**1. Általános Leírás**

**Projekt Célja**

A projekt célja a webprogramozás alapjainak elsajátítása különböző technológiák (HTML, CSS, JavaScript, React) segítségével. A feladatok során a hallgatók több dinamikus weboldalt hoznak létre, melyek tartalmazzák a CRUD műveleteket, AJAX kéréseket, React alkalmazásokat és HTML5 API-kat.

**Felhasznált Technológiák**

* **HTML5 & CSS3**: Az alap weboldalak szerkezetének és stílusának meghatározására.
* **JavaScript**: Dinamikus tartalom kezelésére, CRUD funkciók és AJAX működtetésére.
* **ReactJS**: Egyoldalas alkalmazások fejlesztésére, komponensek és állapotkezelés használatával.
* **Chart.js**: Adatok vizualizálása grafikonokon.
* **AJAX (Fetch API)**: Aszinkron adatbetöltés és küldés egy API-val.
* **GitHub**: Verziókezelés és közös munka biztosítása.
* **Internet Tárhely**: A kész projekt internetes elérhetőségének biztosítása.

**Projekt Feladatok**

A projekt során a hallgatók a következő feladatokat oldják meg:

1. **Vízszintes menüvel ellátott weboldal** (alap HTML és CSS)
2. **Táblázat CRUD funkciókkal**, ahol a felhasználók adatokat adhatnak hozzá, szerkeszthetnek, törölhetnek, és rendezhetnek.
3. **HTML5 API példák** bemutatása, beleértve a Web Storage, Geolocation, Canvas stb.
4. **Chart.js** használata egy 5x5-ös táblázat adataiból vonaldiagram készítésére.
5. **AJAX alkalmazás** egy külső API-val való kommunikációra (Create, Read, Update, Delete).
6. **Objektumorientált JavaScript** alkalmazás, osztályokkal és metódusokkal.
7. **React egyoldalas alkalmazás** két egyszerű alkalmazással: számológép és számláló

**2. Oldalak Bemutatása**

**2.1 index.html – Alap Oldal**

**Leírás**

Az **index.html** oldal tartalmazza a honlap alapstruktúráját, amely a következő elemekből épül fel:

* **Header**: A weboldal fejlécét tartalmazza, amelyben H1-es címsor szerepel: "Web-programozás-1 Előadás Házi feladat".
* **Footer**: Az oldal láblécében a készítők nevei és Neptun kódjai találhatóak.
* **Aside**: Az oldalsó menü, amely tartalmazza az aktív menüpontot.
* **Nav**: A vízszintes navigációs menü, amelyben az oldalak közötti navigáció történik.
* **Content (div-content)**: A tartalmi rész, amely oldalonként változik, és itt található az oldal specifikus tartalma.

A navigációs menüben kiemeljük az aktuálisan aktív menüpontot, hogy a felhasználó láthassa, melyik oldalon tartózkodik.

**Feladatok a index.html-en belül:**

* **Vízszintes menü** létrehozása HTML-ben.
* **Aktív menüpont** dinamikus kiemelése CSS segítségével.
* **Címsor** és **lábléc** hozzáadása.
* A **header** tartalmazza az oldalcímeket.
* A képen szöveg, képernyőkép, szoftver, Betűtípus látható

  Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.A **footer** a készítők információit tartalmazza.

**2.2 tabla.html – Táblázat Oldal**

**Leírás**

A **tabla.html** oldal tartalmaz egy 4x4-es táblázatot, amely lehetőséget biztosít a következő CRUD műveletek végrehajtására:

* **Create**: Új sorok hozzáadása a táblázathoz.
* **Read**: Adatok megjelenítése a táblázatban.
* **Update**: Módosítási lehetőség az egyes sorokhoz.
* **Delete**: Sorok törlése a táblázatból.

**Feladatok a tabla.html-en belül:**

* A táblázat legalább 4x4-es legyen.
* Minden mezőhöz érvényesítést kell alkalmazni (pl. nem lehet üres, minimum és maximum hossz).
* Lehetőség legyen bármely oszlop szerint rendezni a táblázatot.
* A táblázatban keresési és szűrési funkcióval lehet keresni oszlopok alapján.

**Műveletek:**

1. **Add new row**: Legyen egy űrlap, amiben a felhasználó be tudja írni az adatokat (name, height, weight), és azokat hozzáadhatja a táblázathoz.
2. **Edit row**: Legyen egy módosítási lehetőség, ahol az egyes sorok adatai szerkeszthetőek.
3. **Delete row**: Minden sorhoz tartozzon egy törlés gomb.

A képen képernyőkép, szöveg, fekete látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

**2.3 html5.html – HTML5 Újdonságok Oldal**

**Leírás**

Ezen az oldalon több kisebb **példát** kell bemutatni, mindegyik egy-egy HTML5 API funkcióra.  
Ezek lehetnek egy oldalon **egymás alatt**, vagy külön **almenübe** szervezve is.

**Elkészítendő példák:**

| **API neve** | **Mit kell bemutatni?** |
| --- | --- |
| **Web Storage** | Mentés helyi tárhelyre (localStorage / sessionStorage) |
| **Web Workers** | Háttérfolyamatban számolás pl. nagy számok összeadása |
| **Server-Sent Events** | Egy egyszerű automatikus adatfrissítés (pl. idő) |
| **Geolocation API** | Felhasználó helyzetének lekérése |
| **Drag and Drop API** | Egérrel áthúzható elem egy másik dobozba |
| **Canvas** | Egyszerű grafika: pl. rajzolj egy négyzetet vagy kört |
| **SVG** | Egyszerű SVG alakzatok, pl. egy színes kör vagy téglalap |

**További részletek:**

* **Web Storage**:
  + Készíts egy input mezőt és egy mentés gombot. A beírt adatot tárolja localStorage-ben.
  + Ugyaninnen olvassa ki, ha újratöltöd az oldalt.
* **Web Worker**:
  + Hozz létre egy külön .js fájlt, amiben háttérszámítás zajlik (pl. egy nagy számig számolás).
* **Server-Sent Events**:
  + Frissítse az aktuális időt pár másodpercenként (dummy adat helyett JS setInterval is jó helyettesítés, mivel nincs saját szerverünk).
* **Geolocation API**:
  + Egy gombnyomásra jelenítse meg a koordinátákat (latitude, longitude).
* **Drag and Drop**:
  + Legyen két div, az egyiket húzd át a másikba.
* **Canvas**:
  + Rajzoljunk egy kék kört egy <canvas> elemre.
* **SVG**:
  + Hozz létre egy piros négyzetet SVG-vel.

A képen szöveg, képernyőkép, Betűtípus, szám látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

A képen szöveg, képernyőkép, Betűtípus, diagram látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

**2.4 chart.html – ChartJS Diagram Oldal**

**Leírás**

Ezen az oldalon egy **5x5-ös számokkal teli táblázatot** kell megjeleníteni.  
A felhasználó **kiválaszt egy sort** → abból egy **vonaldiagramot** (line chart) generálunk Chart.js segítségével.

**Feladat részletei:**

* Készíts egy egyszerű 5x5-ös táblázatot számokkal (pl. 5 sor, 5 oszlop – számok lehetnek 1-100 között).
* Minden sor végén legyen egy **"Kijelöl"** gomb.
* Ha a felhasználó megnyomja a gombot:
  + A sor adatait ábrázold egy **vonaldiagramon** (Line Chart) Chart.js-sel a táblázat **alatt**.
* A diagram mindig a kiválasztott sor számait mutassa.
* **2.5 ajax.html + ajax.js – AJAX Oldal**
* **Leírás**
* Itt egy **valódi API**-t kell használnod a CRUD (Create, Read, Update, Delete) műveletekhez.  
  **API címe:**  
  <http://gamf.nhely.hu/ajax2/>
* (A code paraméterben majd saját Neptun + kiegészítést használsz → pl.: ABCD12xyz123.)
* **Feladat részletek:**
* **1. Read (lekérdezés)**
* Olvasd ki az adatokat AJAX hívással (fetch vagy XMLHttpRequest).
* Írd ki az összes rekordot egymás alá, szép formázottan.
* A height értékekről számítsd ki:
* **Összeg**
* **Átlag**
* **Legnagyobb érték**
* **2. Create (új adat felvétele)**
* Legyenek input mezők: név, magasság, súly.
* Validation:
* Nem lehet üres.
* Max 30 karakter hosszúság.
* Sikeres mentés után jelezze az oldalon ("Sikeres mentés!" pl.)
* **3. Update (adat módosítása)**
* Beírod az id-t egy mezőbe + "Adat betöltése" gombbal kiolvasod az adatot.
* Megjeleníti az input mezőkben → módosíthatod.
* Majd "Módosítás" gombbal küldi el az új adatot.
* **4. Delete (törlés)**
* Egy input mező, amibe az id-t írod.
* "Törlés" gomb → törli az adott rekordot.
* Sikeres törlés esetén visszajelzés az oldalon. A képen sor, diagram, Diagram, szöveg látható

  Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

**2.5 ajax.html + ajax.js – AJAX Oldal**

**Leírás**

Itt egy **valódi API**-t kell használnod a CRUD (Create, Read, Update, Delete) műveletekhez.  
**API címe:**  
<http://gamf.nhely.hu/ajax2/>

(A code paraméterben majd saját Neptun + kiegészítést használsz → pl.: ABCD12xyz123.)

**Feladat részletek:**

**1. Read (lekérdezés)**

* Olvasd ki az adatokat AJAX hívással (fetch vagy XMLHttpRequest).
* Írd ki az összes rekordot egymás alá, szép formázottan.
* A height értékekről számítsd ki:
  + **Összeg**
  + **Átlag**
  + **Legnagyobb érték**

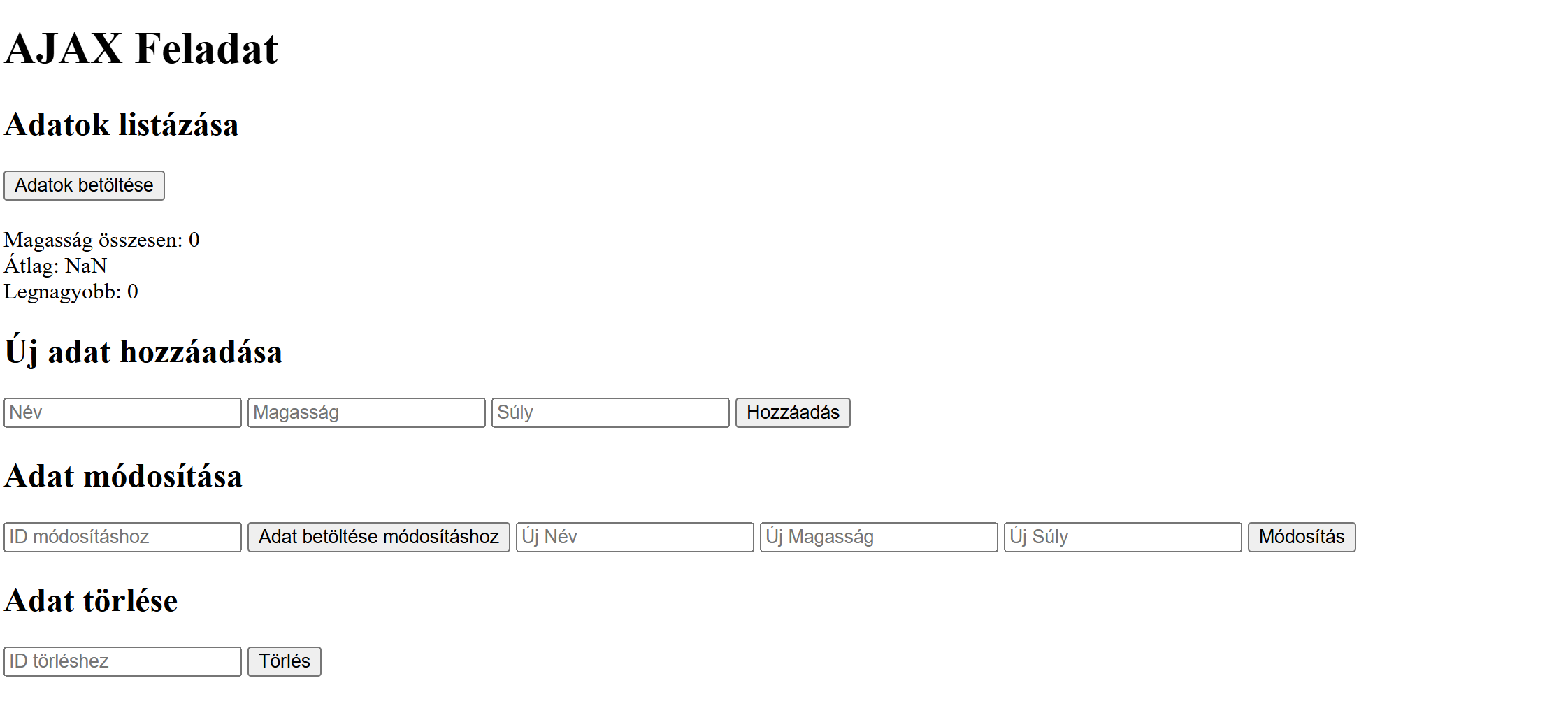
**2. Create (új adat felvétele)**

* Legyenek input mezők: név, magasság, súly.
* Validation:
  + Nem lehet üres.
  + Max 30 karakter hosszúság.
* Sikeres mentés után jelezze az oldalon ("Sikeres mentés!" pl.)

**3. Update (adat módosítása)**

* Beírod az id-t egy mezőbe + "Adat betöltése" gombbal kiolvasod az adatot.
* Megjeleníti az input mezőkben → módosíthatod.
* Majd "Módosítás" gombbal küldi el az új adatot.

**4. Delete (törlés)**

* Egy input mező, amibe az id-t írod.
* "Törlés" gomb → törli az adott rekordot.
* Sikeres törlés esetén visszajelzés az oldalon.
* 

**2.6 – OOJS menü**

**Feladat:**

* Készíts egy **szabadon választott** **Objektumorientált JavaScript** alkalmazást.
* Alkalmazd benne ezeket:
  + class
  + constructor
  + metódusok (tehát az osztályon belül műveletek, funkciók)
  + document.body.appendChild
  + extends
  + super
* <!DOCTYPE html>
* <html lang="hu">
* <head>
* <meta charset="UTF-8">
* <title>OOJS</title>
* </head>
* <body>
* <script>
* class Animal {
* constructor(name) {
* this.name = name;
* }
* speak() {
* const p = document.createElement('p');
* p.textContent = `${this.name} hangot ad ki.`;
* document.body.appendChild(p);
* }
* }
* class Dog extends Animal {
* constructor(name, breed) {
* super(name);
* this.breed = breed;
* }
* bark() {
* const p = document.createElement('p');
* p.textContent = `${this.name} (fajta: ${this.breed}) ugat: Vau vau!`;
* document.body.appendChild(p);
* }
* }
* const animal = new Animal("Általános állat");
* animal.speak();
* const dog = new Dog("Bodri", "Puli");
* dog.speak();
* dog.bark();
* </script>
* </body>
* </html>

2.7 **React alkalmazás – Egyoldalas alkalmazás (SPA)**

A projekt egy **React egyoldalas alkalmazás (SPA)**.

Két menüpont van: **"Számláló"** és **"Névlista"**.

A React alkalmazás **helyi telepítéssel** készült, nem Babel/standalone módban.

**Névlista (NameList) alkalmazás**

* Egy input mezővel új nevet lehet hozzáadni egy listához.
* A hozzáadott nevek megjelennek egy <ul> listában.
* Használt technikák:
  + useState a nevek tömbjének tárolására.
  + Eseménykezelés onChange, onClick-kel.
* function NameList() {
* const [name, setName] = useState('');
* const [names, setNames] = useState([]);
* const handleAddName = () => {
* if (name.trim() !== '') {
* setNames([...names, name]);
* setName('');
* }
* };
* return (
* <div style={{ padding: '20px' }}>
* <h2>Névlista</h2>
* <input
* type="text"
* placeholder="Írj be egy nevet"
* value={name}
* onChange={(e) => setName(e.target.value)}
* style={{ marginRight: '10px' }}
* />
* <button onClick={handleAddName}>Hozzáadás</button>
* <ul style={{ marginTop: '20px' }}>
* {names.map((n, index) => (
* <li key={index}>{n}</li>
* ))}
* </ul>
* </div>
* );
* }
  + A képen szöveg, képernyőkép, Betűtípus látható

    Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

A **Számláló** egy egyszerű React komponens, amely egy számot (counter értéket) tárol, és három gomb segítségével módosítható:

* **Növelés**: a számláló értéke eggyel nő.
* **Csökkentés**: a számláló értéke eggyel csökken.
* **Nullázás**: a számláló értéke nullára áll vissza.

A komponens a React useState hookját használja az aktuális számlálóérték (count) tárolására és módosítására.

**Működés röviden**

* Kezdetben az érték **0**.
* Minden gombkattintás egy eseménykezelő függvényt hív meg:
  + **Növelés**: setCount(count + 1)
  + **Csökkentés**: setCount(count - 1)
  + **Nullázás**: setCount(0)
* Az aktuális érték automatikusan frissül a képernyőn.
* function Counter() {
* const [count, setCount] = useState(0);
* return (
* <div style={{ padding: '20px' }}>
* <h2>Számláló</h2>
* <p>Érték: {count}</p>
* <button onClick={() => setCount(count + 1)}>Növelés</button>
* <button onClick={() => setCount(count - 1)}>Csökkentés</button>
* <button onClick={() => setCount(0)}>Nullázás</button>
* </div>
* );
* }

A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

**Feladat: Web-programozás 1 – Házi feladat**

A beadandó célja, hogy egy dinamikus, modern weboldalt készítsünk el, amely számos front-end technológiát alkalmaz. A projekt során többféle feladatot kellett megoldani, amelyek különböző szintű kihívásokat jelentettek, mind funkcionálisan, mind vizuálisan. A feladatok középpontjában a **React**, **JavaScript**, **HTML**, és **CSS** álltak. Az alábbiakban részletesen bemutatom a feladatokat és azok megvalósítását:

**1. Alap oldal struktúra és menü**

A projekt első lépése egy statikus weboldal létrehozása volt, amely vízszintes menüvel rendelkezik. A **header** és **footer** fix struktúrával készült, míg az oldaltartalom minden aloldalon egyedi volt. Az aktív menüpontot dinamikusan jelöltem, és a fejlécben a feladat címét, míg a láblécben az elkészítők nevét és Neptun kódját jelenítettem meg.

**2. Táblázat CRUD funkciókkal**

A táblázatot **CRUD (Create, Read, Update, Delete)** funkciókkal bővítettem. A felhasználó képes adatokat hozzáadni, frissíteni és törölni egy **4x4-es táblázatban**. A táblázat adatainak validálása és rendezése is megvalósult. Az oszlopokban szűrhetünk és kereshetünk, így az interaktivitás biztosítva van.

**3. HTML5 API példák (3 pont)**

A feladat során több **HTML5 API**-t is implementáltam, mint például:

* **Web Storage**: helyi tárolás használata.
* **Canvas**: egyszerű grafikus rajzolás.
* **Geolocation**: helymeghatározás API.
* **Drag and Drop**: elemek áthelyezése az oldalon.

Ezeket a funkciókat külön komponensbe szerveztem, hogy mindegyik API működése könnyen tesztelhető és bemutatható legyen.

**4. Chart.js használata**

A projekt során egy **5x5-ös táblázat** adatait vonaldiagramon jelenítettem meg a **Chart.js** segítségével. Az adatokat a felhasználó által kiválasztott sor alapján jelenítettem meg, így az interakció szempontjából rugalmas lett.

**5. AJAX alkalmazás**

Az AJAX alkalmazás az API használatával valósult meg. A **CRUD műveletek** az API segítségével történtek. Az adatokat **cURL** és **Postman** segítségével teszteltem a helyes működés érdekében. Az API integrációját követően sikerült a weboldalon dinamikusan megjeleníteni az adatokat, amelyeket a felhasználó az oldal betöltése után is elérhetett.

**6. Objektumorientált JavaScript**

A JavaScript objektumorientált programozást is alkalmaztam a projektben, ahol egy szabadon választott alkalmazásban osztályokat, konstruktorokat, metódusokat, valamint az **extends** és **super** kulcsszavakat is felhasználtam.

**7. React alkalmazás**

A projektben egy egyoldalas **React alkalmazás** készült, amely két menüpontot tartalmazott:

* Az első menüpont egy egyszerű **számláló** komponens volt, amely lehetővé tette a felhasználó számára a szám növelését, csökkentését és nullázását.
* A második menüpont egy egyszerű **névlista** kezelésére szolgált, amelyben a felhasználók hozzáadhatták, frissíthették és törölhették saját nevüket.

A React alkalmazásban alkalmaztam a **useState** hook-ot, a komponensek közötti adatkezelést és az események kezelését.

**Záró megjegyzés**

A projekt megvalósítása során sikerült minden egyes feladatot teljesíteni, és a kívánt funkciókat implementálni. A modern webes technológiák használata, mint a React és a JavaScript, lehetővé tette, hogy dinamikus és interaktív alkalmazásokat hozzunk létre, amelyek felhasználóbarátak és könnyen kezelhetők.