

# Jegyzőkönyv

Web-technológiák

Féléves feladat

Recept Weboldal

Készítette: Orosz Péter  
Neptun kód: WO02D7  
Dátum: 2025. December

# Tartalomjegyzék

<b>1. Bevezetés</b>	<b>2</b>
<b>2. Projekt leírása</b>	<b>2</b>
2.1. Cél . . . . .	2
2.2. Funkciók . . . . .	2
2.3. Mappa- és fájlszerkezet . . . . .	2
<b>3. Használati útmutató</b>	<b>2</b>
<b>4. Műszaki megjegyzések</b>	<b>2</b>
4.1. Elérhetőség és szemantika . . . . .	2
4.2. Reszponzivitás . . . . .	3
4.3. Fejlesztési megjegyzések . . . . .	3
<b>5. Fontos kód részletek</b>	<b>3</b>
5.1. server.js — egyszerű Express szerver és API . . . . .	3
5.2. Kliens: recept beküldése (POST) . . . . .	3
5.3. Kliens: recept törlése (DELETE) . . . . .	4
5.4. Kliens: receptek betöltése (GET) . . . . .	4

# **1 Bevezetés**

Ez a jegyzőkönyv röviden bemutatja a projekt célját és felépítését. A projekt egy egyszerű receptoldal, amely lehetővé teszi receptek megtekintését, hozzáadását és eltávolítását.

## **2 Projekt leírása**

### **2.1 Cél**

A cél egy szemantikusan jól felépített, hozzáférhető és egyszerűen használható receptkezelő weboldal készítése, amely oktatási céllal készült a Web-technológiák kurzushoz.

### **2.2 Funkciók**

- Receptek megtekintése (lista, részletes nézet).
- Új recept beküldése (cím, hozzávalók, leírás).
- Receptek eltávolítása.
- Beágyazott oktatóvideó a főoldalon.

### **2.3 Mappa- és fájszerkezet**

Rövid áttekintés a fontos fájlok ról:

- index.html — kezdőoldal
- recipes.html — receptlista
- add-recipes.html — recept hozzáadása
- remove-recipe.html — recept eltávolítása
- styles/styles.css — alap stílusok
- resources/ — képek és egyéb erőforrások

## **3 Használati útmutató**

A weboldal használata egyszerű:

1. Indítsa el a szervert a projekt gyökérkönyvtárában: `node server.js`
2. A "View Recipes" oldalon böngészheti a meglévő recepteket.
3. Az "Add Recipe" oldalon töltön ki minden szükséges mezőt, majd küldje be a receptet.
4. A "Remove Recipe" oldalon törölheti a nem kívánt bejegyzéseket.

## **4 Műszaki megjegyzések**

### **4.1 Elérhetőség és szemantika**

A HTML szemantikai elemeket (header, nav, main, article, aside, footer) használjuk a jobb strukturáltságért és akadálymentességeért. A képekhez és videókhöz alternatív leírásokat és feliratokat adtunk meg ahol szükséges.

## 4.2 Reszponzivitás

A beágyazott videó és a képek rugalmasan skálázódnak, így a tartalom különböző képernyőméretekben is olvasható marad.

## 4.3 Fejlesztési megjegyzések

A projekt egyszerű, statikus fájlokra épül; további funkciók (pl. adatbázis, szerveroldali feldolgozás) szükség esetén hozzáadhatók.

## 5 Fontos kód részletek

Az alábbi rövid kód részletek a projekt működéséhez gyakran használt mintákat mutatják: egyszerű Node/Express szerver, kliens oldali recept beküldés és törlés fetch segítségével.

### 5.1 server.js — egyszerű Express szerver és API

```
const express = require('express');
const app = express();

app.use(express.json());
app.use(express.static('.')); // statikus fájlok kiszolgálása a projekt gyökérből

let recipes = [];  
// egyszerű in-memory tárolás

app.get('/api/recipes', (req, res) => {
  res.json(recipes);
});

app.post('/api/recipes', (req, res) => {
  const recipe = req.body;
  recipe.id = Date.now();
  recipes.push(recipe);
  res.status(201).json(recipe);
});

app.delete('/api/recipes/:id', (req, res) => {
  const id = Number(req.params.id);
  recipes = recipes.filter(r => r.id !== id);
  res.status(204).end();
});

app.listen(3000, () => console.log('Server running at http://localhost:3000'));
```

### 5.2 Kliens: recept beküldése (POST)

```
const form = document.querySelector('#addRecipeForm');
form.addEventListener('submit', async (e) => {
  e.preventDefault();
  const data = {
    title: form.title.value,
```

```

    ingredients: form.ingredients.value,
    instructions: form.instructions.value
};

const res = await fetch('/api/recipes', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify(data)
});
if (res.ok) window.location.href = '/recipes.html';
});

```

### 5.3 Kliens: recept törlése (DELETE)

```

async function removeRecipe(id) {
  const res = await fetch(`/api/recipes/${id}`, { method: 'DELETE' });
  if (res.ok) {
    const el = document.querySelector(`#recipe-${id}`);
    if (el) el.remove();
  }
}

```

### 5.4 Kliens: receptek betöltése (GET)

```

async function loadRecipes() {
  const res = await fetch('/api/recipes');
  const recipes = await res.json();
  // egyszerű render például:
  const list = document.querySelector('#recipesList');
  list.innerHTML = recipes.map(r => `
    <li id="recipe-${r.id}">
      <h3>${r.title}</h3>
      <button onclick="removeRecipe(${r.id})">Törlés</button>
    </li>
  `).join('');
  document.addEventListener('DOMContentLoaded', loadRecipes);
}

```

**Miért emeltem ki ezeket a kódrészleteket?** Az itt bemutatott példák lefedik az alkalmazás alapvető működését: a szerver (server.js) szolgálja ki a statikus fájlokat és biztosítja az API végpontokat, a kliensoldali POST művelet létrehozza az új recepteket, a DELETE kezeli a törlést és az UI frissítését, míg a GET felel a receptek lekéréséért és megjelenítéséért. Ezek megértése elengedhetetlen a hibakereséshez, a funkcionális bővítéséhez és a biztonsági/validációs rétegek hozzáadásához. A példák egyszerűsítettek, de könnyen kiterjeszthetők tartós tárolásra (adatbázis), autentikációra és részletes hibakezelésre.