

**SAPIENTIA ERDÉLYI MAGYAR TUDOMÁNYEGYETEM
MAROSVÁSÁRHELYI KAR,
INFORMATIKA SZAK**



SAPIENTIA
ERDÉLYI MAGYAR
TUDOMÁNYEGYETEM

SortRace: Rendezési algoritmusok vizualizációja

DIPLOMADOLGOZAT

Témavezető:
Dr. Ing. Kátai Zoltán,
Egyetemi docens

Végzős hallgató:
Lázár Zsolt

2023

UNIVERSITATEA SAPIENTIA DIN CLUJ-NAPOCA
FACULTATEA DE ȘTIINȚE TEHNICE ȘI UMANISTE,
SPECIALIZAREA INFORMATICĂ



UNIVERSITATEA
SAPIENTIA

SortRace: Vizualizator pentru algoritmi de sortare

LUCRARE DE DIPLOMĂ

Coordonator științific:
Dr. Ing. Kátai Zoltán,
Conferențiar universitar

Absolvent:
Lázár Zsolt

2023

**SAPIENTIA HUNGARIAN UNIVERSITY OF
TRANSYLVANIA
FACULTY OF TECHNICAL AND HUMAN SCIENCES
COMPUTER SCIENCE SPECIALIZATION**



SAPIENTIA
HUNGARIAN UNIVERSITY
OF TRANSYLVANIA

SortRace: Vizualization of sorting algorithms

BACHELOR THESIS

Scientific advisor:
Dr. Ing. Kátai Zoltán,
Associate professor

Student:
Lázár Zsolt

2023

Declarație

Subsemnatul/a LAZAR ZSOLT, absolvent(ă) al/a specializării
INFORMATICĂ, promoția 2023... cunoscând
prevederile Legii Educației Naționale 1/2011 și a Codului de etică și deontologie profesională a
Universității Sapientia cu privire la furt intelectual declar pe propria răspundere că prezenta lucrare
de licență/proiect de diplomă/disertație se bazează pe activitatea personală, cercetarea/proiectarea
este efectuată de mine, informațiile și datele preluate din literatura de specialitate sunt citate în
mod corespunzător.

Localitatea, TÂRGU MUREȘ
Data: 15.06.2023

Absolvent

Semnătura... Lazar

UNIVERSITATEA „SAPIENTIA” din CLUJ-NAPOCA
Facultatea de Științe Tehnice și Umaniste din Târgu Mureș
Programul de studii: Informatică

Viza facultății:

LUCRARE DE DIPLOMĂ

Coordonator științific:
dr. ing. Kátai Zoltán

Candidat: Lázár Zsolt
Anul absolvirii: 2023

a) Tema lucrării de licență:

Vizualizator pentru algoritmi de sortare: SortRace

b) Problemele principale tratate:

Studierea algoritmilor de sortare

Clasificarea algoritmilor de sortare

Analizarea instrumentelor de vizualizare a algoritmilor de sortare deja existente

Crearea unui soft-didactic propriu pentru algoritmilor de sortare

c) Desene obligatorii:

Use-case, secvență, arhitectură

d) Softuri obligatorii:

Aplicație pentru vizualizarea algoritmilor de sortare. Mod de lucru interactiv pentru studierea acestora.

e) Bibliografia recomandată:

Kátai Zoltán, C: limbaj și programare, Universitatea Debrecin, Ungaria, 2008.

Kátai Zoltán, Algorithysics: technologically and artistically enhanced computer science education, Editura Scientia, Cluj-Napoca, 2021

f) Termene obligatorii de consultații: lunar

g) Locul și durata practicii: Universitatea „Sapientia” din Cluj-Napoca,
Facultatea de Științe Tehnice și Umaniste din Târgu Mureș.

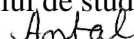
Primit tema la data de: 20.06.2022

Termen de predare: 02.07.2023

Semnătura Director Departament



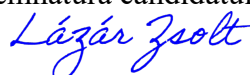
Semnătura responsabilului
programului de studiu



Semnătura coordonatorului



Semnătura candidatului



Kivonat

Dolgozatom témája a rendezési algoritmusok tanulmányozása, és egy interaktív megoldás ezek tanítására.

Az interaktív tanítás napjainkban egyre fontosabb szerepet játszik. Felgyorsult világunkban egyre nehezebb lekötni a diákok figyelmét ezért szükség van olyan pedagógiai módszertanokra, amik segítik a tanár, hogy felkeltse és megtartsa a diákok figyelmét.

Az informatikában, lévén, hogy reál tantárgy, ami logikus gondolkodásra nevel, sok gyakorlati feladat van. Ezek közül egyik alap informatika tananyag a számsorozatoknak a rendezési algoritmusai.

Általam fejlesztett szoftver a SortRace egy olyan internetes oldal, ahol a diákok jobban megérthetik a rendezési algoritmusokat, emellett össze tudják mérni tudásukat egy gyorsasági játék keretein belül.

A dolgozat során be fogom mutatni ennek a szoftvernek a felépítését és a használt technológiák alap működési elveit.

Rezumat

Tema tezei mele este studiul algoritmilor de sortare și o soluție interactivă pentru predarea acestora.

Predarea interactivă joacă un rol din ce în ce mai important în zilele noastre. În lumea noastră în ritm rapid, devine din ce în ce mai dificil să reținem atenția elevilor, prin urmare sunt necesare metodologii pedagogice care să ajute profesorul să atragă și să rețină atenția elevilor.

În informatică, fiind o materie reală care învață gândirea logică, există multe sarcini practice. Printre acestea, unul dintre materialele de bază ale cursului de informatică este algoritmii de sortare a secvențelor de numere.

SortRace, un software dezvoltat de mine, este un site web unde studenții pot înțelege mai bine algoritmii de sortare și pot compara cunoștințele lor în cadrul unui joc de viteză.

În cadrul tezei voi prezenta structura acestui software și principiile de bază de funcționare ale tehnologiilor utilizate.

Abstract

The topic of my thesis is the study of sorting algorithms and an interactive solution for teaching them.

Interactive teaching plays an increasingly important role these days. In our fast-paced world, it is becoming more and more difficult to hold the attention of students, therefore pedagogical methodologies are needed that help the teacher to attract and retain the attention of students.

In informatics, since it is a real subject that teaches logical thinking, there are many practical tasks. Among these, one of the basic informatics course materials is the sorting algorithms of number sequences.

SortRace, a software developed by me, is a website where students can better understand sorting algorithms and can also compare their knowledge within the framework of a speed game.

During the thesis, I will present the structure of this software and the basic operating principles of the technologies used.

Tartalomjegyzék

1. Bevezető	11
2. Rendezési algoritmusok bemutatása	12
2.1. Buborékos rendezés (Bubble Sort)	12
2.2. Minimum-kiválasztásos rendezés (Minimum Sort)	14
2.3. Beszűrő rendezés (Insertion Sort)	14
2.4. Gyorsrendezés (Quick Sort)	15
2.5. Összefésüléses rendezés (Merge Sort)	17
3. A projekt célja	20
3.1. Rendezési algoritmusok vizuális módon történő tanítása	20
3.2. Versenykörnyezetben történő tanulás	20
4. Követelményspecifikációk	22
4.1. Felhasználói követelmények	22
4.2. Rendszerkövetelmények	24
4.2.1. Funkcionális rendszerkövetelmények	24
4.2.2. Nem funkcionális rendszerkövetelmények	24
5. Technológiák bemutatása	25
5.1. Frontend - Angular	25
5.2. Backend - Java SpringBoot	25
5.3. Adatbázis - PostGreSql	26
6. Tervezés	27
6.1. Architektúra	27
6.2. Szekvenciadiagramok	28
6.3. Frontend főbb komponensei	30
6.3.1. Usermanager	31
6.3.2. Play komponens	32
6.3.3. Bubblesort komponens	33
6.4. Backend főbb komponensei	34
6.4.1. Játék logika és az osztályok felépítése	34
6.4.2. Endpointok	37
7. Alkalmazás felhasználói felületének bemutatása	39
7.1. Regisztráció	39

7.2. Bejelentkezés	40
7.3. Kezdőoldal	41
7.4. Játék indítása	41
7.5. Játék	42
7.6. Tanulási játékmód kiválasztása	42
8. További fejlesztési lehetőségek	43
Összefoglaló	44
Köszönetnyilvánítás	45
Irodalomjegyzék	46

1. fejezet

Bevezető

A számítógépes programok hatékony működése alapvetően függ a bennük alkalmazott rendezési algoritmusok hatékonyságától és megbízhatóságától. A rendezési algoritmusok olyan speciális eljárások, amelyek segítségével a számítógép rendezett formában tudja kezelni és elérni adatainkat. Az ilyen algoritmusok kulcsszerepet játszanak a keresési, adatbázis-kezelési és optimalizálási feladatokban. Az informatikai oktatásban fontos szerepet játszik ezeknek az algoritmusoknak a tanítása, mivel sokféle algoritmikai stratégiát érintenek és nagyon jól bevezetik ezeket az algoritmikai stratégiákat.

A dolgozat célja, hogy bemutassa a különböző rendezési algoritmusokat és azok vizuális tanításának lehetőségét egy általam fejlesztett szoftverrel, amelyet SortRace-nek neveztem el. A SortRace egy intuitív és interaktív program, amely lehetővé teszi a felhasználók számára, hogy megismerjék és gyakorolják a rendezési algoritmusok működését valós időben.

Az első részben áttekintjük a leggyakrabban használt rendezési algoritmusokat, például a buborékrendezést, a beszűrő rendezést, a kiválasztó rendezést és a gyorsrendezést. Részletesen bemutatjuk az egyes algoritmusok működését, előnyeit és hátrányait, valamint azok idő- és tárigényét.

A második részben bemutatom a SortRace szoftvert, amelynek célja, hogy interaktív módon segítse a felhasználókat megérteni és elsajátítani ezeket az algoritmusokat. A program vizuális reprezentációja lehetővé teszi, hogy a felhasználók valós időben kövessék nyomon az adatok rendezését, és megfigyeljék az algoritmusok lépéseit és működését.

A dolgozat további részeiben részletesen bemutatjuk a SortRace program felépítését, a funkcióit és a használatát. Emellett az általam végzett kísérletek és eredmények alapján kiértékeljük a program hatékonyságát és tanulási előnyeit a rendezési algoritmusok tanításában.

A rendezési algoritmusok és a vizuális tanítás kombinációja kiemelkedő jelentőséggel bír az informatika és a programozás oktatásában. Reméljük, hogy a SortRace szoftver segítségével a diákok könnyedén megérthetik és elsajátíthatják ezeket az algoritmusokat, és a jövőben hatékonyabb programokat és alkalmazásokat fejleszthetnek.

2. fejezet

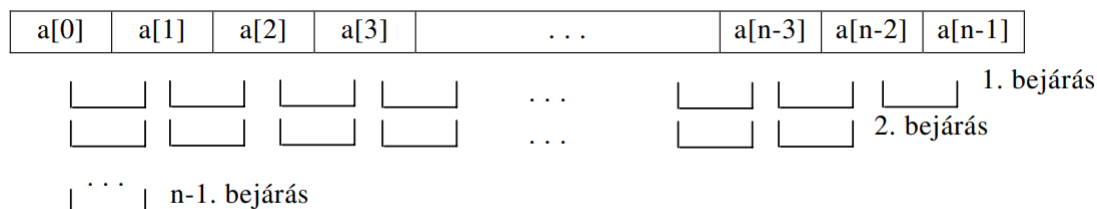
Rendezési algoritmusok bemutatása

[1, 2]

Ebben a fejezetben a rendezési algoritmusokat fogom bemutatni, amelyek az adatok rendezésére és sorba rendezésére szolgálnak. A rendezési algoritmusok olyan speciális eljárások, amelyek segítségével a számítógép rendezi az adatokat valamilyen szabály szerint. Ezeket a rendezési algoritmusokat a következő tulajdonságok alapján fogom vizsgálni: implementáció egyszerűsége, időkomplexitás, legjobb és legrosszabb eset.

2.1. Buborékos rendezés (Bubble Sort)

A buborékos rendezés stratégiáját – első megközelítésben – az alábbi ábra mutatja be.



És hány értékadást? Ez attól függ, hány cserére van szükség. A legrosszabb esetben, ha a számsorozat eredetileg csökkenő sorrendben van, minden összehasonlítást csere követ, ami viszont három értékadást feltételez. Ez összesen $3 * n * (n - 1) / 2$ értékadást jelent.

Ha alapl műveletnek az értékadást és az összehasonlítást választjuk, akkor a buborékos rendezés legrosszabb esetben $4 * n * (n - 1) / 2$ alapl műveletet feltételez. Mivel ez a szám a bemenet méretétől (n) négyzetesen függ, azt mondjuk, hogy ezen algoritmus bonyolultsága (komplexitása) $O(n^2)$. Nem nehéz átlátni, hogy jelenlegi állapotában, legjobb esetben is – ha a számsorozat már eredetileg növekvő sorrendben van – $O(n^2)$ az algoritmus bonyolultsága (bár nem kerül sor cserére, az összehasonlítások száma ugyanannyi marad).

Hogyan lehet javítani az algoritmus bonyolultságán? Vegyük észre, hogy a fenti algoritmus „vak”, abban az értelemben, hogy nem veszi észre, ha a számsorozat már rendezve van. Hogyan lehetne „kinyitni a szemét”? Figyeljük, hogy történik-e csere egy bejárás alatt. Ha nem, akkor a számsorozat rendezett. Sőt a tömb egyszeri bejárása alkalmával szerencsés esetben több elem is a helyére kerülhet. Ha megjegyezzük az utolsó csere helyét, a következő lépésben elég csak addig menni.

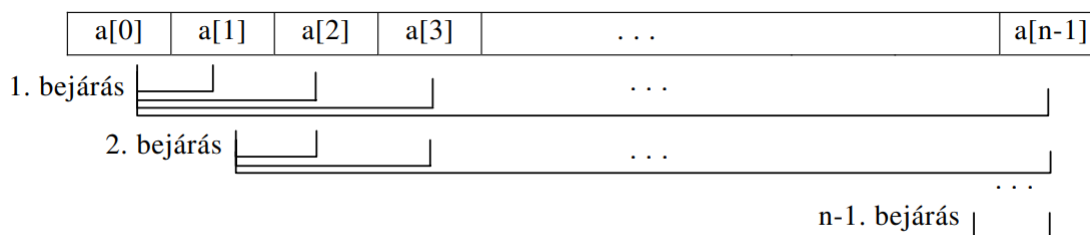
A javított algoritmus legrosszabb esetben továbbra is $O(n^2)$ bonyolultságú, viszont legjobb esetben csak $O(n)$, hiszen az első bejárás után „észreveszi”, hogy a számsorozat rendezett.

```
int a[100], n, u, i, j, v, ucs;
scanf("%d", &n);
for(i = 0; i < n; i++)
    scanf("%d", &a[i]);
u = n - 1;
do
{
    ucs = 0;
    for(j = 0; j < u; j++)
        if(a[j] > a[j + 1])
        {
            v = a[j];
            a[j] = a[j + 1];
            a[j + 1] = v;
            ucs = j;
        }
    u = ucs;
}
while(ucs);
for(i = 0; i < n; i++)
    printf("%d\n", a[i]);
```

2.1. kódrészlet. C példakód buborékos rendezésre.

2.2. Minimum-kiválasztásos rendezés (Minimum Sort)

A minimum-kiválasztásos rendezés bármilyen számsorozat esetén kötelezően végig kell hogy szaladjon $(n-1)$ -szer a számsorozat bizonyos szakaszain. Stratégiája abban áll, hogy az i -edik bejárásban az $a[i..n-1]$ ($i=0..n-2$) szakasz legkisebb elemét előrehozza az i -edik pozícióba. Ezt úgy valósítja meg, hogy $a[i]$ -t sorra összehasonlítja az $a[j]$ ($j=i+1..n-1$) elemekkel, és valahányszor kisebbet talál, felcseréli őket. Ezt mutatja be szemléletesen az alábbi ábra:



2.2. ábra. Minimum-kiválasztásos rendezés

Az alábbi programrészlet a minimum-kiválasztásos rendezés implementációja:

```
...  
for(i=0;i<n-1;i++)  
    for(j=i+1;j<n;j++)  
        if(a[i]>a[j]) {v=a[i];a[i]=a[j];a[j]=v;}  
...
```

2.2. kódrészlet. C példakód minimum-kiválasztásos rendezésre.

Bár megírni egyszerűbb, mint a buborékos rendezést, kevésbé hatékony, ugyanis képtelen észrevenni azt az esetet, amikor a számsorozat rendezetté vált, vagy már eredetileg rendezett. Ez a magyarázata annak, hogy bonyolultsága $O(n^2)$ minden esetben.

2.3. Beszúró rendezés (Insertion Sort)

A beszúró rendezés egy egyszerű és intuitív rendezési algoritmus, amely az új elemeket egy rendezett sorozatba helyezi be. Az algoritmus lépéseiben az egyes elemeket összehasonlítjuk a már rendezett sorozat elemeivel, majd a megfelelő helyre szúrjuk be.

Az algoritmus kezdetben egy üres rendezett sorozattal indul, amelyet fokozatosan fogunk feltölteni. Az elemeket egyesével kiválasztjuk a rendezendő adatsorból. A kiválasztott elemet összehasonlítjuk a rendezett sorozat elemeivel a legjobb hely megtalálása érdekében. Az elemet beillesztjük a megfelelő helyre a rendezett sorozatban, és eltoljuk a nagyobb elemeket, hogy helyet adjunk az új elemnek. Ezek a lépések ismétlődnek, amíg minden elemet beillesztettünk a rendezett sorozatba.

A legjobb esetben az algoritmus akkor teljesít a leghatékonyabban, amikor a bemeneti adatsor már rendezett vagy közel rendezett állapotban van. Ebben az esetben az elemek nagy részét nem kell eltolni, és a beszúró műveletek száma minimális. A beszúró rendezés legjobb esetben $O(n)$ időkomplexitással rendelkezik, ahol n az elemek száma.

A legrosszabb esetben az algoritmus akkor teljesít a leglassabban, amikor a bemeneti adatsor fordított sorrendben van rendezve. Ebben az esetben az összes elemet el kell tolni a rendezett sorozatban, és a beszúró műveletek száma a maximális. A beszúró rendezés legrosszabb esetben $O(n^2)$ időkomplexitással rendelkezik, ahol n az elemek száma.

Fontos megjegyezni, hogy a beszúró rendezés átlagos esetben is $O(n^2)$ időkomplexitással rendelkezik. Az algoritmus hatékonysága azonban javítható, például az elemek cseréjével helyettük az adatsorban vagy az optimálisabb beszúró hely megtalálásával.

Ezek az algoritmusok, amiket eddig felsoroltam, manapság már csak didaktikai célból használnak, nagyon jól fejleszti az algoritmikai gondolkodását egy diáknak, de valós rendezési algoritmusokként nem nagyon használják őket.

Az alábbi programrészlet a beszúró rendezés implementációja:

```
int i, key, j;
for (i = 1; i < n; i++) {
    key = arr[i];
    j = i - 1;

    while (j >= 0 && arr[j] > key) {
        arr[j + 1] = arr[j];
        j = j - 1;
    }
    arr[j + 1] = key;
}
```

2.3. kódrészlet. C példakód beszúró rendezésre.

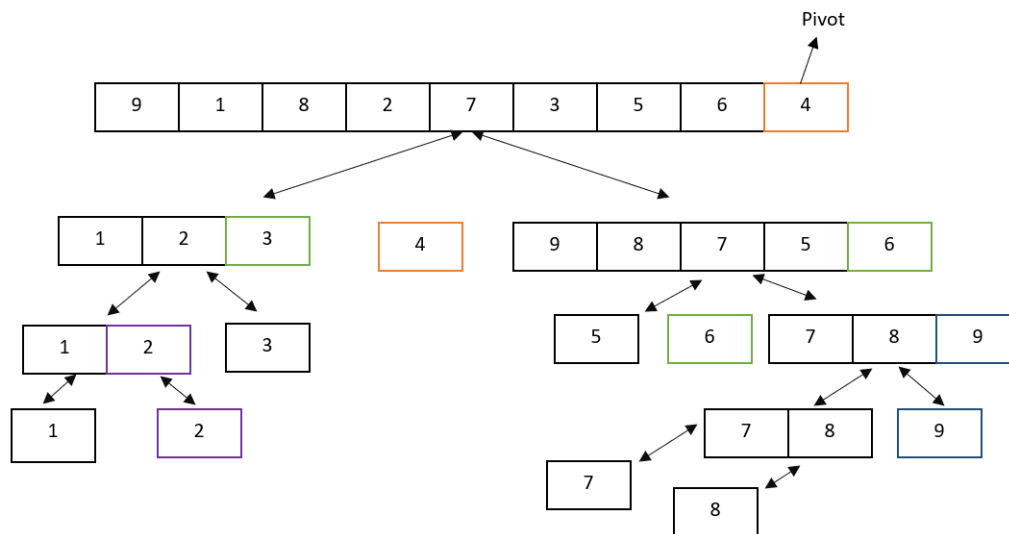
2.4. Gyorsrendezés (Quick Sort)

A gyorsrendezés algoritmus egy nagyon elterjedt megoldás. Ez egy hatékony és gyors rendezési algoritmus, amely osztályozza és helyezi el az elemeket a pivot elemhez viszonyítva. A problémát részproblémákra bontja, majd rekurzívan rendez minden részproblémát. Ezt az algoritmust a rekurzív programozás bevezetésénél tanítják. Ez a stratégia megértése egy mérföldkő lehet egy programozó életében.

A QuickSort működési elve abból áll, hogy válasszunk egy tetszőleges elemet az adatsorból, amit pivot elemnek nevezünk. Az adatsor elemeit osztjuk két részre a pivot elem alapján: a kisebb elemeket rakjuk balra, a nagyobb elemeket pedig jobbra. A 2.4. kódrészletben ezért a partition függvény felel. Rekurzívan alkalmazzuk a QuickSort algoritmust mindkét részre, amíg a rendezett részsorozatok mérete 1 vagy kisebb nem lesznek. A rendezett részsorozatokat összeillesztjük, és így megkapjuk a teljesen rendezett adatsort.

Az alábbi programrészlet a Gyorsrendezés implementációja:

```
void swap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
```



2.3. ábra. Gyorsrendezés

```
int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = (low - 1);
    for (int j = low; j <= high - 1; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}

void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
```

2.4. kódrészlet. C példakód Gyorsrendezésre.

A legjobb esetben az algoritmus akkor teljesít a leghatékonyabban, amikor minden pivot elem az adatsor közepén van, és az adatok szimmetrikusan oszlanak meg a pivottól balra és jobbra. Ebben az esetben az algoritmus $O(n \log n)$ időkomplexitással rendelkezik, ahol n az elemek száma.

A legrosszabb esetben az algoritmus akkor teljesít a leglassabban, amikor a pivot elem mindig a legkisebb vagy a legnagyobb elem. Ebben az esetben az algoritmus $O(n^2)$

időkomplexitással rendelkeznek. Ez akkor fordulhat elő, ha az adatsor már rendezett vagy közel rendezett állapotban van, és mindig az első vagy az utolsó elemet választjuk pivotnak.

Fontos megjegyezni, hogy átlagos esetben a QuickSort gyors és hatékony. Az átlagos esetben az algoritmus általában $O(n \log n)$ időkomplexitással rendelkezik, és jó teljesítményt nyújt nagy adatsorok rendezésére is.

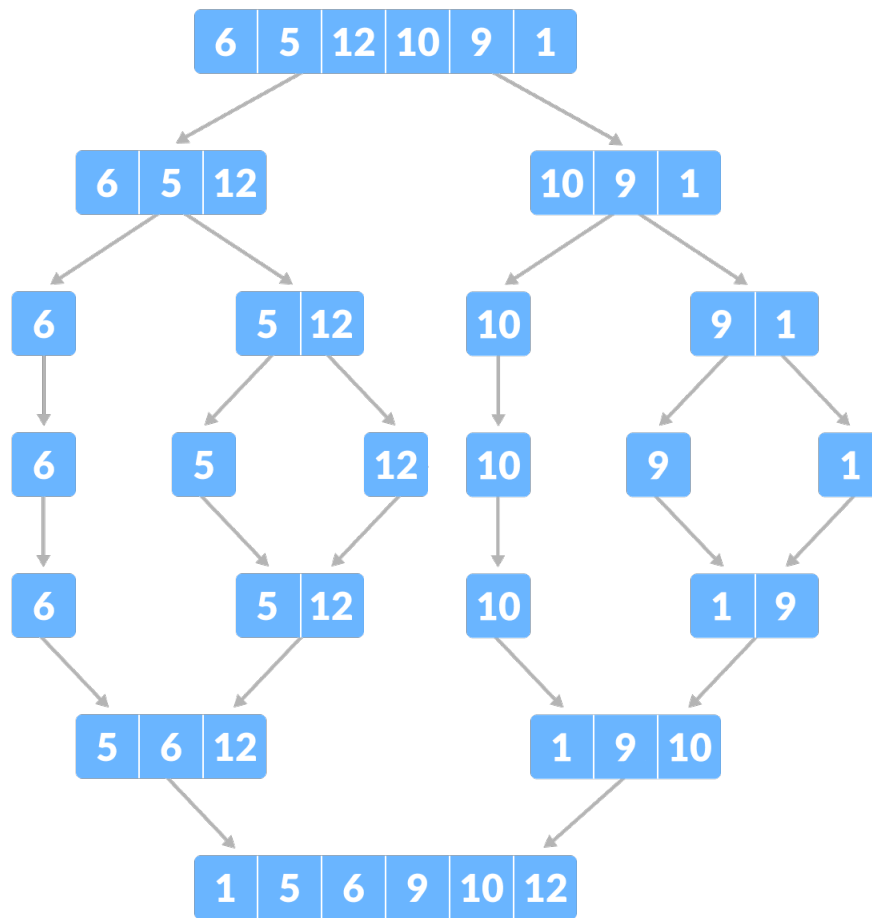
A QuickSort algoritmus gyakran előnyös, mert in-place működik (nincs szükség további memóriára), és viszonylag kevés extra műveletet igényel. Azonban figyelembe kell venni a legrosszabb eset időkomplexitását és a pivot elem választásának stratégiáját a hatékonyság optimalizálása érdekében.

2.5. Összefésüléses rendezés (Merge Sort)

Ez a hatékony rendezési algoritmus a "oszd meg és uralkodj" elven alapul. Az adatsort részekre osztja, majd ezeket a részeket rendezetten összefésüli. Az összefésüléses rendezés mind az átlagos, mind a legrosszabb esetben elég jó időkomplexitással rendelkezik. Az "divide et impera", vagyis "oszd meg és uralkodj" úgyszintén egy algoritmikai stratégia amit jól meg lehet érteni az összefésüléses rendezés algoritmus tanulmányozásával.

Az összefésülő rendezés működési elve a következő lépésekből áll. Az adatsort két részre osztjuk egy középső pont alapján. Rekurzívan alkalmazzuk az összefésülő rendezést mindkét részre, amíg a rendezendő részsorozatok mérete 1 vagy kisebb nem lesznek. A rendezett részsorozatokat összefésüljük, úgy hogy összehasonlítjuk a következő elemeket a részsorozatokban, és a kisebb elemeket egy új rendezett sorozatba helyezzük. A rendezett részsorozatok összefésülése után megkapjuk a teljesen rendezett adatsort.

```
void merge(int arr[], int left[], int right[], int leftSize, int rightSize) {
    int i = 0, j = 0, k = 0;
    while (i < leftSize && j < rightSize) {
        if (left[i] <= right[j]) {
            arr[k] = left[i];
            i++;
        } else {
            arr[k] = right[j];
            j++;
        }
        k++;
    }
    while (i < leftSize) {
        arr[k] = left[i];
        i++;
        k++;
    }
    while (j < rightSize) {
        arr[k] = right[j];
        j++;
        k++;
    }
}
```



2.4. ábra. Összefésüléses rendezés

```

}

void mergeSort(int arr[], int n) {
    if (n <= 1) {
        return;
    }
    int mid = n / 2;
    int left[mid];
    int right[n - mid];
    for (int i = 0; i < mid; i++) {
        left[i] = arr[i];
    }
    for (int i = mid; i < n; i++) {
        right[i - mid] = arr[i];
    }
    mergeSort(left, mid);
    mergeSort(right, n - mid);
    merge(arr, left, right, mid, n - mid);
}

```

2.5. kódrészlet. C példakód Összefésüléses rendezésre.

A fenti módszer legjobb esetben akkor teljesít a leghatékonyabban, amikor az adatsor már rendezett vagy közel rendezett állapotban van. Ebben az esetben az összefésülő rendezés $O(n \log n)$ időkomplexitással rendelkezik, ahol n az elemek száma. Az összefésülő rendezésnél a részsorozatok nem kell tovább osztani, mivel már rendezettek, így az összefésülés műveletének ideje dominál.

A legrosszabb esetben az összefésülő rendezés akkor teljesít a leglassabban, amikor az adatsorban a részsorozatok mindig azonos méretűre osztjuk. Ebben az esetben az összefésülő rendezés $O(n \log n)$ időkomplexitással rendelkezik. Az összefésülő rendezésnél a rekurzív osztások miatt az összefésülés műveletek száma növekszik, ami hatással lehet a teljesített műveletek számára és a memóriahasználatra. Fontos megjegyezni, hogy az összefésülő rendezés az átlagos esetben is hatékony és általában jobb teljesítményt nyújt, mint az egyszerűbb rendezési algoritmusok. Az összefésülő rendezés azonban további memóriát igényel, mivel az adatsorokat külön tárolja a rendezett sorozat összefésülése előtt. Ezért a memóriahasználat szempontjából fontos lehet a hatékonyság optimalizálása során.

3. fejezet

A projekt célja

3.1. Rendezési algoritmusok vizuális módon történő tanítása

A dolgozat célja, hogy bemutassa és vizuálisan tanítsa a különböző rendezési algoritmusokat a SortRace nevű általam fejlesztett szoftver segítségével. A SortRace egy interaktív és intuitív program, amely lehetőséget nyújt a felhasználóknak a rendezési algoritmusok valós idejű megismerésére és gyakorlására.

Az általam fejlesztett szoftvernek a célja, hogy a segítségével a felhasználók gyakorolhassák a rendezési algoritmusok működését vizuálisan, közvetlenül részt vehessenek a rendezési folyamatban és megértsék az algoritmusok lépéseit valós időben.

A SortRace által kínált vizuális tanítási lehetőség lehetővé kell tennie a felhasználók számára, hogy láthatóan kövessék nyomon a rendezési algoritmusok működését. A program megjeleníti az adatokat, amelyeket rendezni kell, és a felhasználók lépésről lépésre nyomon követhetik az algoritmusok által végrehajtott műveleteket, ez segít abban, hogy kialakuljon a rendezési algoritmusokkal kapcsolatos szemlélet és logika a diákok és más felhasználók fejében.

3.2. Versenykörnyezetben történő tanulás

A SortRace-nek emellett célja az is, hogy egy szórakoztató játékelményt is nyújtson a felhasználóknak a játékos módon történő tanulás által, ami lehetővé teszi, hogy a felhasználók versenyezzenek egymással a rendezési feladatok elvégzésében, a rendezési algoritmusok gyors és pontos alkalmazásában, ennek a versenynek az eredményeinek nyomon követésére és összehasonlítására más felhasználók eredményeivel, ami inspiráció lehet a fejlődésre.

A projekt célja, hogy a felhasználók saját tempójukban tudjanak haladni a rendezési algoritmusok elsajátításával és gyakorlásával, illetve az, hogy a kezdők és haladók egyaránt megtalálhatják a számukra megfelelő kihívást.

Céлом az, hogy a versenyzők játékos módon sajátítsák el és értsék meg ezeket a rendezési algoritmusokat, valamint gyorsan és hatékonyan alkalmazzák azokat. A versenyzőknek lehetőségük van gyakorolni, tökéletesíteni a rendezési algoritmusok alkalmazását a gyorsaság és a pontosság terén. A játékos módszer segít abban, hogy a felhasználók aktívan részt vegyenek a folyamatban, és élvezettel tanuljanak.

A versenyzők a gyakorlat során fokozatosan fejleszthetik képességeiket a rendezési algoritmusok alkalmazásában, és fontos, hogy ne csak mechanikusan alkalmazzák az algoritmusokat, hanem megértsék azoknak a működését is.

Az ilyen játékos tanulási módszerek hatékonyak lehetnek a versenyzők motiválásában és az algoritmusok gyors elsajátításában. Emellett a szemlélet és logika kialakulása a fejekben lehetővé teszi, hogy a versenyzők rugalmasan alkalmazzák a rendezési algoritmusokat más problémák megoldására is, és általánosabb számítási készségeket szerezzenek.

4. fejezet

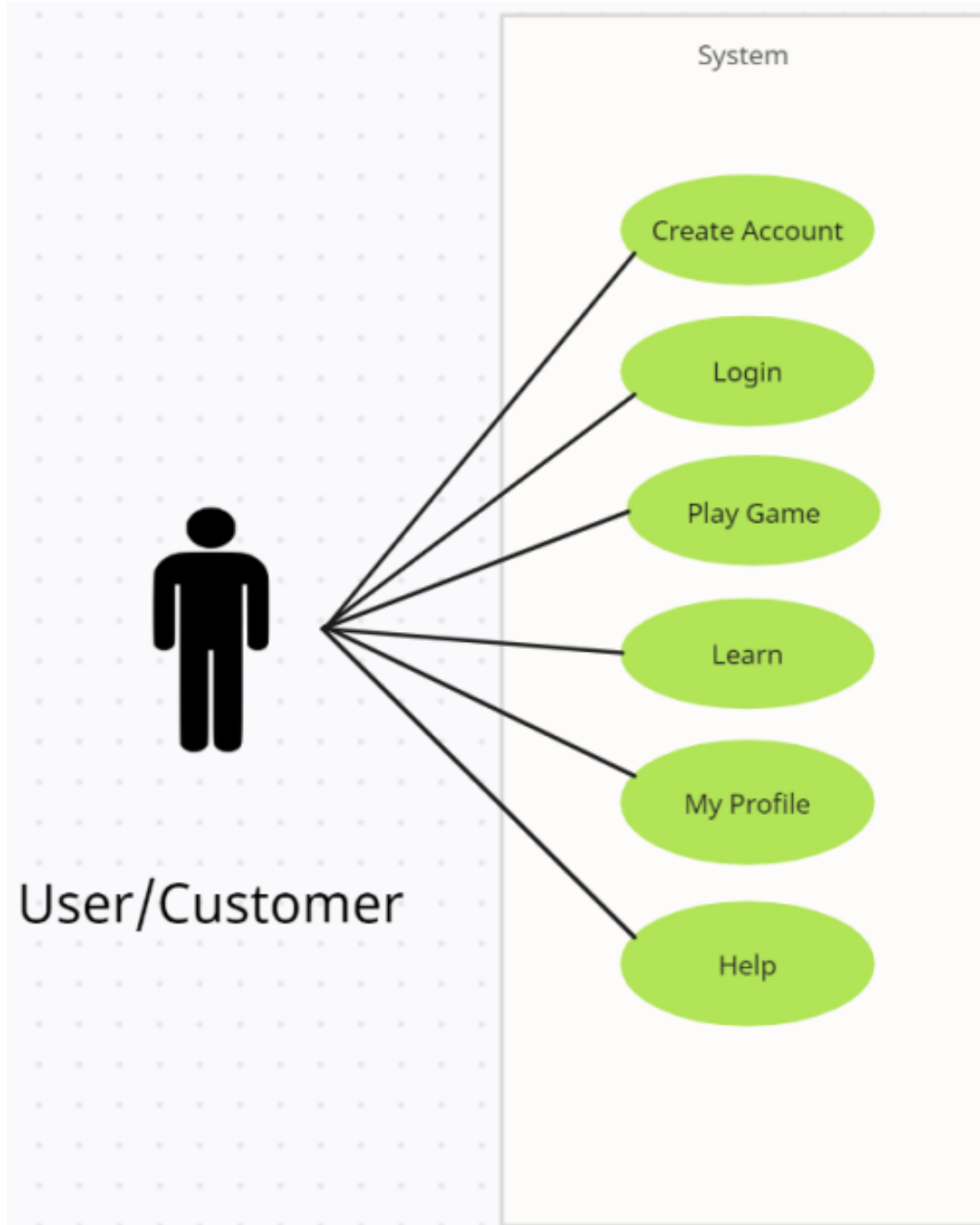
Követelményspecifikációk

4.1. Felhasználói követelmények

A játéktervezésnek több szempontot kell figyelembe vennie annak érdekében, hogy érdekes és szórakoztató legyen a felhasználók számára, és hatékonyan segítse a rendezési algoritmusok elsajátítását. Az alábbiakban részletesebben kifejtem ezeket a szempontokat:

- A játéknak érdekesnek és szórakoztatónak kell lennie a felhasználó számára. A játéktervezésnek olyan elemeket kell tartalmaznia, amelyek felkeltik a felhasználó érdeklődését és motivációját.
- A játéknak megfelelő és részletes bemutatást kell nyújtania a rendezési algoritmusokról. Ez magában foglalhatja az algoritmusok működésének lépéseit, példákat vagy gyakorlati feladatokat, amelyeken keresztül a felhasználók gyakorolhatják az algoritmusok alkalmazását. Fontos, hogy a játék interaktív legyen, és lehetőséget adjon a felhasználóknak arra, hogy maguk próbálják ki az algoritmusokat és tapasztalják meg az eredményeket.
- A játék felülete egyszerű és intuitív kell legyen, hogy könnyen áttekintést nyújtson a rendezési algoritmusokról és a játékmenetről. A fontos információknak könnyen elérhetők, érthetők kell legyenek. A navigáció, az interakció felhasználóbarát kell legyen, hogy a felhasználók könnyen át tudjanak váltani az egyes részek között, illetve hogy szabadon haladhassanak a tananyagban.
- A játéknak lehetőséget kell adjon a felhasználóknak arra, hogy a saját tempójukban tanuljanak és gyakoroljanak, ami magában foglalhatja a tananyag moduláris felépítését, ahol a felhasználók választhatnak a rendezési algoritmusok között és haladhatnak a saját preferenciáik szerint. Emellett a játék lehetőséget adhat a felhasználóknak a gyakorlásra vagy ismétlésre, ha úgy érzik, hogy szükségük van rá.
- Lehetőséget kell nyújtson az eredmények nyomon követésére és azok összehasonlítására más felhasználók eredményeivel, mivel ez inspiráló lehet a versenyzéshez és a fejlődés nyomon követéséhez. A játék lehetőséget adhat arra is, hogy a felhasználók megosszák az eredményeiket vagy tapasztalataikat más felhasználókkal, és interakcióba lépjenek egymással.

- A játék lehetővé kell tennie a felhasználók számára, hogy a nehézségi szintek közötti váltásának. Kezdők számára elérhetőek lehetnek alapvető rendezési algoritmusok, míg haladóknak vagy tapasztaltabb felhasználóknak kihívást jelentő algoritmusokat vagy komplexebb feladatokat kínálhat a játék.



4.1. ábra. Use Case diagram

4.2. Rendszerkövetelmények

4.2.1. Funkcionális rendszerkövetelmények

A felhasználó az alkalmazás indításakor a főoldalon találja magát, ahol lehetősége van regisztrálni vagy bejelentkezni, attól függően, hogy korábban már regisztrált-e vagy sem. A regisztráció során a felhasználó megadja a szükséges információkat, például az email címet és jelszót. Ha már rendelkezik fiókkal, akkor egyszerűen be tud jelentkezni az adott email címen és jelszaván keresztül.

Bejelentkezés után a felhasználó számára elérhetővé válik a "Play" gomb, amelyet megnyomva képes kiadni a választást a lehetséges ellenfelek számát illetően. Ez lehetőséget ad neki arra, hogy kiválassza, hány ellenfelet szeretne keresni a játékhoz, ezután a felhasználó a "Keresés" gombra kattintva elindíthatja az ellenfelek keresését a rendszerben.

Ha a rendszer talál megfelelő partnereket a játékhoz, akkor a felhasználó a "Ready" gombra kattintva elindíthatja a játékot. A játék során a felhasználónak két kártyát kell kiválasztania a rendelkezésre álló kártya-keverékből. Ezután a rendszer meghatározza, hogy melyik kártya rendelkezik a nagyobb értékkel a mögöttük álló számsorozat alapján. A kártyák nagysága alapján rendezni lehet a kártyák mögött álló számsorozatot.

Ezekkel a funkciókkal a játék lehetővé teszi a regisztrált felhasználók számára a multiplayer játékot, itt az ellenfeleket véletlenszerűen választja ki a rendszer. A kártyaválasztás és a rendezés folyamata segíti a felhasználókat a rendezési algoritmusok megértésében és gyakorlásában.

4.2.2. Nem funkcionális rendszerkövetelmények

Az alkalmazás futtatásához szükség van egy olyan eszközre, amely képes böngészőt futtatni és kapcsolódni az internethez. Ez lehet például egy számítógép, laptop, okostelefon vagy táblagép. Az eszközön kell lennie egy olyan böngészőalkalmazásnak, amely támogatja a weboldalak megjelenítését és interakciót a felhasználóval. A böngésző segítségével a felhasználó elérheti az alkalmazás weboldalát.

Az interakció a böngészőben történhet billentyűzet vagy egér segítségével. Például a felhasználó kattintásokkal tudja megnyomni a gombokat vagy választani a kártyákat. A billentyűzetet használva a felhasználó lehetőséget kap az alkalmazásban való navigálásra, a szöveges mezők kitöltésére vagy más funkciók elérésére.

5. fejezet

Technológiák bemutatása

5.1. Frontend - Angular

Az Angular-t választottam ennek a webes játéknak és tanító alkalmazásnak a front-endjéhez, mert Angular egy hatékony keretrendszer, amely lehetővé teszi a gyors és hatékony webalkalmazások fejlesztését, és az optimalizált kódvégrehajtás és a gyors betöltési idők révén a játék sima és zökkenőmentes élményt nyújt a felhasználóknak.

Az Angular komponens alapú architektúrája lehetővé teszi a kódbázis moduláris felépítését és újrafelhasználható komponensek készítését, ami különösen előnyös egy játék és tanító alkalmazás esetén, ahol különböző részleteket és funkciókat kell kezelni.

Emellett az Angular kifejezetten hatékony módszerekkel rendelkezik a dinamikus adatok megjelenítésére és frissítésére a felhasználói felületen. Ez nagyon hasznos egy olyan alkalmazás esetén, ahol a játékosok valós időben követhetik a rendezési algoritmusok működését.

Az adatkötési mechanizmusai, mint például a kétirányú adatkötés, lehetővé teszik a könnyű adatmanipulációt és a felhasználói interakciók figyelését, amely nagy segítséget nyújt a játékosok által végzett műveletek nyomon követésében és a megfelelő válaszok kezelésében.

Végül, de nem utolsó sorban az Angular rendelkezik egy nagy és aktív fejlesztői közösséggel, valamint részletes dokumentációval és útmutatókkal. Ez nagyban segíti a fejlesztőt abban, hogy támogatást és válaszokat találjon a felmerülő kérdésekre, valamint megoszthassa és tanulhassa mások tapasztalatait.

5.2. Backend - Java SpringBoot

A Java Spring Boot-ot választottam backend keretrendszernek.

A Spring Boot egy megbízható és robusztus keretrendszer, amely kifejezetten alkalmas webalkalmazások fejlesztésére, a kiterjedt tesztelési és hibakeresési eszközkészlete segít a stabilitás és a megbízhatóság biztosításában.

A Spring Boot leegyszerűsíti a fejlesztési folyamatot és lehetővé teszi a gyors és hatékony alkalmazásfejlesztést az előre definiált sablonok, komponensek és modulok által, amiknek segítségével a fejlesztők időt takaríthatnak meg és magas szintű produktivitást érhetnek el.

Erőteljes integrációs képességekkel rendelkezik, amelyek lehetővé teszik az alkalmazások egyszerű integrálását más rendszerekkel és szolgáltatásokkal. Emellett a Spring Boot rendelkezik egy aktív közösséggel, amely támogatást, dokumentációt és gyakorlati példákat nyújt a fejlesztőknek.

Lehetőséget kínál a horizontális és vertikális skálázhatóságra, amely fontos a nagyobb felhasználói terhelés kezeléséhez. A keretrendszer kialakítása lehetővé teszi a hatékony erőforráskezelést és a magas teljesítmény elérését.

Mindemellett a Spring Boot beépített biztonsági mechanizmusokkal rendelkezik, amelyek lehetővé teszik a felhasználói hitelesítést, az autorizációt és más biztonsági funkciók beállítását. Ez kritikus jelentőségű egy webalkalmazás esetében, amely adatokat kezel és felhasználói interakciót biztosít.

5.3. Adatbázis - PostgreSQL

A PostgreSQL adatbázis-kezelő rendszert választottam.

A PostgreSQL egy nyílt forráskódú relációs adatbázis-kezelő rendszer, amit széles körben használnak a fejlesztők és a vállalatok, mert nagy megbízhatóságot és stabilitást nyújt, valamint számos fejlesztői közösséget és dokumentációt tartalmaz, amely támogatást nyújt az alkalmazás fejlesztése során.

A PostgreSQL támogatja a relációs adatmodellt, ami lehetővé teszi az adatok logikai összekapcsolását és szervezését. Emellett támogatja a JSON adattípust is, így lehetőség van a strukturált adatok tárolására és kezelésére, ami rugalmasságot biztosít az adatbázis tervezése során.

Számos haladó funkciót, lehetőséget kínál, mint például a tranzakciókezelés, indexelés, felhasználókezelés, eljárások és tárolt eljárások. Ezek a funkciók lehetővé teszik az adatbázis hatékony kezelését és optimalizálását a Spring Boot alkalmazásban.

A PostgreSQL nagy teljesítményű adatbázis, amely képes hatékonyan kezelni nagy adathalmazokat. Emellett skálázhatóságot biztosít a nagyobb terhelés kezeléséhez, például a replikáció és a terheléelosztás beállításával.

A Spring Boot számos beépített támogatást nyújt a PostgreSQL adatbázishoz. Az alkalmazás konfigurációja és az adatbázis kapcsolat kezelése egyszerű és könnyen konfigurálható a Spring Boot beépített adatbázis-kapcsolódási tulajdonságainak felhasználásával.

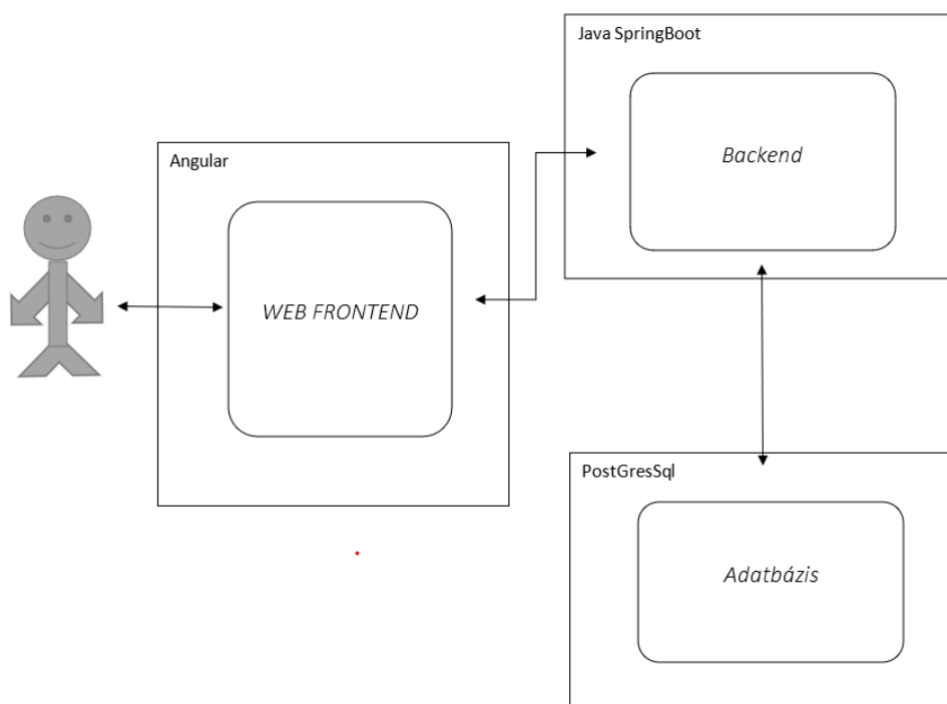
A PostgreSQL használata a Spring Boot backendhez lehetővé teszi egy megbízható, rugalmas és teljesítményorientált adatbázis használatát az alkalmazásban. Az adatbázis könnyen integrálható és konfigurálható a Spring Boot keretrendszerrel, így egy erőteljes és hatékony alkalmazást hozhatunk létre.

6. fejezet

Tervezés

6.1. Architektúra

Ahogy az előző fejezetben említettem a használt technológiákat lássunk most egy architektúrát, hogy hogyan kommunikálnak egymással ezek a komponensek az én projektben.

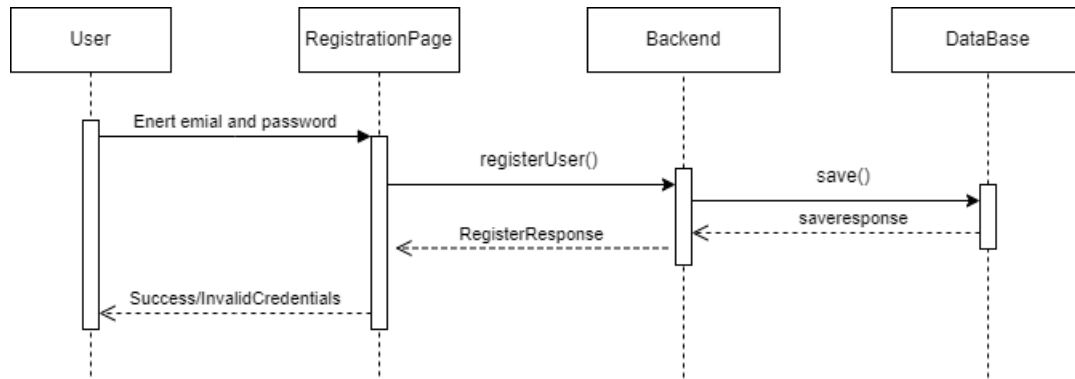


6.1. ábra. A szoftver architektúrája

Látható, hogy a backend központi helyen van, a frontend nem kommunikál az adatbázissal, a backend szolgálja ki a frontendet.

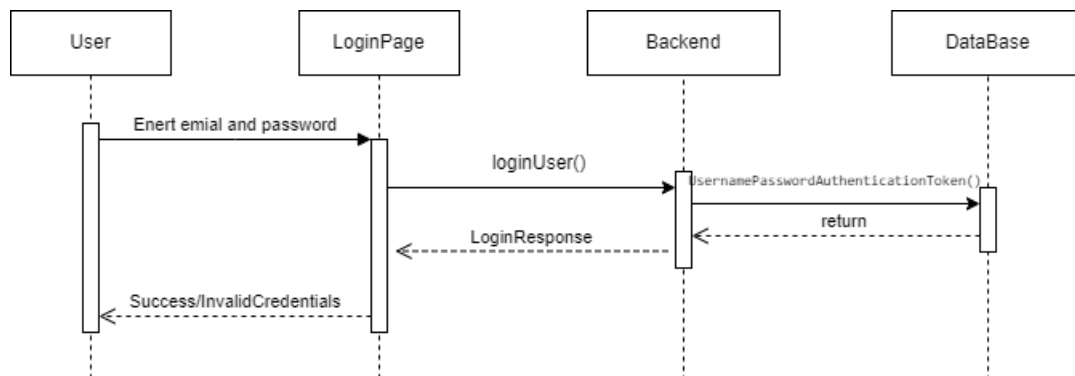
6.2. Szekvenciadiagramok

A regisztrációs funkció teljesen átlagos módon működik. A felhasználó kitölt egy űrlapot, utána a frontend összesíti az adatokat a RegisterUserDTO-ban, a jelszót hashelve rakja be ebbe a struktúrába, ezt elküldi a backendnek, ezután a backend elmenti az adatokat az adatásiban, végül visszaküld egy érvényes token. Enne segítségével tudja a frontend, hogy a játékos be van jelentkezve, és kezdhet új játékot.



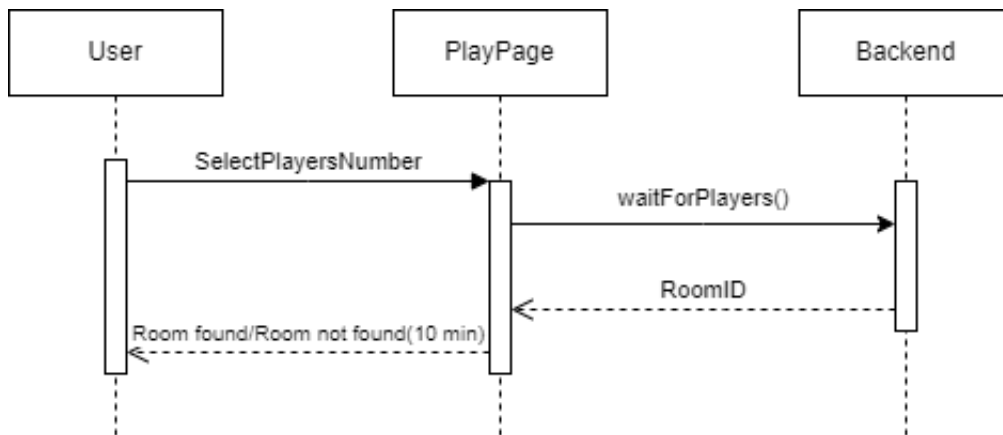
6.2. ábra. Regisztráció

A bejelentkezés nagyon hasonló módon működik a regisztrációhoz, csak ebben az esetben a backend nem új felhasználót hoz létre, hanem ha érvényes adatokat kapott akkor visszatéríti a token.



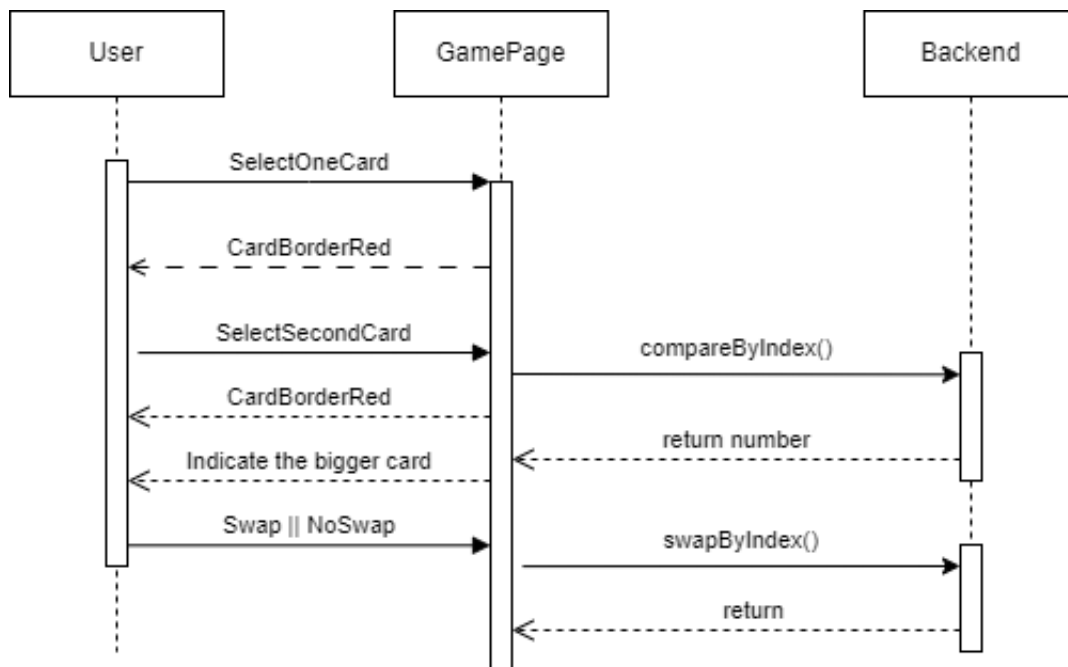
6.3. ábra. Bejelentkezés

Mikor a játékos új játékot akar kezdeni, akkor ki kell válassza, hogy hány ellenféllel akar játszani, elküldi ezt a kérést. A backend keres számára egy létező szobát, amibe belefér, ha nincs ilyen szoba akkor létrehoz egy új szobát. Ezután a backend várakozik. Várja, hogy megteljen a szoba. Ha betelt a szoba akkor ad választ a frontendnek, hogy indulhat a játék.



6.4. ábra. Várkozás a többi játékosra

A játék során a felhasználó kiválaszt két kártyát. Ezeket a kártyákat kiválaszthatja az egér használatával vagy akár 1-10-ig a gombok segítségével. Ha kiválasztotta a két összehasonlítandó kártyát akkor a frontend kérést küld a backendnek. Ebben a post hívásban a CompareRequestDTO van. Ebben a DTO ban két index szerepel, a két kártya pozíciójával. Erre válaszként a két szám különbséget adja vissza, ebből el lehet dönteni, hogy melyik indexen lévő szám a nagyobb. Ekkor jelezzük ezt, hogy melyik a nagyobb érték. A felhasználó dönthet, hogy ki akarja cserélni a két kártyát vagy nem. Ha a cserét választja akkor a backend megcseréli a két számot, és ellenőrzi, hogy a számsor rendezve van. Ha nincs akkor visszaküldi, hogy a csere sikeres volt, ellenben, ha rendezve van akkor visszaküldi az időt. Vagyis, hogy a játék kezdete óta mennyi idő telt el. Ebben az esetben ezt megjelenítjük a frontenden.



6.5. ábra. A játék menete

6.3. Frontend főbb komponensei

[3]

Az Angular projektek általában fel vannak osztva különböző részekre vagy modulokra, hogy jól strukturált és skálázható kódbázist biztosítsanak. Két fő rész, amelyeket használtam, a core és a modules.

A core részben az alkalmazás alapvető működéséhez szükséges központi szolgáltatásokat, osztályokat és segédmetódusokat tartalmazza. Ide tartoznak olyan elemek, mint az autentikáció, az adatkezelés, a globális konfiguráció vagy az interfészek. A core modul célja, hogy a központi üzleti logikát és a globális konfigurációt elkülönítse a többi modultól, így ezek a funkciók könnyen hozzáférhetők és újrafelhasználhatók legyenek a teljes alkalmazásban. A "core" modul lehetőséget ad az alkalmazás inicializálására és konfigurálására is. Például itt lehet definiálni az alkalmazás globális beállításait, mint például az API végpontok URL-je, a nyelvi beállítások, a hibakezelés vagy az alkalmazás szintű gyári osztályok beállításai.

```
@Injectable({ providedIn: 'root' })
export class GameService {
  constructor(private http: HttpClient) {}
  link: string = 'http://localhost:8080/game'
  searchForGame(searchGameDto: SearchGameDto) {
    return this.http.post<Number>(
      this.link+'/searchforgame',
      searchGameDto,
      { observe: 'response' }
    );
  }

  waitForPlayers(id: Number): Observable<any> {
    return this.http.get<any>(this.link+'/waitforplayers/' + id);
  }
  startBubleMethod(id: Number): Observable<any>{
    return this.http.get<any>(this.link+'/startBubleMethod/' + id);
  }
  compareByIndex(compareRequest: CompareRequestDto) {
    return this.http.post<number>(this.link+'/compareByIndex/',
      compareRequest, { observe: 'response'});
  }
  swapByIndex(swapRequest: SwapRequestDto){
    return this.http.post<any>(this.link+'/swapByIndex/', swapRequest, {
      observe: 'response'});
  }
  removePlayer(): Observable<any>{
    return this.http.get<any>(this.link+'/removeplayer/');
  }
}
```

6.1. kódrészlet. A coreban található game.service

A fentebb bemutatott osztály kezeli a frontenden az endpointok elérését.

A modul részében az alkalmazás funkcionális részeit reprezentálják. Ezek a modulok gyakran az alkalmazás különböző komponenseit, szolgáltatásait, direktíváit és egyéb fájlokat csoportosítják egy adott funkcióhoz, elemhez vagy üzleti folyamathoz kapcsolódóan. A modulok általában önálló egységek, amelyeket különböző részfeladatokhoz lehet hozzáadni vagy eltávolítani, ezáltal tisztán tartva az alkalmazás struktúráját. Az egyes modulokban található komponensek, szolgáltatások és direktívák közötti kapcsolatot adatkötéssel és függőségi injektálással hozzák létre. A modulok lehetővé teszik az alkalmazás számára, hogy modulárisan bővüljön és kezelje a különböző fejlesztői csapatok munkáját. Ez a felosztás segít az alkalmazás könnyebb karbantarthatóságában és kiterjeszthetőségében.

Jelenleg a module részben 3 fő komponens található. Ezek a usermanager, play és bubblesort. Mit egy átlagos angular komponens, ezek is 3 fájlt tartalmaznak. Egy html fájlt, ez jeleníti meg a böngészőben a megfelelő adatokat. Egy css fájlt, ami felel a kinézetért és vizuális effektekért. Végül a typescript fájlt, ami felel az adatok szolgáltatásáért.

A Usermanager komponens azt jeleníti meg, hogy mi történjen a felhasználó be van jelentkezve innen tud tovább navigálni, például a játék vagy a tanulás komponensre.

A play komponensben azok a fájlok vannak amik ahhoz kellenek, hogy a játékos tudja játszani a játékot versenyszerűen időre.

A bubblesort komponensben pedig a buborékos rendezés tanulásához szükséges vizuális elemek találhatóak.

Az Angular keretrendszer előnye, hogy a modulok közötti hierarchiát és függőségi rendszert pontosan definiálhatjuk, így az alkalmazásunk csak azokat a modulokat tölti be, amelyeket valóban használ, ami javítja az alkalmazás teljesítményét és a fejlesztési folyamatot.

6.3.1. Usermanager

A usermanager használ modálokat. Ezek a modálok felugró ablakok amikben a felhasználó ki tudja tölteni az adatokat vagy ki tudja választani, hogy milyen játék módokat akar játszani. Tehát, ahhoz hogy a játékos be tudjon jelentkezni nem visszük át egy másik oldalra, hanem megjelenítünk egy kis ablakot ahol kitudja tölteni az adatait.

```
<!-- Login Area Start -->
<div class="modal fade login-modal" id="login" tabindex="-1" role="dialog"
  aria-labelledby="login" aria-hidden="true">
  <div class="modal-dialog modal-dialog-centered" role="document">
    <div class="modal-content">
      <button type="button" id="closeButton" class="close"
        data-dismiss="modal" aria-label="Close"><span
          aria-hidden="true">&times;</span></button>
      <div class="modal-body">
        <div class="text-center mt-4">
          <h3 style="font-weight: bold; color:white">SORT<span
            style="color:#EC2124">RACE</span></h3>
        </div>
        <div class="header-area">
```

```

        <h4 class="title">Great to have you back!</h4>
        <p class="sub-title">Enter your details below.</p>
    </div>
    <div class="form-area">
        <form>
            <div class="form-group">
                <label for="login-input-email">Email</label>
                <input type="email" class="input-field"
                    id="login-input-email" placeholder="Enter
                    your e-mail" [(ngModel)]="userLoginDto.email"
                    name="email" id="email" required>
            </div>
            <div class="form-group">
                <label
                    for="login-input-password">Password</label>
                <input type="password" class="input-field"
                    id="login-input-password"
                    placeholder="Password"
                    [(ngModel)]="userLoginDto.password"
                    name="password" id="password" required>
            </div>
            <div class="form-group">
                <button class="mybtn1" (click)="login();">Log
                    In</button>
            </div>
        </form>
    </div>
</div>
</div>
</div>
</div>
<!-- Login Area End -->

```

6.2. kódrészlet. Login ablak

6.3.2. Play komponens

A play komponens typescriptje olyan fő függvényekért felel mind például a kártyák kiválasztása. Itt nagyon fontos, hogy minden esetben két kártya legyen kiválasztva abban a pillanatban mikor a felhasználó a csere gombra kattint. Még fontos az is, hogy jól láthatók legyenek a kiválasztott kártyák és később a csere után látható legyen, hogy melyik a nagyobb kártya. Itt a selectOneItem függvény mutatom be mivel ez a legfontosabb függvénye a play komponens typescriptjének.

```

selectOneItem(i:Number){
    this.selected = this.selected + 1;
    if(this.selected == 1){
        this.selectedFirst = i;
    }
}

```



```

}else if(this.selected == 2){
  this.selectedSecond = i;
  this.compared.i = this.selectedFirst;
  this.compared.j = this.selectedSecond;
  this.swaping.i = this.selectedFirst;
  this.swaping.j = this.selectedSecond;
  this.gameService.compareByIndex(this.compared).subscribe({
    next: (response: HttpResponse<number>) => {
      console.log(response.body)
      if(response.body != null && response.body > 0){
        this.green = this.selectedFirst;
      }else{
        this.green = this.selectedSecond;
      }
      this.selected = 0;
      this.selectedFirst = -1;
      this.selectedSecond = -1;
    },
    error: (err) => {
      console.log(err);
    }
  });
}
}

containerActive(i:Number){
  if(this.selectedFirst == i || this.selectedSecond == i){
    return true
  }
  return false;
}

containerActiveGreen(i:Number){
  if(this.green == i){
    return true
  }
  return false;
}
}

```

6.3. kódrészlet. Login ablak

6.3.3. Bubblesort komponens

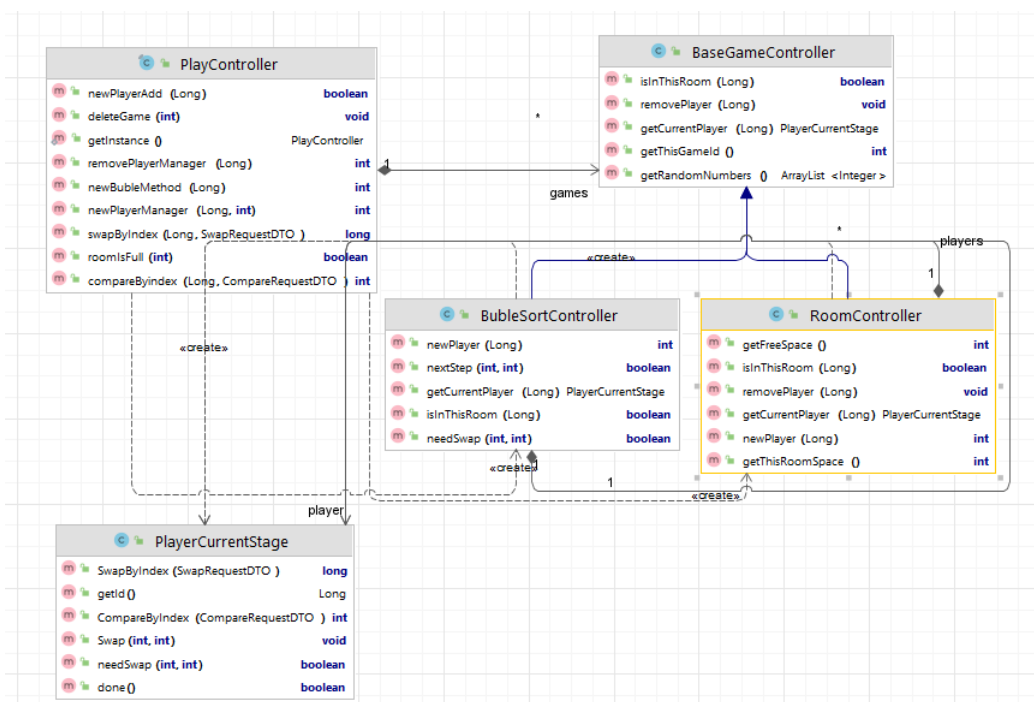
Ez a komponens nagyon hasonlít a play komponensre, mivel a kinézet nagyából ugyan az kell legyen, annyi különbséggel, hogy bizonyos esetekben más endpointokat kell meghívjon a függvény. Itt van egy ellenőrzés rész mielőtt a tanuló kiválaszthatná a következő két kártyát mivel tanulási fázisban nem mindegy, hogy melyik két kártyát válassza ki a felhasználó.

6.4. Backend főbb komponensei

[4]

Ebben a projektben a backend SpringBootban íródott. A projektnek ezen komponense szolgálja ki a frontendet. A regisztrációtól egészen a játék logikáig mindent a backend végez.

6.4.1. Játék logika és az osztályok felépítése



6.6. ábra. Backend főbb osztályai

Mivel ez valós időben történő játék, fontos, hogy egyszerre sok felhasználót szolgáljon ki, de mégis fontos, hogy melyik felhasználót milyen képpen. Vagyis mindegyik játékos a saját számsorozatát kell rendezze. Kezdetben ez szobánként megegyezik. Tehát ha vannak szobák, ahol több játékos van az alap kezdésben mindegyik játékos ugyan azt az összekevert számsort kapja. Ezek a számsorok soha nem jutnak át a frontendre a csalás elkerülése végett. Így szükség van arra, hogy a backend számontartja az összes játékosnak a számsorozatát és a változásokat lementse.

A PlayController osztály egy singleton osztály, ami számontartja az összes jelenleg futó játék menetét. A games atributeuma BaseGameController példányokat tartalmaz. Bármilyen kérés jön a frontendről egy adott felhasználótól ezt a PlayController megkeresi, hogy az adott felhasználó melyik szobában van, és továbbítja a kérést. Nagyon fontos, hogy egy játékos egyszerre csak egy játékban vehet részt. Mondjuk ha elkezd egy rendezési játékot, amíg nem rendezi a számsorozatot nem indíthat új játékot. Ha ezt megpróbálná, mondjuk vissza lép a böngészőben, és új játékot indít, a backend fogja látni, hogy már szerepel egy játékban és oda irányítja vissza a frontendet.

```

public final class PlayController {
    //attribuutumok
    private static PlayController playController;
    private ArrayList<BaseGameController> games = new ArrayList<>();
    //metodusok
    private PlayController() {}
    public static PlayController getInstance(){}
    public boolean roomIsFull(int id)
    public int newPlayerManager(Long id, int roomspace){}
    public int removePlayerManager(Long id){}
    public void deleteGame(int i){ }
    public boolean newPlayerAdd(Long id){}
    public int compareByindex(Long id, CompareRequestDTO compareRequestDTO){}
    public long swapByIndex(Long id, SwapRequestDTO swapRequestDTO){}
    public int newBubleMethod(Long id){}
}

```

6.4. kódrészlet. PlayController osztály attributumai és metódusai

BaseGameController egy ős osztály. Ezt azért láttam fontosnak mert a játék bármikor tovább fejleszthető. Készen áll arra, hogy bármikor új játékmódok jöhetnek létre. Ilyenkor csak kell származtatni egy új osztályt a BaseGameControllerből. Itt elérhetőek az alap metódusok amik az alapjátékmenete megfelelően kiszolgálják. Ezután lehet implementálni azokat a különleges eseteket amiknek az új játékmód kell megfeleljen. A 6.6. ábrán látható, hogy jelenleg két osztály van a származtatva.

```

public class BaseGameController {
    //attribuutumok
    public static int gameId = 0 ;
    private ArrayList<Integer> randomNumbers = new ArrayList<Integer>();
    private int thisGameId;
    //metodusok
    public BaseGameController(){}
    public ArrayList<Integer> getRandomNumbers() {}
    public int getThisGameId() {}
    public boolean isInThisRoom(Long id){ }
    public void removePlayer(Long id){}
    public PlayerCurrentStage getCurrentPlayer(Long ID){}
}

```

6.5. kódrészlet. BaseGameController osztály attributumai és metódusai

A BubleSortController arra hivatott, hogy tanítsa a felhasználót a buborékos rendezésre. Ennek megfelelően olyan metódusokat tartalmaz, amik visszajelzéseket tudnak adni arra, hogy az adott játékos jó sorrendben választja ki a kártyákat vagy nem. Ebben az osztályban példányonként csak egy játékos lehet, mivel a tanulási szakaszban nincs szükség arra, hogy a játékos összemérje tudását más játékosokkal.

```

public class BubleSortController extends BaseGameController {
    //attribuutumok

```

```

private int i=0,j=0,iter=0;
private PlayerCurrentStage player ;
//metodusok
public BubleSortController(){
public int newPlayer(Long id){
public boolean nextStep(int i,int j){
public boolean needSwap(int i, int j){
@Override
public boolean isInThisRoom(Long id){
@Override
public PlayerCurrentStage getCurrentPlayer(Long ID){
}

```

6.6. kódrészlet. BubleSortController osztály attribútumai és metódusai

A másik származtatott osztálya RoomController képes több játékost befogadni. A szoba létrehozásának pillanatában a PlayController megadja, hogy hány játékos kell csatlakozzon ehhez a szobához. Ekkor a szoba legenerálja a véletlenszerű számsorozatot és a játékosokat úgy példányosítja, hogy meg is kapják ezt az alap számsorozatot.

```

public class RoomController extends BaseGameController {
//attribuutumok
private int space;
private ArrayList<PlayerCurrentStage> players = new
    ArrayList<PlayerCurrentStage>();
//metodusok
public int getThisRoomSpace(){
public int newPlayer(Long id){
@Override
public boolean isInThisRoom(Long id){
public int getFreeSpace(){
@Override
public PlayerCurrentStage getCurrentPlayer(Long ID){
@Override
public void removePlayer(Long id){
}

```

6.7. kódrészlet. RoomController osztály attribútumai és metódusai

Az egyik legfontosabb osztály a PlayerCurrentStage. Ennek az osztálynak a példányai felelősek azért, hogy minden játékos megfelelő adatokat kapjon. Mikor arról beszélek, hogy a RoomController létrehoz egy játékost akkor ez a PlayerCurrentStage példányosításával történik. Ha a felhasználó szeretné változtatni a saját számsorozatát akkor ezt a számsorozatot a PlayerCurrentStage példányában változtatja.

```

public class PlayerCurrentStage {
//attribuutumok
private Long id;
private ArrayList<Integer> numbers = new ArrayList<Integer>();
private long start;

```

```

//metodusok
public PlayerCurrentStage(Long id, ArrayList<Integer> randnumbers){}
public int CompareByIndex (CompareRequestDTO compareRequestDTO){}
public boolean needSwap(int i, int j){}
public Long getId(){}
public void Swap(int i,int j){}
public boolean done(){}
public long SwapByIndex(SwapRequestDTO swapRequestDTO){}
}

```

6.8. kódrészlet. RoomController osztály attribútumai és metódusai

6.4.2. Endpointok

A következő endpointokon keresztül éri a frontend a megfelelő funkciókat amiket a backend tud szolgáltatni.

- @POST "user/register"
 - body-ban kap egy registerUserDto object-et, ami tartalmaz egy felhasználónév és jelszó stringet
 - az adatbázisba beírja az új felhasználónak az adatait
- @POST "user/login"
 - body-ban kap egy loginUserDto object-et, ami tartalmaz egy email és egy jelszó stringet
 - az adatbázisból kikeresi a megadott user-t, ha megtalálja válaszként létrehoz egy tokenet és azt adja vissza, másképp 401-es hibakódot ad vissza
- @POST "game/searchforgame"
 - body-ban kap egy CreateRoomDto object-et, ami tartalmaz egy már a felhasználó által kiválasztott szoba méretet
 - az eddigi szobák alapján bele teszi egy már létező szobába a felhasználót, vagy ha nem talál a specifikációknak megfelelő szobát, akkor létrehoz egyet
- @GET "game/waitforplayers"
 - header-ben kap egy Id-t, amely a már előzőleg létrehozott szoba ID-ja, ezzel ellenőrzi, hogy megtelt e a szoba, ezzel lehet várakoztatni a felhasználót a frontend-en
- @POST "game/comparebyindex"
 - body-ban kap egy CompareRequestDto object-et, ami tartalmaz két indexet. A már létező szobához tartozó számsorban a két indexen lévő számot hasonlítja össze, és vissza adja a nagyobbbat
- @POST "game/swapbyindex"
 - body-ban kap egy SwapByIndexRequestDto object-et, ami tartalmaz két indexet. A már létező szobához tartozó számsorban a két indexen lévő számokat kicseréli

-ha a számsor ezután rendezés után sorrendben van akkor az eltelt időt téríti vissza, innen tudja a frontend, hogy a játék lejárt

- @POST "game/startBubbleMethod"

-ezzel az endpointal a játékos létrehoz egy tanuló szobát, amiben a buborékos rendezést tanulhatja

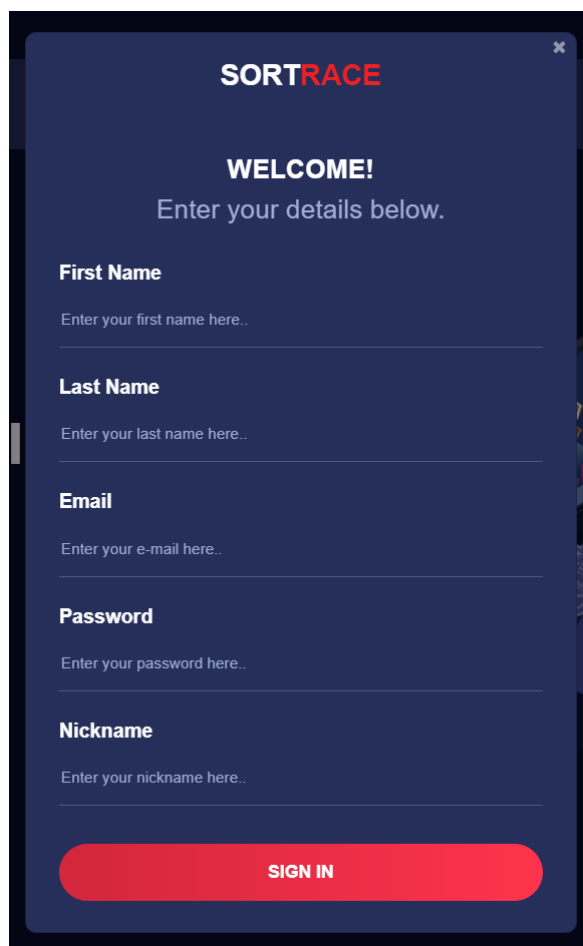
- @POST "game/nextStep"

-ezt az endpointot azok a felhasználók használják aki a buborékos rendezést tanulják, ez az endpoint visszatérít 1-et ha megfelelő lépést választottak, és -1-et ha nem megfelelő pozíciókat választottak

7. fejezet

Alkalmazás felhasználói felületének bemutatása

7.1. Regisztráció



The image shows a registration form for 'SORTRACE'. The form is titled 'WELCOME!' and asks the user to 'Enter your details below.' It contains five input fields: 'First Name', 'Last Name', 'Email', 'Password', and 'Nickname'. Each field has a placeholder text 'Enter your [field name] here..'. At the bottom of the form is a red button labeled 'SIGN IN'.

SORTRACE

WELCOME!
Enter your details below.

First Name
Enter your first name here..

Last Name
Enter your last name here..

Email
Enter your e-mail here..

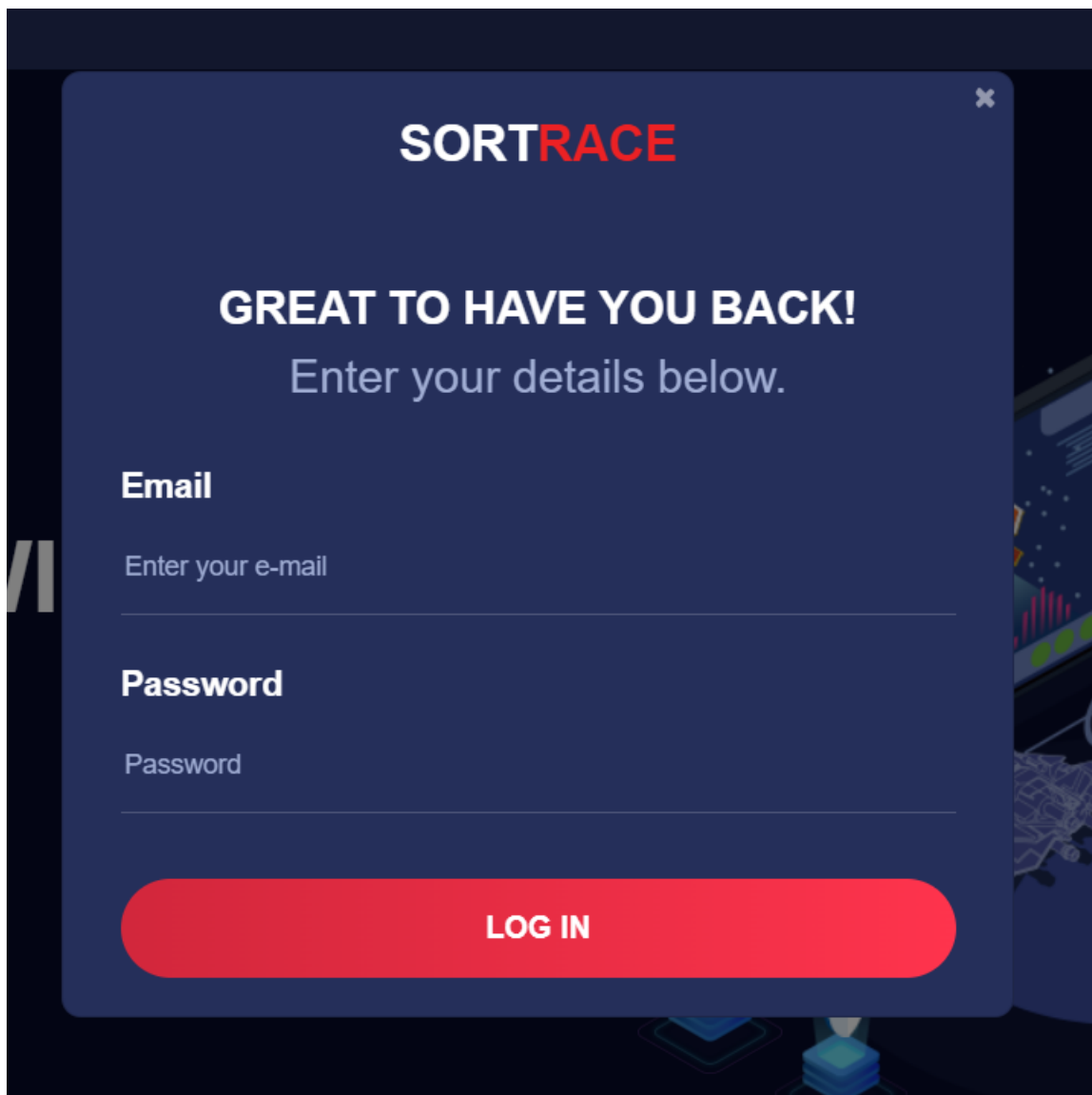
Password
Enter your password here..

Nickname
Enter your nickname here..

SIGN IN

7.1. ábra. Regisztráció

7.2. Bejelentkezés

A dark blue login modal window for SORTRACE. At the top, the SORTRACE logo is displayed in white and red. Below it, the text "GREAT TO HAVE YOU BACK!" is in bold white, followed by "Enter your details below." in a lighter blue. There are two input fields: "Email" with the placeholder "Enter your e-mail" and "Password" with the placeholder "Password". Both fields have white borders. At the bottom, there is a large red button with the text "LOG IN" in white. A small white 'x' icon is in the top right corner of the modal.

SORTRACE

GREAT TO HAVE YOU BACK!
Enter your details below.

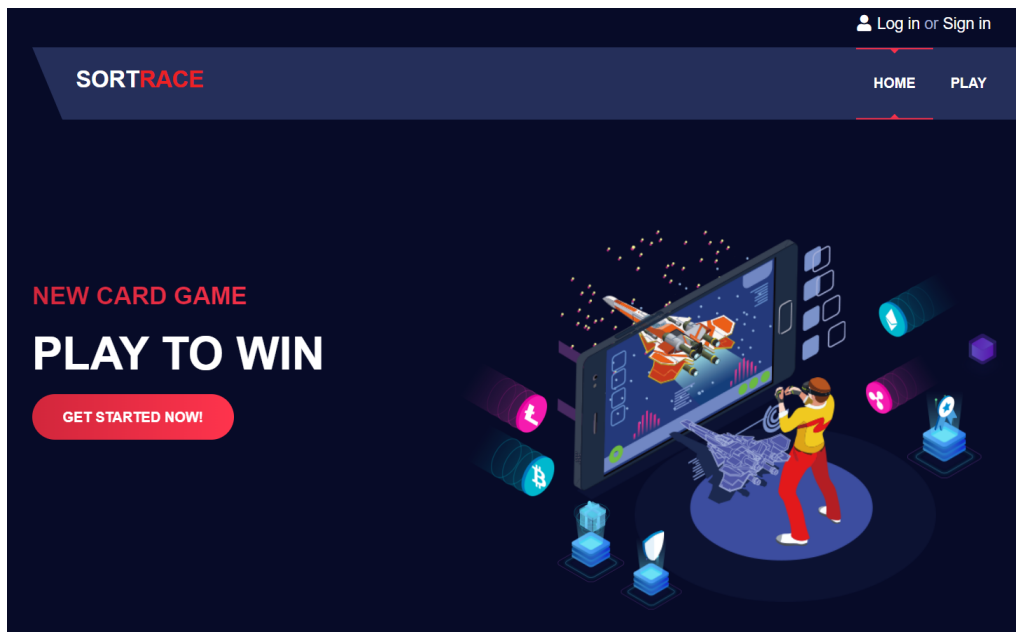
Email
Enter your e-mail

Password
Password

LOG IN

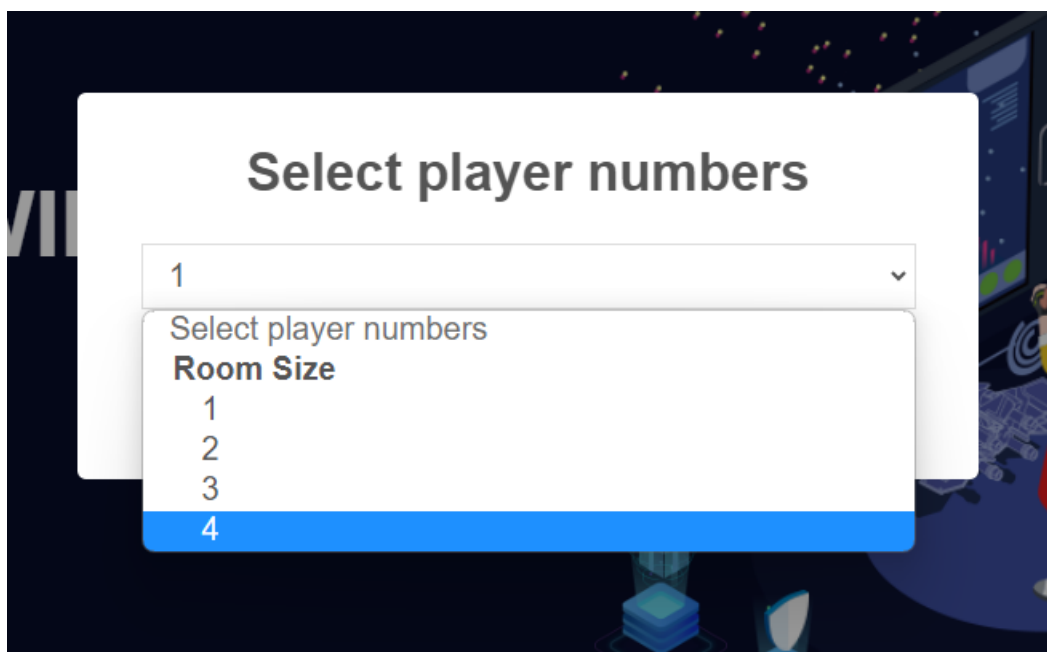
7.2. ábra. Bejelentkezés

7.3. Kezdőoldal



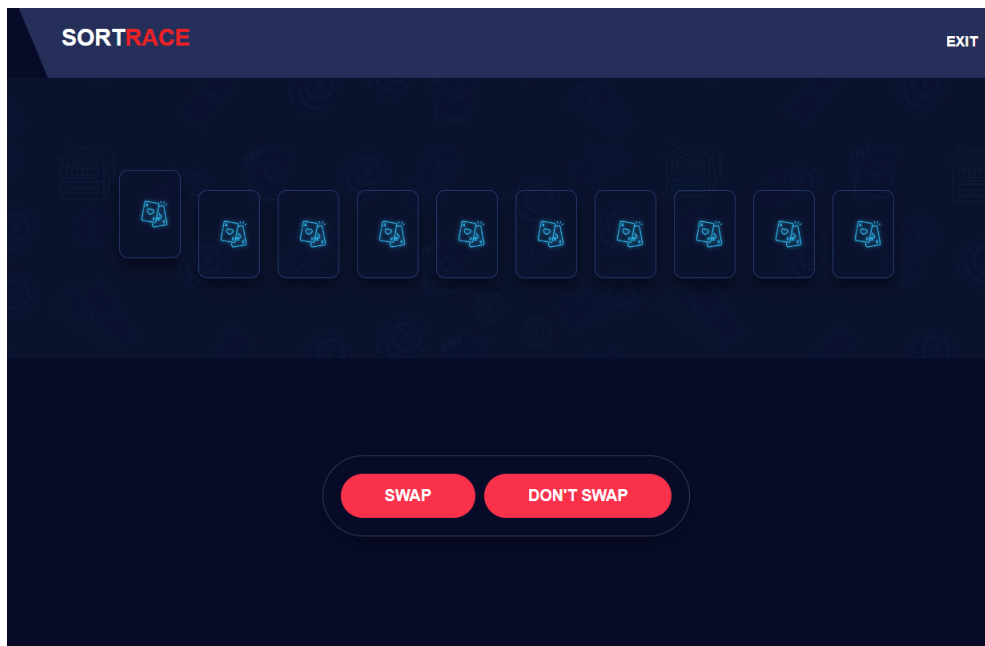
7.3. ábra. Kezdőoldal

7.4. Játék indítása



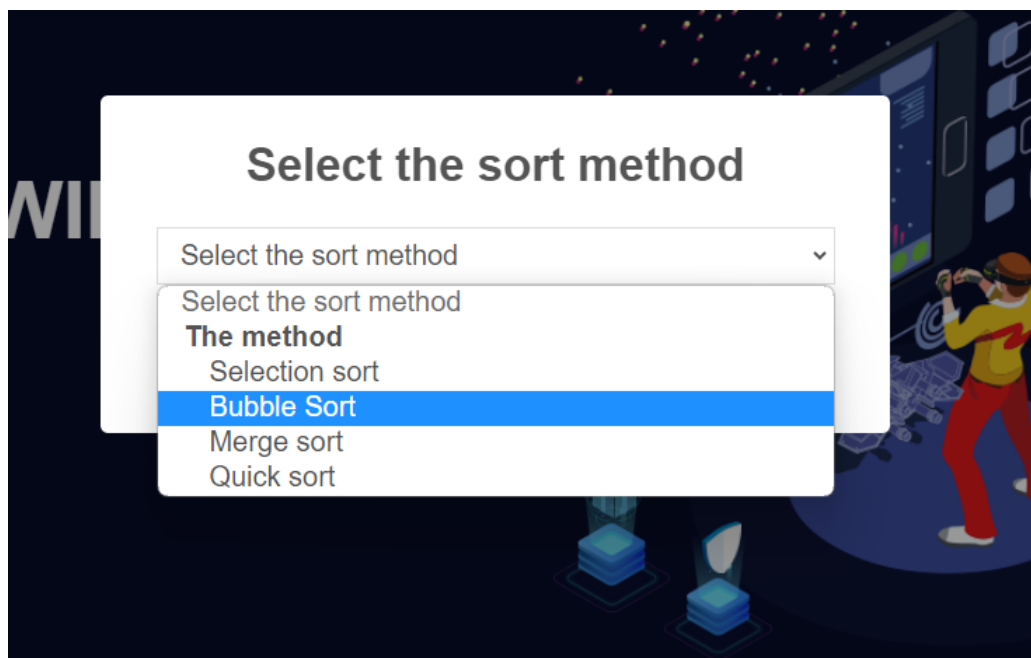
7.4. ábra. Játék indítása

7.5. Játék



7.5. ábra. Játék

7.6. Tanulási játékmód kiválasztása



7.6. ábra. Tanulási játékmód kiválasztása

8. fejezet

További fejlesztési lehetőségek

Egy új játékmód az alkalmazásba sokat fejlesztené a felhasználói élményen, amely hasonló szabályokkal működne, mint az eddigiek. Azonban a legjelentősebb fejlesztési lehetőség a billentyűzet támogatása lenne, amely lehetővé tenné a felhasználók számára, hogy azonnal kiválasszák a kártyákat a megfelelő billentyűkombinációk segítségével. Emellett a kiválasztott kártyákat szintén billentyűkkel azonnal összehasonlíthatják, és a csere is lehetséges lenne egy másik billentyű segítségével. Ez az új funkcionalitás lehetővé tenné a felhasználók számára, hogy versenykörnyezetben használják az alkalmazást, és mindenki azonos esélyekkel induljon a játékban.

Ez a frissítés további versenyképességet adna az alkalmazásnak, és interaktívabbá tudná tenni a játékelményt. A billentyűzet támogatásával a felhasználók gyorsan és hatékonyan választhatnak ki kártyákat, ami növeli a reakcióidőt és az interaktivitást. Ezáltal a játékosok képesek lesznek a rendezési algoritmusokat versenyszerűen alkalmazni, és egyenlő esélyekkel indulni a játék során.

Az új funkcionalitás bevezetése révén az alkalmazás még izgalmasabbá és versenyképesebbé válik, ami a felhasználók számára vonzóbb játékelményt nyújt.

A legfontosabb tovább fejlesztési lehetőség mégis az, ha ez a szoftver közkedvelt, és sok felhasználó használja, vagyis sok adatot szolgáltatnak emiatt. Ezek az adatok tanulmányozásával még optimálisabbá lehet tenni az informatikai módszertanokat.

Összefoglaló

A dolgozat kiindulópontja a rendezési algoritmusok tanulmányozása volt. Ennek során a különböző rendezési módszereket részletesen vizsgálta, elemzést végezve az algoritmusok hatékonyságáról és működéséről. Ez magában foglalta az egyszerűbb rendezési algoritmusok, mint például a buborékrendezés vagy a beszűrő rendezés, valamint az összetettebb módszerek, mint a gyorsrendezés vagy a halomrendezés tanulmányozását is.

A dolgozat második részében a SortRace szoftver került bemutatásra. A SortRace egy interaktív szoftveralkalmazás, amelynek célja segíteni a diákoknak a rendezési algoritmusok megértését és elsajátítását. Az alkalmazás egy webes platformon működik, és két fő komponensre osztható: a frontendre és a backendre.

A frontend az Angular keretrendszert használja, amely lehetővé teszi a felhasználók számára az interaktív tanulást. A frontend rész felelős az alkalmazás felhasználói felületéért, ahol a diákok láthatják a rendezési folyamatok vizualizációját, és interaktívan részt vehetnek a rendezési algoritmusokkal kapcsolatos feladatokban. A vizualizáció segít a diákoknak jobban megérteni a rendezési folyamatot és az algoritmusok működését.

A backend rész a Spring Boot keretrendszert használja, amely felelős az adatkezelésért és a kommunikációért a frontend és a szerver között. Ez biztosítja az alkalmazás működéséhez szükséges logikát, mint például a rendezési algoritmusok implementációit és a felhasználói válaszok feldolgozását. A backend részében tárolódnak az adatok és az eredmények is.

A SortRace lehetővé teszi a diákok számára, hogy tanulási teret kapjanak a rendezési algoritmusok gyakorlásához. A gyorsasági játék keretein belül a felhasználóknak lehetőségük van a rendezési algoritmusok hatékonyságának mérésére és összehasonlítására. Emellett a szoftver könnyen bővíthető újabb rendezési algoritmusokkal vagy további interaktív tanulási lehetőségekkel, amelyek segítenek a diákoknak a további fejlődésben és a témakör mélyebb megértésében.

A dolgozat célja, hogy bemutassa a SortRace szoftver felépítését és működését, valamint kiemelve annak pedagógiai értékét az interaktív rendezési algoritmusok tanításában. Az alkalmazás lehetővé teszi a diákok számára, hogy aktívan részt vegyenek a tanulási folyamatban, motiválja őket a gyakorlásra és fejleszti a logikus gondolkodásukat.

Az alábbi linken megtalálható a projekt.

<https://github.com/LazarZsolt13/SortRaceBackEnd>

<https://github.com/LazarZsolt13/SortRaceFronEnd>

Köszönetnyilvánítás

Ezúton is szeretném megköszöni a Sapientia egyetemnek, hogy lehetővé tették az, hogy itthon tanuljak az anyanyelvemen. Nagyon sok nagyszerű tanárt ismerhettem meg, akiknek komoly munkájuk látszik ebben a dolgozatban. Ezekben az években kiváló informatikai képzést kaptam.

Külön szeretném kiemelni Dr. Kátai Zoltánt a vezetőtanáromat, aki segítette türelemmel és odaadással ennek a projektnek a létrejöttét.

Irodalomjegyzék

- [1] K. Zoltán, *C: nyelv és programozás (C: limbaj și programare)*. Ungaria: Universitatea Debrecin, 2008.
- [2] —, *Algoritmusok felülnézetből (Algoritmi – o privire de ansamblu)*. Cluj-Napoca: Editura Scientia, 2007.
- [3] „Angular architecture,” <https://angular.io/guide/architecture>.
- [4] „Spring boot,” <https://spring.io/projects/spring-boot>.