SMART KITCHEN – Web-programozás 1 Beadandó dokumentáció

Készítették: Orosz-Naszádi Krisztina és Bűdy Szabolcs Neptun kódok: QS3OF7 és F1WQL0

Tantárgy: Webprogramozás 1 – Előadás Tanár: Dr. Subecz Zoltán

Projekt neve: Smart Kitchen

GitHub Repository**:** <https://github.com/Orosznaszadi/SmartKitchen> Publikus elérhetőség: <http://smartkitchen.nhely.hu/>

# Bevezetés

A Smart Kitchen projekt célja egy összetett, kliensoldali technológiákat használó, interaktív webalkalmazás létrehozása, amely egy modern konyhai tematikára épül. A fejlesztés során szigorúan csak HTML, CSS, JavaScript és React eszköztárat alkalmaztunk, szerveroldali technológia nélkül, a Web-programozás 1 tantárgy beadandó követelményeinek megfelelően. A weboldal egy többoldalas statikus szerkezetből és két különálló React Single Page Application- ből (SPA) áll, melyeket egységes vizuális megjelenés és logikus felépítés jellemez. A projekt célja nem csupán az, hogy bemutasson bizonyos technológiai megoldásokat, hanem az is, hogy ezeket egy tematikus és jól működő környezetbe illessze.

A projekt során olyan funkciók kerültek implementálásra, mint:

* + dinamikus terméktáblázat (CRUD műveletekkel),
  + keresés és rendezés JavaScript segítségével,
  + kosárkezelés,
  + Chart.js integráció táblázat-adatok grafikus megjelenítésére,
  + AJAX műveletek egy külső API segítségével (POST/GET/DELETE/UPDATE),
  + HTML5 API-k (pl. localStorage, Web Workers, Geolocation, Canvas, SVG, SSE),
  + két React alkalmazás: egy Webshop és egy Delivery-játék.

E dokumentáció célja, hogy részletesen bemutassa az elkészült rendszer működését, a megvalósítás során használt technikákat, a kódok szerkezetét és logikáját, valamint a munkamegosztást.

# Az oldal általános felépítése

A Smart Kitchen projekt statikus HTML oldalai egységes szerkezetet követnek, mely biztosítja a könnyű bővíthetőséget, átláthatóságot és a navigáció következetességét. Minden oldal

alapstruktúrája az alábbi elemekből épül fel:

* + **<header>** – a főcímet tartalmazza:

Web-programozás 1 Előadás Házi Feladat

* + **<nav>** – vízszintes menüsor az aloldalakhoz
  + **<aside>** – itt helyeztük el a közösségi média ikonokat (Facebook és Instagram), amelyek az oldal vizuális egységét erősítik, és hangulati elemként szolgálnak
  + **<div id="content">** – az adott aloldal tartalma, ez dinamikusan változik
  + **<footer>** – készítők nevei és Neptun kódjai

Ez a szerkezeti logika minden HTML oldalra érvényes, beleértve az index.html, menu.html, contact.html, order.html, ajax.html, chartjs.html, oojs.html fájlokat is.

A navigáció logikáját a menu.js kezeli: aktív menüpont kijelölés, navigációs események,

aloldalak betöltése. Az egységes dizájnt a styles.css biztosítja, a reszponzív megjelenéshez rugalmas méretekkel, Flexbox alapú elrendezéssel. A megjelenés modern, minimalista stílusú, mobilbarát, világos színpalettával és jól olvasható betűtípusokkal dolgozik.

# Menüoldal (menu.html) – CRUD műveletek + interaktivitás

A menu.html az egyik legkomplexebb oldal, ahol a felhasználó interaktív módon kezelhet egy terméklistát. A menu.html fájl tartalmaz egy űrlapot, egy táblázatot, keresőmezőt, rendezési

lehetőséget, valamint egy dinamikus kosarat is. Ezek együttműködését a script.js biztosítja. A fő tartalmi blokk egy JavaScript által kezelt táblázat, amely támogatja a CRUD műveleteket**:** Create (létrehozás), Read (megtekintés), Update (módosítás), Delete (törlés). A felhasználó a megfelelő beviteli mezők segítségével tud új terméket hozzáadni: név, ár, kategória, mennyiség megadása után a „Hozzáadás” gombra kattintva a termék azonnal megjelenik a táblázatban.

Cél: bemutatni a táblázat működését (CRUD), a kosár interakcióját, validációt, keresést, rendezést

Fájlok: menu.html, script.js, menu.js, styles.css

# Létrehozás (Create)

Az új termék létrehozása előtt a program validálja az adatokat:

* A név legalább 2 karakter hosszúságú legyen
* Az ár csak pozitív szám lehet
* A mezők nem lehetnek üresek Kódrészlet – Validált termék hozzáadása:

javascript

MásolásSzerkesztés

function addProduct(name, price, category) { if (name.trim().length < 2) {

alert("A név túl rövid!"); return;

}

if (isNaN(price) || price <= 0) {

alert("Az ár csak pozitív szám lehet!"); return;

}

const newProduct = { id: Date.now(), name, price, category }; products.push(newProduct);

renderTable();

}

# Megtekintés (Read)

A táblázat alapértelmezetten betölti az összes terméket a products tömbből. A renderTable() függvény újrarendereli a teljes táblázatot, ha módosítás vagy szűrés történik.

# Módosítás (Update)

A felhasználó a meglévő sorokat szerkesztheti: a „Szerkesztés” gombra kattintva az adatok betöltődnek a mezőkbe, majd mentés után frissül a sor.

Kódrészlet – Sor szerkesztése:

javascript

MásolásSzerkesztés function editProduct(id) {

const product = products.find(p => p.id === id); nameInput.value = product.name;

priceInput.value = product.price; categoryInput.value = product.category; currentEditId = id;

}

# Törlés (Delete)

Minden sor mellett található egy „Törlés” gomb, amellyel a felhasználó eltávolíthatja a terméket. A sor azonnal eltűnik a táblázatból.

# Rendezés és Keresés

A felhasználó bármely oszlopcímre kattintva sorba rendezheti az adatokat növekvő vagy csökkenő sorrendben. Ezt egy sortTableBy(column) függvény valósítja meg.

A felső keresőmező segítségével valós idejű szűrés történik, amely csak a megadott szóra illeszkedő sorokat jeleníti meg, minden karakterbevitelnél frissül a találati lista.

Kódrészlet – Keresés:

javascript

MásolásSzerkesztés searchInput.addEventListener("input", e => {

const searchTerm = e.target.value.toLowerCase();

filtered = products.filter(p =>

p.name.toLowerCase().includes(searchTerm)

);

renderTable(filtered);

});

# Kosárkezelés

Minden termékhez tartozik egy *Kosárba* gomb. A kosár az oldalsávban található, és minden egyes hozzáadás után frissül. A felhasználó bármely terméket kosárba tehet, majd később onnan eltávolíthatja. A kosár az összesített árakat is kiszámítja és kijelzi.

A kosár az alábbi logikával működik:

* + egyedi azonosító alapján azonosítja a terméket
  + növeli a mennyiséget, ha újra hozzáadjuk
  + újraszámolja az összértéket Kódrészlet – Kosár:

javascript

MásolásSzerkesztés function addToCart(id) {

const product = products.find(p => p.id === id); const existing = cart.find(c => c.id === id);

if (existing) {

existing.quantity++;

} else {

cart.push({ ...product, quantity: 1 });

}

renderCart();

}

A renderCart() függvény frissíti a kosár DOM-ot, újraszámolja az összértéket, és lehetővé teszi a törlést gombnyomásra.

# HTML5 API-k bemutatása

A projekt egyik legfontosabb célkitűzése az volt, hogy modern, böngésző által támogatott HTML5 API-kat demonstráljunk, gyakorlati példákon keresztül. Minden egyes API külön fájlban vagy

külön szekcióban szerepel, a menu.html alatt elérhető almenüként, így logikusan szétválasztva és jól strukturáltan mutatja be azokat.

Az alábbiakban ismertetjük, hogy mely API-kat implementáltuk, hogyan működnek, milyen célra használhatók, és mit lát a felhasználó az oldalon.

# Web Storage API – storage.js

A Web Storage API segítségével adatok menthetők és olvashatók vissza localStorage vagy sessionStorage használatával. Mi localStorage-t alkalmaztunk, így az adatok a böngésző bezárása után is megmaradnak.

Az oldalra látogató felhasználó egy űrlapon keresztül beírhat például egy nevet vagy kedvenc ételt, amelyet a rendszer automatikusan eltárol. A következő látogatáskor ezek az értékek

visszatöltődnek.

Kódrészlet:

javascript

MásolásSzerkesztés

document.getElementById("saveBtn").addEventListener("click", () => { const value = document.getElementById("favFood").value; localStorage.setItem("favFood", value);

});

window.addEventListener("load", () => {

const saved = localStorage.getItem("favFood");

if (saved) document.getElementById("favFood").value = saved;

});

A beírt adat újra megjelenik, nem kell újra begépelni.

# Web Workers – worker.js, workerDemo.js

A Web Workers lehetővé teszik, hogy hosszabb számítási feladatokat ne a főszálon futtassunk, így az oldal nem akad meg.

A példában egy számláló fut háttérfolyamatként (worker.js), amit a főoldalon indíthatunk (workerDemo.js).

Kódrészlet: javascript

MásolásSzerkesztés

// worker.js

self.onmessage = function(e) {

let count = 0;

setInterval(() => {

postMessage(++count);

}, 1000);

};

javascript

MásolásSzerkesztés

// workerDemo.js

const worker = new Worker("worker.js"); worker.onmessage = (e) => {

document.getElementById("output").innerText = `Idő: ${e.data} másodperc`;

};

Folyamatosan frissülő időszámláló – az oldal akadás nélkül működik.

# Server-Sent Events – sse.js

Ez az API lehetővé teszi, hogy a szerver időnként automatikusan adatokat küldjön a kliens felé. Ez egyirányú kommunikáció, ideális például élő értesítésekre.

A példánkban a kliens egy message eseményt hallgat, amelyet a szerver 5 másodpercenként küld.

Kódrészlet:

javascript

MásolásSzerkesztés

const eventSource = new EventSource("server.php"); eventSource.onmessage = (event) => {

const container = document.getElementById("messages"); container.innerHTML += `<p>${event.data}</p>`;

};

Új üzenetek jelennek meg automatikusan, frissítés nélkül.

# Geolocation API – geo.js

Ez az API a felhasználó földrajzi helyzetének meghatározására szolgál. A példánkban egy gombra kattintva lekérjük az aktuális szélességi és hosszúsági koordinátákat, amelyeket kiírunk a képernyőre.

Kódrészlet:

javascript

MásolásSzerkesztés

navigator.geolocation.getCurrentPosition(position => { const { latitude, longitude } = position.coords;

document.getElementById("geo").innerText = `Lat: ${latitude}, Lon: ${longitude}`;

});

Az oldal megmutatja, hol van a felhasználó a böngésző szerint.

# Drag and Drop API – dragdrop.js

Lehetővé teszi, hogy elemeket húzd-és-ejtsd módszerrel mozgassunk. A példánkban a felhasználó egy ételt húzhat a kosárba.

Kódrészlet:

javascript

MásolásSzerkesztés

item.ondragstart = e => e.dataTransfer.setData("text", e.target.id); cart.ondrop = e => {

const id = e.dataTransfer.getData("text"); const item = document.getElementById(id); e.target.appendChild(item);

};

# Canvas API – canvas.js

A Canvas lehetővé teszi, hogy programozottan rajzoljunk a böngészőben. A példánkban egy animált szállító autó mozog a képernyőn (order.html – rendelés).

Kódrészlet:

javascript

MásolásSzerkesztés

const canvas = document.getElementById("myCanvas"); const ctx = canvas.getContext("2d");

let x = 50;

function draw() {

ctx.clearRect(0, 0, canvas.width, canvas.height); ctx.fillRect(x, 100, 50, 50);

x += 2;

requestAnimationFrame(draw);

}

draw();

Látványos, animált interakció – egyedi hangulatot ad az oldalnak.

# SVG – Logómegjelenítés

A projekt logója egy SVG fájl, amely a menu.html-ben helyezkedik el. Az SVG méretezhető, vektoros formátum.

Kódrészlet:

html

MásolásSzerkesztés

<img src="logo.svg" alt="Smart Kitchen logó" width="100" />

Tiszta, éles megjelenés bármilyen képernyőn.

# Chart.js – Adatok megjelenítése vonaldiagramon

A chartjs.html aloldalon egy táblázatban jelenítjük meg az adatokat, és ezekből vonaldiagramot generálunk a Chart.js könyvtár segítségével. Ez az oldal az adatelemzés vizuális megközelítését demonstrálja, amely a modern webalkalmazásokban egyre elterjedtebb.

# Funkciók:

* + A táblázat sora kattintható
  + A kattintott sor adatai dinamikusan megjelennek vonaldiagram formájában
  + Az új adat automatikusan frissíti a grafikon kinézetét

# Technológia

A Chart.js egy egyszerű, de erőteljes JavaScript könyvtár, amelyet beágyaztunk a projektbe egy CDN-linken keresztül. A canvas elem szolgál a grafikon kirajzolásának céljára.

html

MásolásSzerkesztés

<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>

A táblázat HTML-ben van definiálva, és minden sorhoz eseményfigyelőt kapcsoltunk, amely JavaScript segítségével lekéri az értékeket, és frissíti a diagramot.

# Működés részletesen

1. A felhasználó rákattint egy sorra a táblázatban.
2. A JavaScript egy Array.from(row.cells) művelettel begyűjti a cellák értékeit.
3. Ezeket számokká konvertálja, majd átadja a Chart.js-nek.
4. A chart.data.datasets[0].data tömb frissül.
5. A chart.update() újrarajzolja a diagramot valós időben. Kódrészlet – kattintás és diagramfrissítés

javascript

MásolásSzerkesztés

function drawChart(values) { chart.data.labels = ['1', '2', '3', '4', '5']; chart.data.datasets[0].data = values; chart.update();

}

document.querySelectorAll("#dataTable tbody tr").forEach(row => { row.addEventListener("click", () => {

const values = Array.from(row.cells).map(cell => Number(cell.textContent)); drawChart(values);

});

});

A fenti logika biztosítja, hogy a felhasználó azonnal láthassa a választott sor numerikus értékeinek vizuális ábrázolását.

# Felhasználói élmény

Ez az oldal remekül bemutatja, hogyan lehet adatot egyszerűen vizualizálni JavaScript és Chart.js segítségével. A táblázatban szereplő adatok nem csak számsorok – azok interakcióba is lépnek a grafikon motorral, így sokkal érthetőbbé és látványosabbá válnak.

A vizualizáció segíti a felhasználót az adatok közötti összefüggések gyors felismerésében – például melyik érték kiugró, vagy hol van trend.

# AJAX kapcsolat API-val – ajax.html, ajax.js

A ajax.html aloldalon egy interaktív felületet hoztunk létre, amely kapcsolatban áll egy távoli szerverrel az alábbi API-n keresztül:

API URL: <http://gamf.nhely.hu/ajax2/>

Ez az API támogatja az alábbi műveleteket:

* + create – új rekord létrehozása
  + read – rekordok lekérdezése
  + update – meglévő rekord módosítása ID alapján
  + delete – rekord törlése

A kommunikáció POST metóduson keresztül történik, application/x-www-form-urlencoded formátumban. Minden kéréshez szükséges a code paraméter, amely azonosítja a felhasználót és az adatainak szeparálását biztosítja.

# Lekérdezés (Read)

A readData() függvény POST kérést küld op=read és code=NeptunXYZ paraméterekkel, és a válaszként kapott JSON tömböt kiírja a képernyőre.

Kódrészlet – adatok lekérése:

javascript

MásolásSzerkesztés function readData() {

fetch("<http://gamf.nhely.hu/ajax2/>", {

method: "POST",

body: new URLSearchParams({ op: "read",

code: "ABC123xyz456"

})

})

.then(response => response.json())

.then(data => displayData(data.list));

}

Kimenet:

json

MásolásSzerkesztés

{

"rowCount": 2,

"list": [

{ "id": "1", "name": "Zöldség", "height": "180", "weight": "75", "code": "ABC123xyz456" },

...

]

}

# Statisztikai feldolgozás

A lekért lista alapján az alábbiakat számoljuk ki:

* + height értékek átlaga
  + legnagyobb érték
  + összeg

Kódrészlet – statisztika számítás:

javascript

MásolásSzerkesztés

const heights = data.list.map(item => parseInt(item.height)); const total = heights.reduce((a, b) => a + b, 0);

const avg = (total / heights.length).toFixed(2);

const max = Math.max(...heights);

A kapott eredményeket a felhasználó az oldal alján azonnal látja.

# Új rekord létrehozása (Create)

A felhasználó űrlapon keresztül adhat meg nevet, magasságot és testsúlyt. A mezők validálva vannak:

* + nem lehetnek üresek
  + maximális hosszuk 30 karakter Kódrészlet – validáció + küldés:

javascript

MásolásSzerkesztés function createData() {

if (name.length === 0 || name.length > 30) return alert("Hibás név");

// stb...

fetch("<http://gamf.nhely.hu/ajax2/>", { method: "POST",

body: new URLSearchParams({ op: "create",

name, height, weight,

code: "ABC123xyz456"

})

})

.then(res => res.json())

.then(() => alert("Sikeres rögzítés"));

}

# Módosítás (Update)

1. Az ID mezőbe beírjuk a rekord azonosítóját
2. A "getDataForId" gomb betölti a meglévő adatokat
3. A mezők szerkeszthetők
4. A "Mentés" gomb elküldi a módosított adatokat

# Törlés (Delete)

A felhasználó egy ID-t megadva törölhet egy rekordot. A sikeres törlésről az oldal vizuális visszajelzést ad.

# Felhasználói élmény

Az AJAX-os oldal célja, hogy a felhasználó úgy végezhessen adatkezelési műveleteket, hogy ne kelljen az oldalt újratöltenie. Minden művelet valós időben történik, és a visszajelzések (alert, lista frissítés, statisztika) azonnal megjelennek. Ez már egy modern, dinamikus felhasználói élményt nyújt.

# Objektumorientált JavaScript – oojs.html

Ebben a részben bemutatjuk az objektumorientált JavaScript (OOJS) megközelítés gyakorlati

alkalmazását. A oojs.html oldalon saját class osztályokat definiáltunk, amelyek lehetővé teszik, hogy a kód strukturáltabb, könnyebben karbantartható és kiterjeszthető legyen.

A cél az volt, hogy a DOM-manipulációkat (pl. elemek létrehozása, hozzáadása az oldalhoz) ne sima függvényekkel, hanem objektumorientált módon kezeljük, kihasználva a modern JavaScript nyelvi eszközeit.

# Alkalmazott OOJS eszközök:

* + class kulcsszó: osztály létrehozására
  + constructor(): példányosítási logika
  + Metódusok (pl. render(), update())
  + extends + super(): öröklés, alosztály létrehozása
  + document.body.appendChild(): DOM-manipuláció

# Alaposztály létrehozása – Product

Létrehoztunk egy Product osztályt, amely egy terméket reprezentál. A konstruktor paraméterként kapja a nevét, árát és kategóriáját, majd az element változóba HTML elemet generál.

Kódrészlet:

javascript

MásolásSzerkesztés class Product {

constructor(name, price, category) { this.name = name;

this.price = price;

this.category = category;

}

render() {

const div = document.createElement("div"); div.className = "product";

div.innerHTML = `<h3>${this.name}</h3><p>Ár: ${this.price} Ft</p>`;

return div;

}

}

# Öröklés – DiscountedProduct osztály

A DiscountedProduct az Product alosztálya, amely kiegészül egy új mezővel: kedvezmény. Ez a mező új logikát is hoz: módosítjuk az ár megjelenítését.

Kódrészlet:

javascript

MásolásSzerkesztés

class DiscountedProduct extends Product { constructor(name, price, category, discount) { super(name, price, category);

this.discount = discount;

}

render() {

const div = super.render();

const discounted = this.price - (this.price \* this.discount / 100); div.innerHTML += `<p>Kedvezményes ár: ${discounted} Ft</p>`; return div;

}

}

# Megjelenítés – appendChild a DOM-hoz

Miután példányosítottuk a termékeket, egyszerűen megjelenítjük őket a weboldalon. A document.body.appendChild() segítségével a létrehozott DOM-elemeket hozzáadjuk a főoldalhoz.

javascript

MásolásSzerkesztés

const p1 = new Product("Serpenyő", 3990, "Konyhai eszköz"); const p2 = new DiscountedProduct("Fakanál", 1990, "Faáru", 20);

document.body.appendChild(p1.render()); document.body.appendChild(p2.render());

# Felhasználói élmény

Ahelyett, hogy HTML elemeket kézzel írnánk a DOM-ba, az osztályalapú felépítés lehetővé teszi, hogy bármikor új terméket hozzunk létre és automatikusan jelenítsük meg azt. Ez nemcsak esztétikusabb kódot eredményez, de segíti a bővítést és az új logika bevezetését.

Ez a megközelítés különösen hasznos lehet komplex alkalmazásoknál, ahol számos elem jön- megy a DOM-ban.

# G. React alkalmazás 1: Webshop

A projekt részeként két különálló React alkalmazást valósítottunk meg. Az első egy Webshop, amely modern, egyoldalas alkalmazásként működik, a React keretrendszer alapjaira építve.

A Webshop célja egy egyszerű vásárlói élmény biztosítása: a felhasználó termékeket tekinthet meg, azokat kosárba teheti, majd a kosár tartalmát is kezelheti. A komponensalapú felépítés biztosítja az átláthatóságot és újrafelhasználhatóságot.

# Fejlesztési környezet

* React 19
* Vite (vite.config.js)
* Tailwind CSS keretrendszer
* JSX fájlszerkezet: main.jsx, App.jsx, Home.jsx (webshop), About.jsx, Contact.jsx
* Komponensstruktúra: Navbar.jsx, Footer.jsx

# Alkalmazott funkciók

* Komponens-alapú architektúra
* useState használata: étel lista és a bevásárlókosár
* Terméklista dinamikus kirajzolása tömb alapján
* React Router (menüpontok között navigálás)

Kódrészlet – useState + kosárlogika

const addToCart = (item) => {  
 setCart(prevCart => {  
 const existing = prevCart.find(cartItem => cartItem.id === item.id);  
 if (existing) {  
 return prevCart.map(cartItem =>  
 cartItem.id === item.id  
 ? { ...cartItem, quantity: cartItem.quantity + 1 }  
 : cartItem  
 );  
 } else {  
 return [...prevCart, { ...item, quantity: 1 }];  
 }  
 });  
};

const increaseQuantity = (id) => {  
 setCart(prevCart =>  
 prevCart.map(item =>  
 item.id === id ? { ...item, quantity: item.quantity + 1 } : item  
 )  
 );  
};  
  
const decreaseQuantity = (id) => {  
 setCart(prevCart =>  
 prevCart  
 .map(item =>  
 item.id === id ? { ...item, quantity: item.quantity - 1 } : item  
 )  
 .filter(item => item.quantity > 0) *// ha 0, törlés* );  
};  
  
const removeItem = (id) => {  
 setCart(prevCart => prevCart.filter(item => item.id !== id));  
};

Itt a Home komponens felel az állapotkezelésért, és továbbadja az addToCart függvényt a ProductList komponensnek.

# Terméklista megjelenítése

<div className="flex-1 p-6 max-w-7xl mx-auto space-y-12">  
 <h1 className="text-4xl font-bold mb-8"><i className="fa fa-file-alt"></i> Étlap</h1>  
 {Object.entries(menuData).map(([category, items]) => (  
 <section key={category}>  
 <h2 className="text-3xl font-bold mb-6">{category}</h2>  
 <div className="grid grid-cols-1 sm:grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-6">  
 {items.map((item, index) => (  
 <div key={index} className="bg-white rounded-2xl shadow-md overflow-hidden flex flex-col">  
 <div className="h-40 bg-gray-100 flex items-center justify-center">  
 {item.image ? (  
 <img  
 src={item.image}  
 alt={item.name}  
 className="object-cover h-full w-full"  
 />  
 ) : (  
 <span className="text-gray-500">Nincs kép</span>  
 )}  
 </div>  
 <div className="p-4 text-center flex-grow flex flex-col justify-between">  
 <div>  
 <h3 className="text-lg font-semibold">{item.name}</h3>  
 <p className="text-sm text-gray-500">{item.price} Ft</p>  
 </div>  
 <button  
 onClick={() => addToCart(item)}  
 className="mt-4 bg-green-500 text-white px-4 py-2 rounded hover:bg-green-600"  
 >  
 Kosárba  
 </button>  
 </div>  
 </div>  
 ))}  
 </div>  
 </section>  
 ))}  
</div>

A kosár oldalsáv megjelenítése, ahol a felhasználó megnézheti, mit vásárolt meg.

<div className="w-60 bg-gray-100 p-6 border-l border-gray-300">  
 <h2 className="text-2xl font-bold mb-4"><i className="fa fa-shopping-cart"></i> Kosár</h2>  
 {cart.length === 0 ? (  
 <p className="text-gray-500">A kosár üres.</p>  
 ) : (  
 <ul className="space-y-4">  
 {cart.map((item) => (  
 <li key={item.id} className="border-b pb-3">  
 <div className="flex justify-between items-center">  
 <div>  
 <p className="font-semibold">{item.name}</p>  
 <p className="text-sm text-gray-500">  
 {item.quantity} × {item.price} Ft  
 </p>  
 </div>  
 <div className="text-right">  
 <p className="font-bold">{item.price \* item.quantity} Ft</p>  
 <div className="flex justify-end mt-1 space-x-1">  
 <button  
 className="px-2 py-1 text-sm bg-gray-200 rounded hover:bg-gray-300"  
 onClick={() => decreaseQuantity(item.id)}  
 >  
 −  
 </button>  
 <button  
 className="px-2 py-1 text-sm bg-gray-200 rounded hover:bg-gray-300"  
 onClick={() => increaseQuantity(item.id)}  
 >  
 +  
 </button>  
 <button  
 className="px-2 py-1 text-sm bg-red-200 text-red-700 rounded hover:bg-red-300"  
 onClick={() => removeItem(item.id)}  
 >  
 x  
 </button>  
 </div>  
 </div>  
 </div>  
 </li>  
 ))}  
 </ul>  
 )}  
 {cart.length > 0 && (  
 <div className="mt-6 border-t pt-4 text-right">  
 <p className="text-lg font-bold">Összesen: {totalPrice} Ft</p>  
 </div>  
 )}  
</div>

# Állapotkezelés (useState)

A useState lehetővé teszi, hogy az alkalmazás emlékezzen a felhasználó tevékenységeire (pl. mit tett a kosárba), és ezek alapján automatikusan frissüljön a felület.

Minden új termék hozzáadásával a cart állapot tömbje bővül, a React pedig újrarendereli a kosarat.

# Felhasználói élmény

A felhasználó azonnal visszajelzést kap, amikor egy terméket hozzáad a kosárhoz. A kosár dinamikusan frissül, a komponensek újrarenderelődnek – nincs szükség manuális frissítésre.

Ez a modern felhasználói élmény, amit a hagyományos HTML oldalak nem nyújtanak ilyen szinten.

# Fájlok:

* App.jsx, main.jsx, Home.jsx, About.jsx, Contact.jsx
* Navbar.jsx, Footer.jsx
* vite.config.js, index.html

# React alkalmazás 2: Delivery Game

A projekt második React SPA alkalmazása egy játékos, interaktív élmény: a **„Food Delivery Game”** nevű ételszállító játék, amely egy **10x10-es rácson** zajlik. A felhasználó egy biciklis futárt irányít, aki **ételt szállít ki egy étteremből a megrendelőhöz**.

# A játék célja

* + Étel felvétele a konyhánál (0,0 pozíció)
  + Kiszállítás a címzetthez (9,9 pozíció)
  + Mozgás nyilakkal (↑ ↓ ← →)
  + Pontszám növelése minden sikeres kör után a távolság arányosan
  + Büntetés: letérés az útvonalról -1 pont (vizuális figyelmeztetéssel)
  + A pontok gyűjtésére 60 másodperc áll rendelkezésre

# Technológia

* + Állapotkezelés: useState
  + Eseménykezelés: useEffect
  + Megjelenítés: 10x10 grid, minden mező div
  + Fő kódrészlet – Mozgás kezelése

const handleKeyDown = (e) => {  
 setPosition((prev) => {  
 const newPos = { ...prev };  
 if (e.key === "ArrowUp" && prev.y > 0) newPos.y--;  
 if (e.key === "ArrowDown" && prev.y < gridSize - 1) newPos.y++;  
 if (e.key === "ArrowLeft" && prev.x > 0) newPos.x--;  
 if (e.key === "ArrowRight" && prev.x < gridSize - 1) newPos.x++;  
  
return newPos;  
 });  
};

*useEffect*(() => {  
 window.addEventListener("keydown", handleKeyDown);  
 return () => window.removeEventListener("keydown", handleKeyDown);  
}, [path, isDelivering]);

# Játékmenet logika – pickup és delivery

A játékmenetet egy második useEffect figyeli, ami az állapotváltozásra reagál:

*useEffect*(() => {  
 *// Pickup food (csak ha visszaér a konyhához)* if (  
 position.x === restaurant.x &&  
 position.y === restaurant.y &&  
 !hasFood &&  
 !isDelivering  
 ) {  
 setHasFood(true);  
 setIsDelivering(true); *// Most, hogy van étel, elindulhat a kiszállítás* const newDelivery = getRandomDeliveryPoint();  
 setDelivery(newDelivery); *// Új célpont* setPath(generatePath(restaurant, newDelivery)); *// Új útvonal a konyhától a házig* }  
  
 *// Deliver food* if (  
 isDelivering &&  
 hasFood &&  
 position.x === delivery.x &&  
 position.y === delivery.y  
 ) {  
 setHasFood(false); *// Ételt kézbesítette* setIsDelivering(false); *// Kiszállítás vége* setPath(new Set()); *// Az útvonal eltűnik* const distance = Math.abs(delivery.x - restaurant.x) + Math.abs(delivery.y - restaurant.y);  
 setScore((s) => s + distance); *// Pontszám növelése a távolsággal* setDelivery(null); *// Kiszállítás befejeződött, nincs ház* }  
}, [position, hasFood, isDelivering, delivery]);

A karakter kinézete is változik állapottól függően,   
 nincs nála étel: visszatérés az étterembe  
 van nála étel kiszállítás közben  
 letér az útvonalról kiszállítás közben

Megjelenítés – rács kirajzolása

const getCellContent = (x, y) => {  
 if (x === position.x && y === position.y) {  
 return (  
 <img  
 src={  
 !path.has(`${position.x},${position.y}`) && isDelivering  
 ? "images/delivery\_food\_red.png" *// ha rossz cellán vagyunk, piros ikon és még szállítunk* : hasFood  
 ? "images/delivery\_food.png" *// ha van étel* : "images/delivery\_empty.png" *// ha nincs étel (visszatértünk az étteremhez)* }  
 alt="Futár"  
 className="w-8 h-8"  
 />  
 );  
 } else if (x === restaurant.x && y === restaurant.y) {  
 return <img src="images/restaurant.png" alt="Restaurant" className="w-8 h-8" />;  
 } else if (delivery && x === delivery.x && y === delivery.y && isDelivering) {  
 return <img src="images/house.png" alt="House" className="w-8 h-8" />;  
 }  
 return null;  
};

**Felhasználói élmény**

Ez az alkalmazás játékos formában mutatja meg, hogyan lehet egy grid-alapú logikát felépíteni Reactben. A játék gyorsan reagál a billentyűleütésekre, a pontszám minden sikeres forduló után frissül, így a felhasználó azonnali visszajelzést kap.

# Fájlok:

* + App.jsx, main.jsx, App.css
  + Használ: useState, useEffect, függvényeket

# GitHub verziókezelés és közös munka

A projektet két fős csapat dolgozta ki, a GitHub platformon történő együttműködés segítségével. Mindenki külön fiókkal rendelkezik, és a projektet SmartKitchen néven hoztuk létre.

GitHub repo: <https://github.com/Orosznaszadi/SmartKitchen> A verziókezelés célja az volt, hogy:

* + külön fejlesztési lépéseket naplózzunk (legalább 5 commit/fő)
  + látható legyen, ki milyen részen dolgozott
  + könnyen tudjuk integrálni a két React alkalmazást

A commit üzenetek átláthatóak, szakaszonként kerültek feltöltésre.

# Nethelyes tárhelyre való publikálás

A projekt publikusan elérhető a következő címen:

<http://smartkitchen.nhely.hu/>

A feltöltés SFTP-n keresztül történt, a tárhelyhez a következő információkat használtuk:

* + **Cím:** ftp.nethely.hu:22
  + **Felhasználónév:** smartkitchen
  + **Jelszó:** BG74C4Bk9mC!8ta

# Munkamegosztás

A kétfős csapat tagjai egyenlő arányban vettek részt a munkában. A megosztás a következőképp alakult:

Orosz-Naszádi Krisztina:

* + CRUD táblázat, keresés, rendezés, kosár
  + Honlap teljes struktúrája (index.html, menu.html stb.)
  + HTML5 API példák
  + Chart.js diagram integráció
  + Dokumentáció írása és formázása

Bűdy Szabolcs:

* + AJAX API kapcsolódás és validáció
  + React Webshop fejlesztés
  + React Delivery játék létrehozása
  + GitHub technikai integráció
  + Webes publikálás, Nethely konfigurálás
  + Verziókezelés (merge, rebase, push/pull)
  + Dokumentáció írása és formázása

A munka közben állandó egyeztetés zajlott, a fejlesztés szorosan együttműködésben zajlott. Mindketten kb. azonos mennyiségű időt töltöttünk a fejlesztéssel.

# Tanulságok

Ez a projekt nem csak technikai kihívást jelentett, hanem kiváló lehetőséget arra is, hogy elmélyítsük tudásunkat:

* + hogyan épül fel egy modern, többfunkciós weboldal
  + miként használhatók fel a HTML5 új képességei
  + mik a React keretrendszer alapjai és előnyei
  + hogyan valósítható meg csapatmunka GitHub segítségével
  + milyen a gyakorlati projektstruktúra, dokumentáció, publikálás

# Összegzés

A *Smart Kitchen* projekt kiváló terepet biztosított számunkra arra, hogy bemutassuk és kipróbáljuk, amit megtanultunk a Web-programozás 1 tantárgy keretében.

Önállóan, mégis csapatmunkában dolgozva:

* + Komplex kliensoldali megoldásokat építettünk
  + Ügyeltünk az egységes dizájnra és UX-re
  + Strukturált és skálázható kódot írtunk
  + Éles környezetben teszteltük és publikáltuk

Bízunk benne, hogy munkánk eléri célját, és a dokumentáció alapján a fejlesztés átlátható, értékelhető lesz.

Köszönjük a lehetőséget! Orosz-Naszádi Krisztina Bűdy Szabolcs