitóoldal a regisztrációs oldalra is kell tartalmazzon hivatkozást. A regisztrációs űrlapon a felhasználónak meg kell adnia a tanulmányaival kapcsolatos információkat, a szakmai érdeklődési körét, illetve az oldal használatával kapcsolatos további információkat.

A bejelentkezés után az alkalmazás főoldaláról elérthetőek legyenek a tervezett funkciók. Minden különálló funkció, a csapatösszeállító, a tanuló-társ választó és a korrepetitor-kereső külön menüpontba kerüljön. Az összes funkciónál elérhető legyen néhány beviteli mező a szűrőfeltételek megadására. Ezen szűrő rögzített eleme legyen az oldalnak és alatta jelenjenek meg a keresési eredmények.

A felhasználó igényelhet a GDPR-ban foglaltak szerint adat exportot, illetve teljeskörű fizikai törlést az adatbázisból. Ezért lennie kell adminisztrátor jogkörű felhasználónak is, aki ezeket a kéréseket megkapja, hogy feldolgozhassa.

## Alkalmazandó technológiák

Létrehozandó egy adatbázis, amiben a hallgatók személyes adatait, az általuk hallgatott tantárgyakat illetve ezek közötti kapcsolatokat kell tárolni.

Az üzleti logika, és az adatbázis manipuláció megvalósításához egy szerveroldali webalkalmazást kell megvalósítani, mely kommunikál a fentebb említett adatbázissal és az onnan kinyert adatokat egy interfészen keresztül teszi elérhetővé.

Továbbá egy kliensoldali webalkalmazás szükséges a felhasználói felülethez, amely a szerveroldali interfészen keresztül szerzi meg a felhasználó számára megjelenítendő adatokat.

## Célok meghatározása

Az alkalmazás célja, hogy a felhasználói, az egyetem hallgatói, könnyebben találják meg egymást közös tanulás és szakmai fejlődés céljából. Mindezt egy könnyen értelmezhető és letisztult felület biztosítsa számukra, melyet intuitív módon lehet használni. A felületnek a kialakítása során cél az, hogy a felhasználói élmény pozitív legyen.

Fontos az aktuális technológiai lehetőségeket áttekintve a feladathoz megfelelő, egymáshoz jól illeszthető technológiák kiválasztása.

A fejlesztéssel kapcsolatban az a cél, hogy az elkészült alkalmazás az aktuális biztonsági elvárásokat teljesítse, könnyen karbantartható, rétegzett architektúrájú legyen. Az alkalmazás minél nagyobb tesztlefedettséggel rendelkezzen.

# Irodalomkutatás

A különböző alkalmazandó technológiákat a következőkben megvizsgálom, a feladat szempontjából a legalkalmasabbat keresve. TODO: valami jobb bevezető szöveg, miért jönnek a relációs adatbázisok ide rögtön…

## Adatbázisok

Az elkészítendő alkalmazásom lényeges része az adattárolás és az adatokon végzett műveletek. A következőkben megvizsgálom, hogy milyen lehetőségek vannak az ilyen nagy adatmennyiség kezelésére, ahol az adatok emberek adatai, akik között sokféle és nagy mennyiségű kapcsolati információ áll rendelkezésre.

Néhány adatbázisok vizsgálatához szükséges elv, tétel ismertetése:

**CAP-tétel:** Eric Brewer tétele, melyben kimondja, hogy az alábbi tulajdonságok közül kettőnél többet nem valósíthat meg egy elosztott rendszer:

* Konzisztencia (Consistency)
* Rendelkezésre állás (Availability)
* Partíciótűrés (Partition tolerance)

**ACID elvek:** Egy adatbázis tranzakciófeldolgozó képességének alapelemei, melyek az adatbázis mindenkori integritásához szükségesek. Az elvek a következő tulajdonságokat követelik meg:

* Atomicitás (Atomicity) – Több művelet egységként kezelése, vagy sikeresen végrehajtódik minden művelet, vagy egy sem.
* Konzisztencia (Consistency) – Az adatok a tranzakció előtti állapotból ismét érvényes állapotba kerüljenek annak végeztével.
* Izoláció (Isolation) – Több tranzakció egyidejű végrehajtása során az egyes tranzakciók nem lehetnek hatással egymásra.
* Tartósság (Durability) – Tranzakciós log-ot olyan módon kell tárolni, hogy az esetleges szoftveres, vagy hardveres meghibásodás után is megmaradjon.

**BASE elvek:** Szintén a tranzakciókezelésre vonatkozó alapelvek, a teljesítendő tulajdonságok a következők:

Általános rendelkezésre állás - Basic Availability: Az adatbázis majdnem mindig elérhető

Megengedő állapot - Soft state: Az adattárolóknak nem kell mindig írás-konzisztensnek lenniük, sem a replikáknak

Konzisztencia - Eventual consistency: Egy idő után konzisztens lesz az egész rendszer, ha addig nem érkezik új kérés.

### NoSQL adatbázisok

A hagyományos relációs adatbázisok az adatokat táblákban tárolják, és az adat sémát gondosan megtervezik az adatbázis építése előtt, az adatok konzisztenciája biztosított minden esetben, melynek következménye a merevség. Relatív kevés adat tárolására tervezve. Nem tudnak kielégítő választ nyújtani a mai követelményekre: webes technológiák, óriási adatmennyiség, skálázhatóság, rugalmas adatstruktúra és objektumközpontúság. Erre megoldást az ún. következő generációs, nem relációs adatbázisok próbálnak adni.

A NoSql adatbázisok az alábbi feltételeknek próbálnak megfelelni:

* Nem relációs
* Elosztott
* Open-source
* Horizontálisan skálázható

Ezek a fő irányvonalak, de gyakran további jellemzők is érvényesek a NoSQL megoldásokra, mint:

* Séma mentesség
* Replikáció támogatás
* API[[1]](#footnote-1) szolgáltatás
* ACID helyett BASE elvek
* Big data

Gyakran definiálják még a NoSQL-t not only SQL-ként, az ilyen adatbázis rendszerek gyakran rendelkeznek olyan lekérdező nyelvvel, mely hasonlít az SQL-re szemantikáját vagy felépítését tekintve. A NoSQL rendszereket nem a klasszikus relációs adatbáziskezelő rendszerek leváltására használják a nagyvállalatok általában, hanem szűkebb, operatív célokra. Ahol fontos a magas szintű adatkonzisztencia, ezeket egészítik ki a relációs adatbázisokkal.

Fajtáit az adattárolás elve szerint csoportosítva mutatom be:

**Kulcs-érték tárolók**

A kulcs-érték tárolók vagy kulcs-érték adatbázisok egy egyszerű adatmodellt valósít meg, ahol egy egyedi kulcshoz párosít egy értéket, mint egy asszociatív tömb. Az adattípusokat rugalmasan kezeli. A modell nagymértékben skálázható, a webalkalmazások gyorsítótárazásához illetve munkamenet (session) – kezelésénél jó teljesítményt nyújt. A különböző implementációk abban különböznek, hogy memóriát (RAM), SSD[[2]](#footnote-2) meghajtót vagy merevlemezt (HDD) használnak. Ilyen adatbázisok például: Aerospike, BerkeleyDB, Redis, Riak vagy LevelDB.

**Dokumentumtárolók**

A dokumentumtárolók félig strukturált adatokat és azok leíróit tartalmazzák egy adatszerkezetben. Tartalmazhat kulcs-érték párokat, tömböket, vagy akár beágyazott dokumentumokat is. A dokumentum adatbázisok használatának elterjedése a webes technológiákhoz köthetőek, azon belül is a JavaScript-hez és a JSON[[3]](#footnote-3)–höz, mely a webes kommunikáció egy széles körben használt adatformátuma. A dokumentumtárolókat CMS rendszerek illetve mobil applikációk adattárolására használják leginkább. Ilyen adatbázisokra példák: CouchDB, DocumentDB, MongoDB, Elastic vagy Couchbase Server.

**Oszlopcsaládok**

Hasonlít a relációs adatbázisok szerkezetére, de az adattáblákat sorok helyett oszlopok szerint rendezve tárolja. A sorok felépítése: kulcs és oszlop. Kétdimenziós kulcs-érték tárolóként is megvalósítható. Általában ajánló-rendszerekhez, kibertámadás- csalás felderítésre, és különböző adatfeldolgozásra használják. Ilyen adatbázisokra példák: HBase, MapR, Cassandra, MonetDB, HyperTable vagy Amazon SimpleDB.

**Gráfadatbázisok**

A gráfadatbázisok csomópontok és élek halmazaként szervezik az adatokat, ahol a csomópontok feleltethetők meg a rekordoknak és az élek a közöttük lévő kapcsolatoknak. Mivel az adatok közötti kapcsolatokat a gráf élekként tárolja, így gazdagabb lehetőségeink vannak az adatok közötti kapcsolatok modellezésére. Továbbá az adat szerkezetét tekintve a relációs adatbázisok szigorú séma leíróihoz képest a gráf adatbázis adatmodellje képes a az esetleges változásokhoz alkalmazkodni és fejlődni vele. Olyan rendszerekben használt, ahol fontos az adatok közötti kapcsolat: CRM[[4]](#footnote-4) rendszerek, közösségi hálók, útvonaltervezők vagy webáruházak termékajánló rendszerei. Például ilyen adatbázisokra: TITAN, Sparksee, Neo4j, Trinity, OrientDB vagy InfiniteGraph.

A feladat szempontjából az ideális választás a gráfadatbázis, mert az adatok közötti kapcsolatokat adatszerkezeti szinten tárolja, és ezáltal a kapcsolatok elemzése egyszerűbb, gyorsabb mint a többi NoSQL típus használatával.

### Gráfadatbázisok

A gráfadatbázisok részletes vizsgálata előtt érdemes a gráf pontos definíciójával megismerkedni.

A gráf meghatározása: (2)

G (V,E) gráf egy rendezett pár, ahol V egy nem üres véges halmaz, a pontok (csúcsok) halmaza, E pedig a V halmaz elemeiből képzett (nem feltétlenül különböző) párok egy halmaza. Ez utóbbit az élek halmazának nevezzük. Ha az éleknek irányítása van, azaz az élek E(G) halmazának elemei rendezett párok, akkor a gráfot irányított gráfnak nevezzük és G(V,E)-vel jelöljük.

**A gráfadatbázis [2]**

A gráfadatbázis-kezelő rendszer, a továbbiakban: gráfadatbázis, egy olyan online adatbázis-kezelő rendszer, mely létrehozás, olvasás, frissítés, és törlés (CRUD) műveletek segítségével manipulálja a gráf adatmodellt. A gráfadatbázisok többnyire tranzakcionális (OLTP) rendszerek használatához lettek létrehozva. Ebből kifolyólag tranzakcionális performanciára optimalizáltak.

Két szempont szerint csoportosíthatjuk magas absztrakciós szinten a gráfadatbázisokat:

A fizikai adattárolás módja:

* Natív gráf tárolás, ahol az adattárolás fizikailag is gráf adatstruktúrában van megvalósítva.
* Szerializált, ahol a gráf adatait relációs-, objektum- vagy más általános célú adatbázisokban tárolják.

A feldolgozást végző engine:

* Natív gráf feldolgozás: Index-mentes, vagyis az adatbázisban egy V1 csomópont fizikai mutatót tartalmaz V2 csomópontra, amennyiben vezet él a V1-ből V2-be.
* Globális index használat

A gráfadatbázis szignifikáns része a kapcsolat a csomópontok között. Más adatbázisoknál ezt idegen-kulcsokkal, map-reduce-val stb... valósítjuk meg. A csomópontok és az élek absztrakcióinak összekapcsolásával tetszőlegesen részletes modellt alkothatunk, ami gyakran közel van a valós problémához. A létrejött modell intuitív módon értelmezhető, kifejező.

**Adatmodell szerinti csoportosítás:**

**Property graph:** Csomópontokból és élekből áll. A csomópontok tulajdonságokkal és címkékkel rendelkezhetnek. A tulajdonságokat kulcs-érték párokként adhatjuk meg. Az élek kapcsolatok a csomópontok között. Az éleknek van neve és iránya, mindig pontosan egy forrás-csomóponttal és pontosan egy cél-csomóponttal. Az élek is rendelkezhetnek kulcs-érték párral definiált tulajdonsággal.

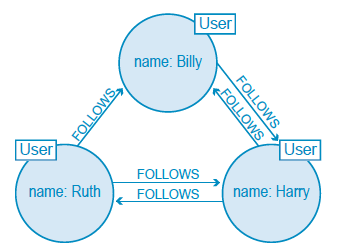
**Hypergraph**: Egy általános gráfmodell melyben egy él, vagyis ún. hyper-edge akármennyi csomópontot összeköthet. A hyper-edge forrása és vége is lehet több csomópont. A hyper-gráffal könnyen modellezhetünk több a többhöz kapcsolatot. Hasonlít a property graph-ra, attól egy általánosabb modellt valósít meg.

**Triple**: A triple egy alany-predikátum-tárgy felépítésű adatszerkezet. Például.: *„A csomópont – része- a gráfnak”.* Amennyiben sok ilyen felépítésű adat áll rendelkezésünkre, akkor kapcsolatokat deríthetünk fel az adatok között. RDF és SPARQL kompatibilis általában.

A gráfadatbázis fajtákat áttekintve a triple tárolókat ki lehet zárni, mint lehetséges választás, mert inkább statikus adathalmazra érdemes használni, ellenben az alkalmazás adatstruktúrája dinamikus kell legyen (pl.: Neptun változások, felhasználói igények változása) és a lekérdező nyelve, a SPARQL pedig lassú. A hypergraph is megfontolandó választás lehet, azonban a feladat szempontjából nem elég a kapcsolatok meglétét ismernünk, hanem annak tulajdonságait is szükséges tárolnunk. Mindezek figyelembevételével a property graph adatmodell illeszkedik leginkább a feladathoz. A következőkben ilyen adatmodellt használó gráfadatbázisokat vizsgálok meg.

**Labeled property graph**

A property graph adatmodell egy széles körben alkalmazott megvalósítása mely címkékkel egészíti ki az alap property graph adatmodell struktúráját. A felépítése egy példán keresztül bemutatva:



1. Ábra - gráf adatmodell

Vegyük példaként a Twitter alkalmazást. Van 3 felhasználó, őket 3 csomópont reprezentálja. A csomópontoknak lehetnek tulajdonságai, mindenkinek eltároljuk a nevét egy tulajdonságként, ahol a kulcs a „name” lesz. Mindannyian kapnak egy [User] címkét, ami felhasználóként kategorizálja őket, ettől kezdve a felhasználók halmazába kerülnek. Ezek után felépítjük a kapcsolati rendszert köztük, Ruth követi Billy-t, Billy követi Harry-t és fordítva, majd Harry követi Ruth-t és fordítva. Minden kapcsolat egy irányított éllel reprezentálható.

### Neo4j

A Neo4j az egyik legelterjedtebb gráf adatbázis. Tranzakciói megfelelnek az ACID elveknek. CAP-tétel szerint A-P. Java nyelven implementálták, elérése kliens alkalmazásokból a Cypher lekérdező nyelvvel tranzakcionális HTTP végponton, vagy a bolt protokollon keresztül érhető el. Létezik open source Community verziója, ez egy példány futtatására ad lehetőséget. Az Enterprise verzió rendelkezik online backup-val, skálázható. Jól dokumentált, .Net, Java, R, JavaScript, Python, PHP programozási nyelvekhez rendelkezik támogatással.

**Szerverarchitektúra : Causal Cluster**

Három fő célja van ennek a felépítésnek: biztonság, skálázhatóság és általános konzisztencia. A biztonságot a core szerverek biztosítják. A skálázhatóságot a read-replika szerverek biztosítják. Az általános konzisztencia azt biztosítja, hogy egy kliens-alkalmazás legalább a saját írás műveleteit lássa. [16]

**Adatmodell**

A neo4j adatmodellje a fentebb ismertetett labeled property graph.

**Cypher**

Deklaratív gráf lekérdező nyelv. A lekérdezéssel azt kell megfogalmazni, hogy az adatot milyen meghatározott minta alapján szeretnénk kiválasztani az adathalmazból. Egy egyszerű példa egy cypher utasításra:

MATCH (u) RETURN u

A MATCH utasítás valamilyen minta szerint listázza a node-okat. Ebben az esetben az összeset.

Példa szelekcióra, a labeled property graph Twitter példájánál maradva, mely a Ruth által követett embereket keresi, a következőképp épül fel:

MATCH (r:User {name:’Ruth’})-[:FOLLOWS]->(u:User)

Zárójelekkel hivatkozunk egy csomópontra: (r : User). Az r változónévvel hivatkozhatunk később a csomópontra. A {}-ek között megadhatunk további megszorításokat a kereséshez. A –[:Follows]-> szintakszissal pedig a keresendő kapcsolatok tulajdonságát adhatjuk meg. Tehát a ’Ruth’ nevű u csomópontokkal :Follows tulajdonságú kapcsolatban alló csomópontokat keressük.

### OrientDB

Open source NoSql adatbáziskezelő rendszer, multi-modell kategóriába sorolható. Támogat gráf, dokumentum, kulcs-érték és objektum-tárolási modelleket. A relációk az adatok között minden esetben gráfokhoz hasonlóan, fizikai mutatóval tároltak. Lekérdező nyelve a Gremlin, de emellett gráfbejárással bővített SQL-t is támogat. Java nyelven íródott. ACID elveknek megfelelő tranzakciókezelést használ. A fizikai adattárolás módját tekintve natív. [17]

**Adatmodellje**

Labeled property graph.

**Szerverarchitektúrája**

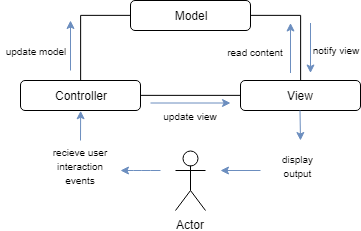
Multi-master replikáció, lehetővé teszi a földrajzilag elkülönített klaszterek létrehozását.

A két gráfadatbázis között nehéz dönteni, szinte ugyanolyan funkcionalitást kínálnak, és a gráfadatmodelljük is egyforma, illetve mindkettőnek van intuitívan elsajátítható lekérdező nyelve. A választást így a dokumentáció mérete és minősége, valamint az elérhető mintaprojektek alapján érdemes meghozni. Ez alapján a Neo4j a jobb választás, rengeteg oktatóanyag és részletes dokumentáció áll rendelkezésre a hivatalos weboldalukon.

## Szerveroldali technológiák

A webalkalmazás elkészítéséhez szükséges egy szerveroldali alkalmazás, amit egy webszerver által futtatva a felhasználók számára elérhetővé válik az alkalmazás. A továbbiakban megvizsgálom, hogy milyen technológiai lehetőségek közül érdemes választani és milyen tervezési javaslatok léteznek az architekturális felépítésre, milyen szabványokat kell ismerni.

**MVC tervezési minta**

****

**2**. Ábra - MVC sematikus ábra

**3**. Ábra - MVC sematikus ábra

Az MVC-Model-View-Controller egy szoftvertervezési, architekturális tervezési minta. Az alkalmazást 3 rétegre osztja, modell (Model), nézet (View) és vezérlő (Controller). A modell tartalmazza általában az üzleti logikát és az adatokat, a nézet a felhasználói felületet és a vezérlő pedig a vezérlési szerkezeteket. Így az alkalmazás rétegei nem függenek szorosan egymástól, ha egy komponenst cserélni kell, akkor elég azt módosítani, a többi rétegen nem kell változtatnunk. A tervezési minta meghatározza ezen komponensek és a felhasználó interakciói közötti kapcsolatokat.A vezérlő felhasználói interakciót fogad, és átkonvertálja utasításokká a modell, vagy a nézet felé. A nézet az adatok reprezentálásáért felelős, valamilyen felhasználó számára értelmezhető módon. A modell a központi eleme a tervezési mintának, teljesen független a felhasználói felülettől. Közvetlenül kezeli az üzleti logikát, az adatokat. [3]

Implementációtól függ ezen kapcsolatok pontos megvalósítási módja. A különböző technológiák MVC megvalósításában lehetnek eltérések ettől a modelltől. Az elkészítendő alkalmazást ennek a tervezési mintának alapján tervezem elkészíteni.

### ASP.NET Core

Microsoft által fejlesztett open-source webes keretrendszer, szerveroldali C# webapplikációk fejlesztéséhez. Windows, Linux, macOS és Docker kompatibilis. [4]

**ASP .NET Core MVC**

Keretrendszer szerver és kliensoldali alkalmazásfejlesztéshez ASP[[5]](#footnote-5).NET Core-ban. A keretrendszer MVC tervezési minta alapján sok előre definiált elemet tartalmaz a webfejlesztéshez:

* Routing
* Model binding
* Model validation
* Dependency injection
* Web APIs
* Razor view engine
* Tag Helpers
* View Components

Saját webszervere van, a Kestrel, ez Windows és Linux operációs rendszerekre is telepíthető.

### Java EE

Java Platform Enterprise Edition, szerveroldali programozási platform. Legfrissebb verziója a Java EE 8. Sok különböző forrású API-t tartalmaz, melyek lehetővé teszik az alkalmazás rétegzését, elosztott működését. Közösségi fejlesztésű szabvány definiálja. Ez egy de facto szabvány, nem hivatalos, de csak az ezt megvalósító alkalmazások lesznek Java EE kompatibilisek. [5]

Többek között a következő API-k a részei (Java SE API-val együtt): [6]

* EJB - Enterprise JavaBeans Technology
* Java Servlet Technology
* JavaServer Pages Technology
* Java API for JSON Processing Java API for RESTful Web Services
* Dependency Injection for Java
* Java Message Service API
* Java API for WebSocket

A Java EE alkalmazásszerverek képesek kezelni a fentebb említett API-k összességét, integrálni tudják azokat, hogy egy jól működő webalkalmazást építhessünk össze belőlük, melynek csak az üzleti logikájára kell koncentrálnunk. Ilyen Java EE 8 alkalmazásszerver a GlassFish Server, Red Hat JBoss Enterprise Application Platform, vagy az IBM WebSphere Application Server. [7]

### NodeJS

Open-source futtató környezet a JavaScript böngészőn kívüli futtatásához. A Google Chrome c++-ban írt V8 engine-jét használja, ami gépi kódra fordítja a JavaScriptet. Esemény vezérelt architektúrájú, képes aszinkron I/O művelet végrehajtásra. A NodeJS használatával az egész webalkalmazás, szerveroldal és kliens íródhat egyazon programozási nyelven, JavaScriptben. [8]

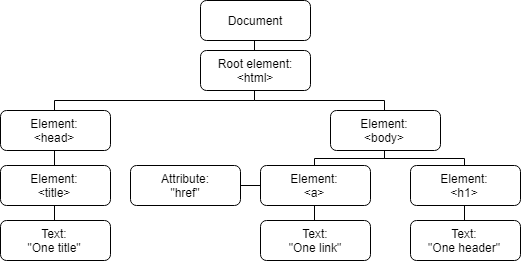
A szerveroldali alkalmazáshoz ASP .NET Core-t választom, mivel megfelelően tesztelt, robosztus és az alkalmazandó MVC tervezési mintához sok előre definiált elemmel rendelkezik, valamint platformfüggetlen. A választott adatbázishoz, a neo4j-hez is rendelkezik driver-rel. A NodeJS nem rendelkezik olyan kiforrott megoldásokkal, sok a community fejlesztésű komponens hozzá, nincs jól támogatott kivételkezelés hozzá, így a .NET–es megoldás sokkal megbízhatóbb ezekből a szempontokból. A JavaEE hasonló funkcionalitást ad mint az ASP .NET Core, a választás közöttük az alábbi publikáció alapján történt [12]. A publikáció teszteredményei alapján a dinamikus JSON olvasás/írás .NET környezetben nagyságrendekkel gyorsabb, mint a JavaEE-ben. Ez fontos az elkészítendő alkalmazás szempontjából, mert a kommunikáció a Neo4j adatbázissal JSON formátumban támogatott.

## Felhasználói felület

Az alkalmazás elkészítéséhez szükséges egy kliensoldali alkalmazás fejlesztése, melynek segítségével a felhasználók könnyedén hozzáférhetnek a kínált funkciókhoz. Fontos a felhasználói élmény, mely érdekében egy letisztult, ergonomikus felhasználói felületet szeretnék létrehozni. A továbbiakban megvizsgálom, hogy milyen technológiai lehetőségek állnak rendelkezésre ennek a megvalósításához valamint milyen alapfogalmak szükségesek ezek megértéséhez.

**DOM - Doucument Object Model :**

Platform és nyelvfüggetlen interfész, ami programok és szkriptek számára lehetővé teszi hogy elérjék és módosíthassák a struktúráját és stílusát a HTML, XML és XHTML (valamint ezekhez hasonló fa struktúrájú) dokumentumoknak. A dokumentum a módosítás feldolgozását követően általában visszatöltésre kerül pl. a böngészőben.



3. Ábra - A DOM felépítése

Amikor egy weboldal betöltésre kerül, akkor a böngészőben létrejön a HTML oldal alapján a HTML DOM, ami egy fa struktúrájú, objektumokból álló architektúra. [9]

### Javascript:

A JavaScript egy objektum-alapú, nem típusos script nyelv, mely az ECMAScript szabványt valósítja meg. Elkülöníthetjük a kliens illetve a szerveroldali alkalmazását.

Kliensoldali alkalmazása a HTML oldalak dinamikussá tételéhez használatos. A HTML-kódba ágyazott JavaScript kódot a böngésző megjelenítés helyett értelmezi azt, és futtatja (egy JS engine segítségével pl.: V8). [10]

### Angular

Az Angular egy platform, és keretrendszer a kliensoldali alkalmazások fejlesztéséhez HTML-ben és TypeScript-ben. A TypeScript egy olyan ECMAScript szabványt megvalósító típusos nyelv ami JavaScriptre fordul, és annak egy kibővítése. A TypeScript nyílt forráskódú, és a Microsoft támogatja. [11]

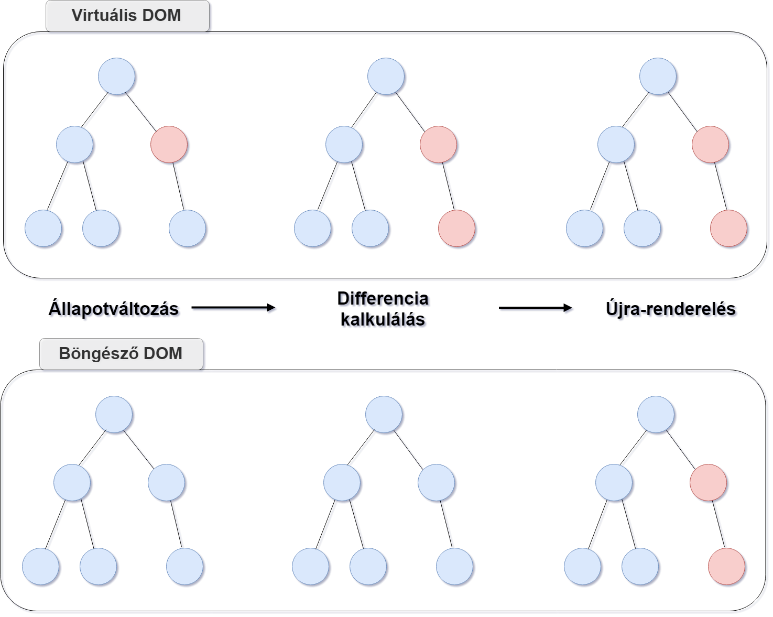
Az Angular gazdag lehetőségeket kínál a különböző előre gyártott sablonnal, tesztelhető, és független a szerveroldali technológiától. Főbb előre definiált funkciók: Web API[[6]](#footnote-6) kezelés, függőség injektálás (dependency injection), kétirányú adatkötés, form validáció. Az Angular platformfüggetlen, fejlesztését a Goog

le támogatja.

### React

A React egy deklaratív JavaScript könyvtár felhasználói felület készítéshez, melyet a Facebook hozott létre. Általában single-page, vagyis egyoldalas weboldalakhoz használják. A felhasználói felületet több kisebb összefüggő kódrészletből, vagyis komponensből felépítve állíthatjuk össze. [15]

A React működése a virtuális DOM-ra alapul, ami a DOM és a generált HTML közé tehető. A virtuális DOM egy memóriában lévő gyorsítótárazott megfelelője a DOM-nak. Felismeri, hogy a kódban történő változások által mely részét kell a virtuális DOM-nak frissíteni. Végül a React a virtuális DOM-nak csak az új részeit másolja át a DOM-ba, ahelyett hogy teljesen felülírná az egészet. Ez a React fő erőssége, mivel ez a módszerrel optimalizálhatjuk a dinamikus webalkalmazásunk renderelési idejét.



4. Ábra - A virtuális DOM működése

Egy webalkalmazás elkészítéséhez mindezek mellett még számos funkcionalitásra szükségünk lehet, mint a routing, API kommunikáció stb.., ezért más JavaScript könyvtárakkal együtt érdemes használni.

### Java Server Faces – helyett vue.js?

Java keretrendszer webalkalmazás felhasználói felületének készítéséhez. MVC tervezési mintát használ, komponensvezérelt, a Java Server Pages technológiájára épít. A felület leírója az ún. Facelet, ami XML konfigurációs fájlokat használ. A keretrendszer lehetőséget biztosít a felületi elemek adatkötéséhez, és szerveroldali eseményekhez rendeléséhez. Sok előre definiált UI alapkomponenst tartalmaz, és lehetőséget ad a saját komponensek definiálására.

Egy ún. FacesServlet dolgozza fel a kliensoldali kéréseket, ami betölti a megfelelő View-template-et, legenerálja a komponensfát, és a megfelelő események lekezelése után visszaküldi az ezek alapján létrejött HTML választ. A UI komponensei minden válasz előtt mentésre kerülnek és a következő kérésnél (a view következő előállításakor) újra betöltésre kerül. [18]

Az Oracle támogatja, nagyon részletes dokumentációja van, egy kiforrott technológia.

### ASP .NET Core MVC Razor engine -??

Az ASP .NET Core MVC alkalmazásokhoz a beépített felületi komponensek a View-k. A keretrendszerben definiált view-k renderelése a Razor engine-nel hajtódik végre. A Razor egy olyan nyelv, melyben a felületeket html-be ágyazott C#-kóddal definiálhatjuk. Unit tesztelhető. Működik vele a kódkiegészítés.

A felhasználói felület kiválasztása során az egyik szempont a függetlenség a szerveroldali technológiától, mivel így a szerveroldal esetleges cseréjénél nem kell a kliensoldalt is cserélni. Az ASP .NET-hez tartozó Razor, és a Java EE-hez tartozó Java Server Faces így nem megfelelő, mivel azok technológiailag kapcsolódnak a szerveroldali alkalmazáshoz. Az Angular és a ReactJS, valamint a tiszta JavaScript ebből a szempontból megfelelő választások lehetnek. Az Angular inkább követi az objektum-orientált elveket, typescript nyelven lehet fejleszteni benne, ami típusos nyelv és komplett szolgáltatáscsomaggal rendelkezik. A ReactJS, csak egy JavaScript könyvtár, JavaScript-ben kell fejleszteni és egyénileg kell összeállítani a számunkra megfelelő komponenseket. A tisztán JavaScript kliensoldal létrehozásához nagyon sok funkcionalitást kellene egyénileg lefejleszteni, miközben ezekre létezik már számos kiforrott implementáció, így ez a lehetőség nem ideális választás. Ezen szempontok alapján a Neo4j adatbázishoz és ASP .NET Core szerveroldali alkalmazáshoz az Angular tűnik a jó választásnak.

## Hasonló rendszerek bemutatása

Az alábbiakban az alkalmazásomhoz hasonló, publikus webalkalmazásokat vizsgálom meg, milyen funkciókat kínálnak, milyen módszereket használnak a tanuló-társ szűrésre. TODO: rendszerfelépítés vizsgálata a funkcionális mellett.

### MoocLab

MOOC – Massive Open Online Courses

Egy online tudásbázis, MOOC rendszer nyílt kurzusokkal, nagy felhasználóbázissal a világ minden részéről. A study buddy modul releváns a vizsgálat szempontjából. A study buddy egy olyan személy aki hasonló érdeklődési körű, és a cél a jövőben a közös tanulás, készülés vizsgákra és a követelmények közös teljesítése vele az online kurzusokon, vagy a saját oktatási képzésben. [13]

**Kínált funkciók:**

Regisztráció után beállítható a személyes érdeklődés, tanult tárgyak, tudományterületek, rövid bemutatkozás, mely publikus lesz a többi felhasználó számára. Egy tanuló-társ a különböző szűrőfeltételek megadásával kereshető. A szűrőfeltételek alapján kiadott találatok közül lehet választani, hogy kivel szeretnénk felvenni személyesen a kapcsolatot, és annak egy privát üzenetet küldhetünk.

### Stacks

MOOC felhasználóknak készült, több forrású online kurzus elérhető például Udacity, Udemy, vagy edX kurzusai. A tanulók közötti kapcsolatokra helyezik a hangsúlyt, az emberekkel, akiket ismersz már az oldalon (study buddy-d), felveheted a kapcsolatot, hogy segítsétek egymást. Továbbá lehetőséget ad új emberek megismerésére, olyan módon, hogy ajánlásokat tesz a hallgatott kurzusaid alapján. Megfigyelésem alapján csak a megjelölt kurzusok alapján szűri meg a felhasználókat, de a pontos működést nem sikerült felderíteni. [14]

Kínált funkciók:

* User kezelés
* Kurzusok keresése
* Ismerős ajánlás
* Privát üzenet

Az elkészítendő alkalmazás szempontjából érdemes lehet megfontolni a következő funkcionalitást, mely szerint akikkel már valamilyen kapcsolatban áll egy hallgató, azt az ajánlások során előnyben részesíti majd, illetve több opcionális szűrőfeltétel megadásával kereshet további hallgatókat. Ellenben azt nem tartom jó funkcionalitásnak, hogy az egymás számára ismeretlen felhasználók, akiknek nincs közös kurzusuk, nem vehetik fel egymással a kapcsolatot. Akár a szakmai érdeklődésük alapján is alakíthatnak jó tanuló-párosokat. A vizsgált rendszerekben, miután találtak egy potenciális tanuló-társat, a felhasználók privát üzenet formájában folytathatják a további kommunikációt. A szakdolgozatom szempontjából ez a plusz funkció, úgy gondolom, már túl komplikálttá tenné a fejlesztést, így a lehetséges jövőbeni fejlesztések közé tenném. A gráfadatbázis ilyen szempontból is jó választás, mivel egy későbbi kiegészítő funkció hozzáadása is kivitelezhető sémaváltoztatás nélkül, mivel az adatmodell rugalmas.

## A megoldási módszer kiválasztása, a választás indoklása

Az alkalmazáshoz kiválasztott alkalmazandó technológiák az irodalomkutatás alapján a következők:

* Adatbázis réteg: Neo4j
* Szerveroldali alkalmazás: ASP .NET Core
* Kliensoldali alkalmazás: Angular

A választás során az egyes technológiáknál vizsgált szempontok mellett fontos volt, hogy egymással hogy köthetőek össze az egyes rétegek. Kompatibilisnek kellett lenniük egymással. Az adatbázis réteg, vagyis a Neo4j rendelkezik .NET-es driverrel, így a szerveroldali alkalmazás és az adatbázis kommunikációja megoldható. A kliens és a szerveroldal között szoros függőség nincsen, a szerveroldal API végpontokra érkező kéréseket szolgál ki. A kliensoldali szolgáltatás ezen API végpontokra küldött kérésekkel gyűjti be a működéséhez szükséges adatokat a szerveroldali alkalmazástól. Az Angular által használt TypeScript erősen típusos nyelv, ami a fejlesztés során könnyebbséget ad, hiszen a C# is egy erősen típusos nyelv, így hasonló programozási szemléletmóddal lehet fejleszteni a szerver és kliensoldali alkalmazást.

# A részletes specifikáció leírása

Az alkalmazástól elvárt működés és annak pontos specifikációja az irodalomkutatás alapján a következő:

Az elkészítendő alkalmazás egy webalkalmazás, mely az egyem hallgatói számára elérhető. Az alkalmazás 2-féle felhasználói szerepkört különböztet majd meg: hallgató illetve adminisztrátor. A hallgató szerepkörbe tartozó felhasználók az adatlapjukat, a csapattárs ajánlót, a korrepetitor keresőt és a tanuló-társ ajánló funkciókat használhatják. Az adminisztrátor szerepkörbe tartozó felhasználók által elérhető funkciók az alábbiak: adminisztrációs kérések fogadása és feldolgozása (új jelszó igénylés, GDPR export/törlés), félévzárás, új félév felvétele a rendszerbe illetve rendszerinformációs irányítópult.

Az alkalmazás használata minden esetben regisztrációhoz kötött, amit a nyitóoldalon lehet elérni. A hallgatói regisztráció része a Neptun rendszerből exportált adatok feltöltése. Továbbá a személyes adatokat, elérhetőséget kell megadni.

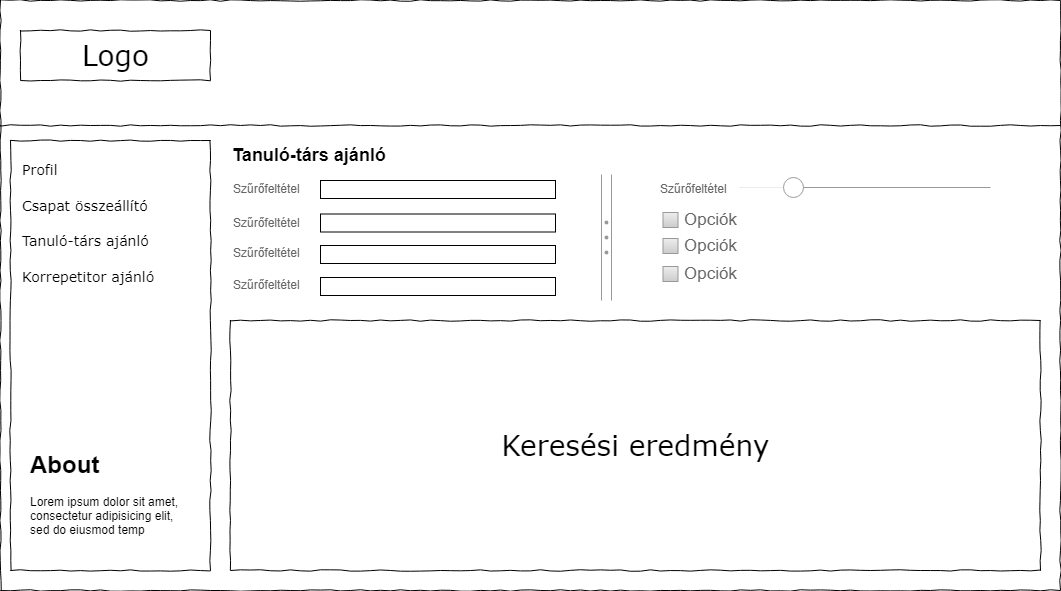
A funkciók használati esetei:

|  |  |  |
| --- | --- | --- |
|  | Hallgató | Adminisztrátor |
| Regisztráció | x |  |
| Adatlap | x |  |
| Csapattárs ajánló | x |  |
| Korrepetitor ajánló | x |  |
| Tanuló-társ ajánló | x |  |
| Adminisztrátor irányítópult |  | x |
| Adminisztrátori kérések listázása |  | x |

A csapattárs ajánló funkció, az aktuális félévben hallgatott tantárgyak gyakorlati és/vagy labor kurzusaira ad csapattárs ajánlásokat, különböző keresési feltételek megadásával.

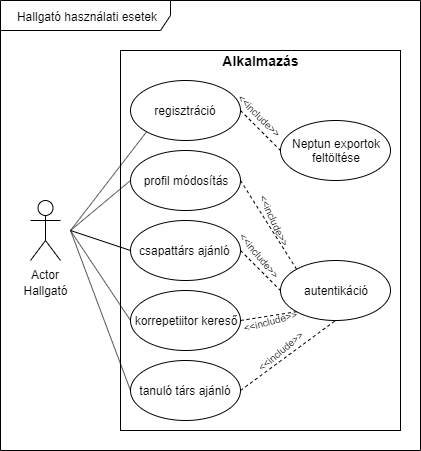
A tanuló-társ ajánló funkció egyes tantárgyakra való készüléshez, tanuló csoport szervezéshez ad hallgató ajánlásokat. A kiválasztott tantárgyon kívül számos szűrőfeltétel megadásával lehet szűkíteni a keresési eredményeket, mint például a:

Az alkalmazás egy koncepcionális képernyőterve:

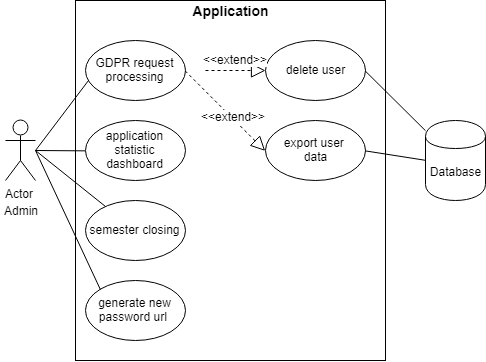


5. Ábra - Képernyőterv

Az összes kereső/ajánló funkció hasonlóképp fog kinézni, egy szűrőfeltételeket összeállító részből és egy keresési eredményeket megjelenítő részből.



6. Ábra - Hallgató használati esetek



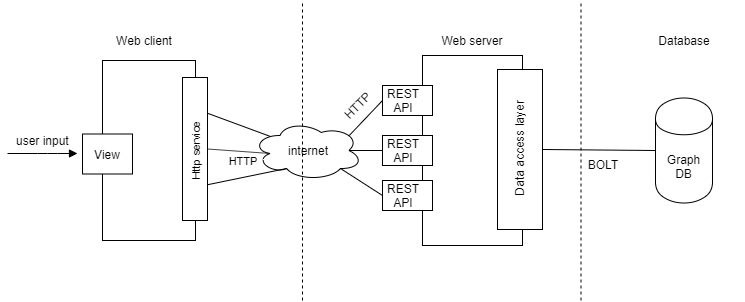
7. Ábra - Adminisztrátor használati esetek

# Tervezés

A tervezést a rendszerterv elkészítésével kezdem, a főbb modulok és azok közötti kapcsolatok leírásával. Az adatforrások, vagyis a Neptun exportok vizsgálata alapján megtervezem a gráf adatmodellt. Az alkalmazás nagyobb moduljainak leírom részletes architekturális tervét.

## Rendszerterv

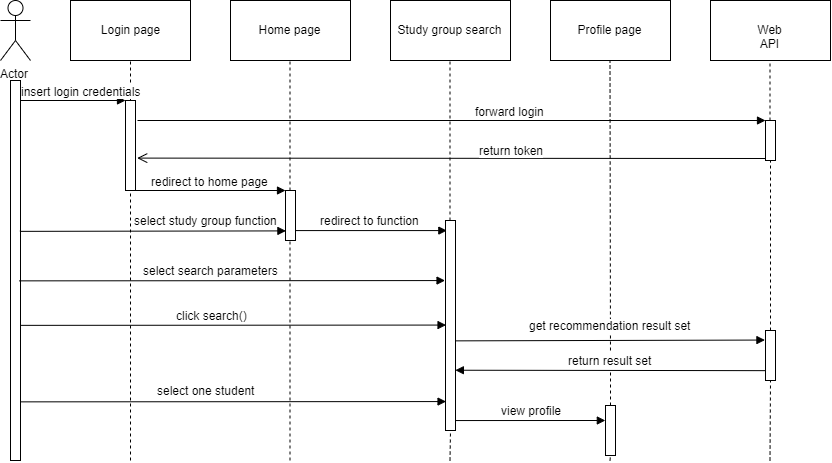
Az alkalmazás három fő részre osztható, web-kliens, webszerver, és adatbázisszerver. A kliensoldali alkalmazás fejlesztéséhez Angular-t fogok használni. A kliens a webszerverhez REST API végpontokon keresztül kapcsolódik, HTTP protokoll használatával, JSON formátumú üzenetekkel. A webszerver REST API végpontokat szolgáltat a kliens számára, melyben JSON üzenteket dolgoz fel illetve küld az számára. A webszerver ASP .NET Core webalkalmazás lesz. Az adatelérési réteg a Neo4j által szabványosított BOLT protokoll használatával kommunikál a gráfadatbázissal. A szerveroldal a Neo4j által szolgáltatott .NET driver segítségével, Cypher lekérdezések formájában küld kérést az adatbázis felé. A gráfadatbázis közösségi verziójú Neo4j lesz.



8. Ábra - Rendszerterv

## Az alkalmazás folyamatai

Az alábbi szekvencia diagram a hallgatói tanuló-csoport ajánló funkció működését írja le. A felhasználó, a kliensoldali alkalmazás és a webszerver kommunikációjára fektetve a hangsúlyt. A keresési eredmények közül lehetőség van kiválasztani egy adott hallgatót, és annak megtekinteni az adatlapját. Az adatlapon megadott elérhetőségek alapján vehetik fel egymással a kapcsolatot.



12. Ábra - Hallgató funkció szekvencia diagramja

## Adatforrások – Neptun Export módjai

A hallgatók saját leckekönyvüket és kurzus listájukat a Neptun-rendszerből exportálni tudják. Az alkalmazás használata ezeknek a Neptun exportoknak a megosztásához lesz kötve (enélkül nem lehet felépíteni a kapcsolati rendszert). Ennek módja a Neptun által generált .xlsx formátumú export-ok, csv formátumra konvertált feltöltése regisztrációkor. A szakdolgozatom keretein belül ezen folyamatokat tesztadatok generálásával fogom kiváltani, a személyiség jogi törvények megsértésének elkerülése végett.

### CSV feldolgozás saját fejlesztésű komponenssel

Az alkalmazáshoz egy export feldolgozó komponenst fogok fejleszteni mely a Neptun-ból kinyert adatok Cypher utasításokká alakításáért lesz felelős. Az exportokon adattisztítást, és normalizálást kell végezni az átalakítás előtt.

A könnyebb feldolgozhatóság érdekében az alkalmazás .csv formátumú Neptun exportok feldolgozására lesz felkészítve. A felhasználók ezt a konverziót egyszerűen elvégezhetik egy táblázatkezelő program segítségével.

**Hallgatói Neptun export:**

Kétféle exportra lesz szükség a megfelelő adatmodell felépítéséhez:

* Az eddig felvett összes tárgy listája: Neptun > Tanulmányok > Leckekönyv > Minden félév > Exportálás Excel fájlba
* Összes hallgatott kurzus listája: Neptun > Tárgyak > Felvett kurzusok > \*aktuális félév > Exportálás Excel fájlba

A Kurzusok Neptun exportjának felépítése a következő:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Tárgy kódja | Tárgy neve | Kurzus kódja | Kurzus típusa | Óraszám | Oktatók | Várólista |
| *NAIHS3STND* | *Haladó szoftvertechnológia* | *ST3\_EA\_01\_ST\_SPEC* | *Elmélet* | *2* | *Oktató neve* |  |

Ebből nem minden információ releváns számunkra, mint például a várólista, vagy az óraszám, ezeket ki kell szűrni a feldolgozás során.

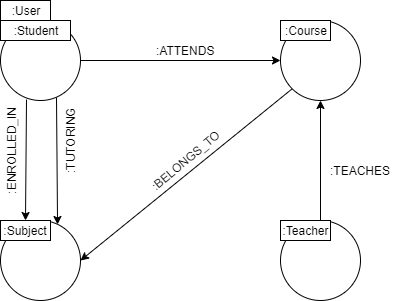
A leckekönyv exportjának a felépítése:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Tárgykód | Tárgy címe, előadó neve | Kr. | Félév | Köv. | Óra heti (E/GY/L) | Óra féléves (E/GY/L) | Aláírás | Jegyek | Megjegyzés | Várólista | Teljesített |

Ezen adatokból a tárgykód, a tárgy előadója, a kreditek, a félév, a jegyek és a teljesített mezők fontosak az alkalmazás szempontjából. A tárgykód és félév alapján kapcsolhatjuk majd össze az egyes tárgyakat a kurzusaikkal.

## Az adatmodell tervezése

Az adatmodell tervezése során az elsődleges szempont az volt, hogy kompatibilis legyen a Neptun-export adatstruktúrájával, mivel az alkalmazás az export feldolgozása segítségével építi fel a felhasználók és a tantárgyak közötti kapcsolatokat. A kapcsolatok felépítése fontos, mivel ez alapján tesz majd az alkalmazás ajánlásokat. Az alábbiakban látható a gráf adatmodell sematikus ábrája.



9. Ábra - A gráf adatmodell

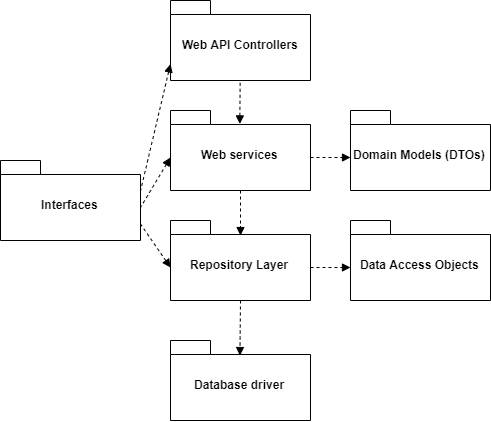
A csomópontok jelölik a fontosabb halmazokat, mint a Felhasználó – [User], Hallgató – [Student], Kurzus – [Course], Tantárgy – [Subject] és az Oktató – [Teacher]. A hallgatók megadhatják, hogy milyen tantárgyakból vállalnak korrepetálást, ezt a [:TUTORING] kapcsolat jelzi. A tantárgyakhoz kapcsolódó kurzusoknál tároljuk a félévet, így egyazon tárgyat hallgatók, akik más félévekben teljesítették azt, kapcsolatban fognak állni egymással. Az adatok alapvető struktúráját mutatja az ábra, ettől az éles rendszerben lehet némi eltérés, lehetnek kivételes esetek (Pl. egy Hallgató demonstrátor, így az Oktató halmaznak is része). A gráf adatmodell ilyen szempontból rugalmas, nem kell minden esetben a fenti sémának eleget tennie, lehet eltérés attól.

* User:
  + User ID
  + User name
  + Email
  + Password
* Student:
  + First name
  + Last name
  + Messenger username
  + Instagram username
  + Profile image path
* Course:
  + Course code
  + Course type
  + Semester
* Subject:
  + Subject code
  + Name
  + Credits
  + Suggested semester
* Teacher:
  + Name
  + Email
* ENROLLED\_IN:
  + Semester
  + Grade

## Szerveroldal tervezése

A szerveroldal ASP .NET Core 2.2 Web API alkalmazás lesz. A szerveroldal REST API végpontokat szolgáltat majd a kliens felé.

REST API: leírás.



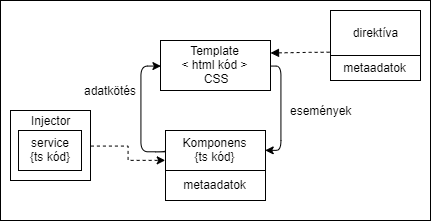
10. Ábra - Szerveroldali alkalmazás csomag diagramja

A szerveroldali alkalmazás projektjeit és azok függőségeit mutatja a csomag diagram. A Web API projekt tartalmazza a controller osztályokat, melyek a bejövő HTTP kéréseket fogadják, feldolgozzák a services réteg hívásával és választ küldenek a hívó fél számára. A web services projekt tartalmazza az üzleti logikát. A repository projekt teremti meg a kapcsolatot az üzleti logika és az az adatelérési réteg (Database driver) között. Repository tervezési minta alapján. Repository tervezési minta leírása.

## Kliens tervezése

A web kliens Angular 8 alkalmazás lesz, melynek felépítését az alábbiakban ismertetem.

Az alkalmazás modulokból épül fel, mindig van egy gyökér-eleme, a root-modul és az tartalmazza a többi alkalmazáslogikáért felelős modult. A modul egységbe zárja a kisebb funkcionális részeket, a komponenseket.



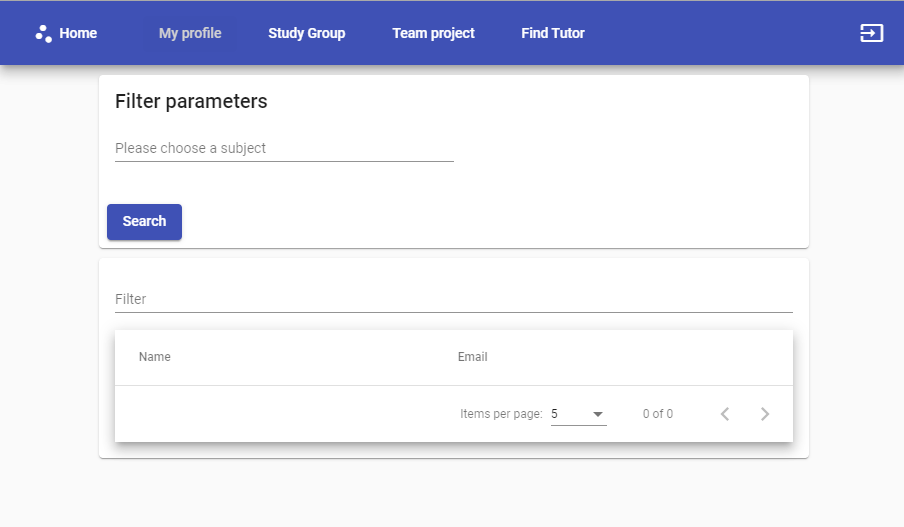
11. Ábra - Angular alkalmazás architektúrája

Egy komponenst egy HTML template, egy CSS stylesheet, és egy typescript osztály ír le. A komponenshez tartozó TS osztály az alkalmazáslogikát és az adatokat tartalmazza és össze van kötve a HTML template-el, ami a megjelenítendő view-t írja le. A HTML template fájlba angular specifikus direktívákat írhatunk, melyek tartalmazhatnak alkalmazáslogikát, módosíthatják a HTML elemeket. A service-ek az alkalmazáslogika azon részét tartalmazzák, mely nem köthető konkrét felülethez, pl.: API hívás, navigáció. A service-eket függőség injektálással használhatjuk a komponensek kódjában.

### Felhasználói felület

A felhasználó felületet tervezésekor fontos szempont volt hogy az egyszerű, letisztult legyen. Az Angular Material Design könyvtárát használtam fel, mely sok előre definiált UI komponenst tartalmaz, melyek reszponzívak, egységes stílusúak.

Az alkalmazásom témája oktatási, illetve közösségi, mely témákhoz gyakran társul a kék szín. Az Óbudai Egyetem arculatának és a Neptun felületének is domináns színe, így én is ezt választottam a kliens tervezése során, hogy jobban illeszkedjen az egyetemi …-ba.



12. Ábra - A felhasználó felület dizájnja

# A megvalósítás leírása

Ebben a fejezetben ismertetem a fejlesztés menetét, a megvalósítás során használt szoftvereket, külső komponenseket illetve az alkalmazott tervezési mintákat. Bemutatom az elkészült rendszer funkcionalitását, működésének folyamatát, a fejlesztés során felmerült problémákat, azok megoldásait.

## Verziókezelés

A git egy nyílt forráskódú, eloszott verziókezelő rendszer. A forráskód Bitbucket-en elérhető, a kliens- és a szerveroldali alkalmazások külön git repository-ban.

## Fejlesztés kezdeti lépései

Ahhoz, hogy az elkészítendő szoftverek fejlesztését elkezdhessem, első lépésként szükség volt az adatbázis létrehozására, és konfigurálására.

Létre kellett hozni egy új neo4j 3.5.12-es Community verziójú adatbázist. Be kellett állítani a BOLT protokollon keresztül milyen port-on legyen elérhető az adatbázis, illetve szükség volt egy felhasználóra, amivel a szerveroldali alkalmazás kapcsolódni tud majd az adatbázishoz. A UNIQUE megszorításokat létre kellett hozni, hogy az adatmodell helyességét biztosítsam:

CREATE CONSTRAINT ON (node:UserName) ASSERT node.UserName IS UNIQUE

CREATE CONSTRAINT ON (node:Subject) ASSERT node.SubjectCode IS UNIQUE

A szerveroldali alkalmazást a tervezés fejezetben leírtak alapján több projektre osztva hoztam létre. A Web API projekt létrehozásakor néhány általános konfigurációs beállítást el kellett végezni a funkciófejlesztés előtt:

* **CORS**: A CORS a Cross-Origin Resource Sharing rövidítése, egy mechanizmus ami a HTTP fejléceket használva mondja meg a böngészőknek, hogy egy webes kliensalkalmazás ami adott helyen fut, milyen más forrású erőforrásokhoz férhet hozzá. Ahhoz hogy a kliensoldali alkalmazástól a szerveroldal HTTP kéréseket fogadjon, konfigurálni kellett a Web API projekt CORS policy-jét.
* **Exception middleware**: Írtam egy saját kivételkezelő middleware-t, mely az API-hívások előtt és után közvetlen futtatásra kerül. A middleware lekezeli a kivételeket és a kliensnek továábítja a hibaüzenetet és a megfelelő http hibakódot. Az API híváson belül keletkezett kivételeket így nem kell külön minden Controller metódusában lekezelni. Ezzel letisztultabbak maradtak a Controller-ek metódusai.
* **Swagger**: Egy külső komponens, mely az API végpontok manuális teszteléséhez generál egy grafikus felhasználói felületet.
* **Serilog logger:** Külső komponens, mely a logolásért felelős.
* **JWT[[7]](#footnote-7) autentikáció:** A kliens- és a szerveroldal biztonságos kommunikációját biztosítja. A sikeres szerveroldali autentikációt követően a szerver egy JWT tokent küld válaszként a kliensnek, ami a további HTTP kérései fejlécében elküldi a szerveroldalnak a token-t, ami validálja azt.

A Web API projektben a .Net Core beépített JWT megvalósítását használva a Controller osztályok metódusait elég Autentikációs és autorizációs annotációkkal ellátni ahhoz hogy megtörténjen a token validálás.

## A fejlesztés menete

Első lépés a szerveroldali alkalmazás architekturális felépítése, és az adatbáziskapcsolat megvalósítása.

A Web API modul . NET Core 2.2 Web Application projekt, a többi modul .NET Core Class Library projekt. (Neo4j .NET Driver n.d.)

A gráfadatbázis és a szerveroldali alkalmazás összekötésénél több drivert is megvizsgáltam, a közösségi fejlesztésű .NET Client-et és a Neo4j által fejlesztett Neo4j .NET Drivert, a továbbiakban Client és Driver néven hivatkozom rájuk. A Client egy „vastag” kliens, BOLT és HTTP protokollt támogat, a lekérdezéseket .NET-es objektumokkal a LINQ-hoz hasonlóan lehet felépíteni. A Driver egy „vékony” kliens, BOLT protokollt támogatja, a lekérdezéseket Cypher nyelven, szövegként kell megírni, de lehetőség van paraméterek (kulcs-érték párok) hozzácsatolásához. A Client képes az adatbázisszervertől visszakapott eredményeket átalakítani C# objektumokká, a Driver nem.

A megvalósítás során a Neo4j .NET Driver-t választottam, mert kisebb, és a Cypher nyelven írt lekérdezések véleményem szerint könnyebben értelmezhetőek, valamint ezt támogatja hivatalosan a Neo4j. A visszatérési értékeket nem tudja átalakítani C# objektumokká, de erre a feladatra találtam egy szintén „vékony” egyszerű külső komponenst, a Neo4jMapper-t, ami ezt a funkcionalitást biztosítja.

**BOLT**: Egy egyszerű hálózati protokoll, adatbázis szerverekkel való kommunikációra. A protokoll utasítás alapú, a kliens szöveges üzenetként a lefuttatandó utasítást és egy paraméterekből álló halmazt küld a szerver felé, ami válaszként visszaadja az utasítás eredményeként kapott rekordokat, és egy üzenetet a futtatás körülményeiről. [boltprotocol.org]

A csv neptunos import-ok feldolgozását a CSV Reader package felhasználásával oldottam meg.

# Tesztelés

A szoftver teszteléséhez szükség volt egy olyan tesztadathalmaz generálására, mely szerkezete hasonlít a hallgatók valós exportjaihoz. Ilyen tesztadatokat egy saját fejlesztésű tool-al generáltam, mely véletlenszerű csv Neptun exportokat generál.

APOC library segítségével importálható az adatbázisba.

Backend tesztelés

Frontend tesztelés

# Az eredmények bemutatása, értékelése, hasonló rendszerek eredményeivel összevetése

# A megvalósítás elemzése, alkalmazásának és továbbfejlesztési lehetőségeinek számbavétele

# A szakdolgozat tartalmi összefoglalója magyarul és angol nyelven

# Irodalomjegyzék

[1] Bagyinszki, J., György, A.: Diszkrét matematika főiskolásoknak. *Typotex,* 2001

[2] Robinson, I., Webber, J., Eifrem, E.: Graph Databases. *O’Reilly Media, 2015*

[3] Leff, A., Rayfield, J.T. Web-Application Development Using the Model/View/Controller Design Pattern. *EDOC*, 2001

[4] ASP .NET Core dokumentáció, Microsoft  
(<https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-2.2>), utoljára megtekintve: 2019.05.10

[5] A Perrone, P.J., A Chaganti, V.S.R.R., A Schwenk, T.: J2EE Developer's Handbook*. Sam's Publishing*, 2003

[6] Java EE dokumentáció,  
(<https://javaee.github.io/tutorial/overview008.html>), utoljára megtekintve: 2019.05.10

[7] Java EE kompatibilitás, Oracle  
(<https://www.oracle.com/technetwork/java/javaee/overview/compatibility-jsp-136984.html>), utoljára megtekintve: 2019.05.16

[8] NodeJS dokumentáció,  
(<https://nodejs.dev/>en/docs/), utoljára megtekintve: 2019.05.16

[9]World Wide Web Consortium,  
<https://www.w3.org/DOM/>), utoljára megtekintve: 2019.05.16

[10] JavaScript dokumentáció,  
(<https://tc39.github.io/>), utoljára megtekintve: 2019.05.16

[11]Angular dokumentáció,  
(<https://angular.io/guide/architecture>), utoljára megtekintve: 2019.05.02

[12] Kronis, K., Uhanova, M.: Performance Comparison of Java EE and ASP.NET Core Technologies for Web API Development. *Applied Computer Systems*. 23. 37-44. 10.2478/acss-2018-0005.

[13] MoocLab,  
(<https://www.mooclab.club/pages/study_buddy/>), utoljára megtekintve: 2019.05.16

[14] Stacks,  
(<https://stacks.courses>), utoljára megtekintve: 2019.05.16

[15] ReactJS dokumentáció,  
(<https://reactjs.org/docs/glossary.html>), utoljára megtekintve: 2019.05.16

[16] Neo4j szerverarchitektúra,  
(<https://neo4j.com/docs/pdf/neo4j-operations-manual-3.5.pdf>), utoljára megtekintve: 2019.05.02

[17] OrientDB hivatalos dokumentáció,  
(<https://orientdb.com/docs/last/index.html> ), utoljára megtekintve: 2019.05.02

[18] Java Server Faces dokumentáció,  
(<https://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html>), utoljára megtekintve: 2019.05.02

# Irodalomjegyzék

Bagyinszki, J., György, A. *Diszkrét matematika főiskolásoknak.* Typotex, 2001.

Bagyinszki, J., György, A.: Diszkrét matematika főiskolásoknak. *Typotex,* 2001

1. Alkalmazásprogramozási interfész [↑](#footnote-ref-1)
2. Solid State Drive [↑](#footnote-ref-2)
3. JavaScript Object Notation – szöveges adatformátum, általában strukurált objektumok hálózati kapcsolaton való továbbítására használják [↑](#footnote-ref-3)
4. Customer Relationship Management – Ügyfél-kezelő rendszer [↑](#footnote-ref-4)
5. Active Server Pages [↑](#footnote-ref-5)
6. Alkalmazásprogramozási interfész [↑](#footnote-ref-6)
7. JSON Web Token [↑](#footnote-ref-7)