Web Programming Project Report

Name: Szabó Lilla Dorottya

Neptun Code: BDCRL0

Date: 2025.04.18.

GitHub Repository: BDCRL0/WebPrograming-Lecture: Web Programing Lecture Homework

Live Demo: http://webprog10.nhely.hu

Table of Contents

Introduction

This project was created as part of the Web Programming 1 course and demonstrates the application of modern front-end web development concepts. The goal was to design a multi-page website that integrates several core technologies, each showcasing different aspects of web programming.

The site includes both static and dynamic pages, features object-oriented programming using JavaScript, and performs real-time data operations via AJAX. One page also functions as a Single Page Application (SPA) built with React.

Throughout the project, the following technologies were used:

- **HTML5** for semantic structure and modern input types
- **CSS3** for responsive design and visual styling with a unified blue-black-orange theme
- **Vanilla JavaScript** to handle dynamic interactivity and DOM manipulation
- **Chart.js** to render responsive, interactive graphs
- **AJAX** to communicate with a backend API for Create, Read, Update, and Delete (CRUD) operations
- **React** to build an interactive, client-side SPA with component-based architecture

In addition to fulfilling the assignment requirements, several enhancements were made, such as row modification in the table, duplicate entry detection, client-side validation, click-to-remove features, and responsive design improvements.

This document serves as a summary of the project's structure and features, supported by screenshots and testing documentation.

Home Page

The Home Page serves as the central navigation hub of the project. It provides users with an intuitive menu bar that links to each of the functional sections: Table, HTML5 Features, Charts, AJAX, Object-Oriented JavaScript (OOJS), and the React Single Page Application (SPA).

The layout follows a consistent visual theme using shades of blue, black, and orange for a modern and clean appearance. The navigation menu is fixed at the top for easy access across all devices and resolutions.

The Home Page also includes a brief welcome message, encouraging users to explore the project and select a section of interest. Each link in the menu corresponds to a different technology or concept showcased in the course.

Features:

- Simple and clean design
- Clearly labeled navigation menu
- Responsive layout
- Footer with author information and Neptun code

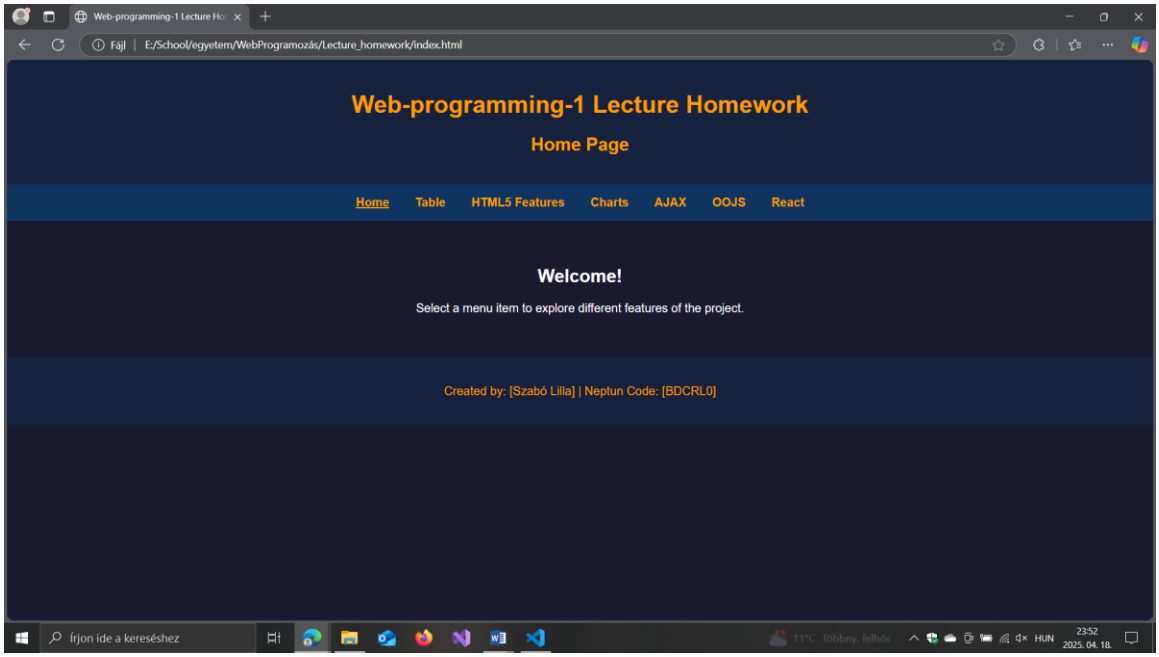


Table Page

The Table Page demonstrates dynamic content creation and manipulation using vanilla JavaScript. It features a custom-built table that supports **CRUD operations** — Create, Read, Update, and Delete — entirely on the client side.

Users can add new entries by filling in the input fields for name, age, city, and country. As a usability enhancement, the system checks for **duplicate rows** — if a new entry matches an existing one exactly, a confirmation dialog prompts the user to decide whether to continue.

Each row in the table includes **action buttons**:

- **Modify:** Replaces cell values with input fields, allowing in-place editing and saving changes.

- **Delete:** Instantly removes the row from the table.

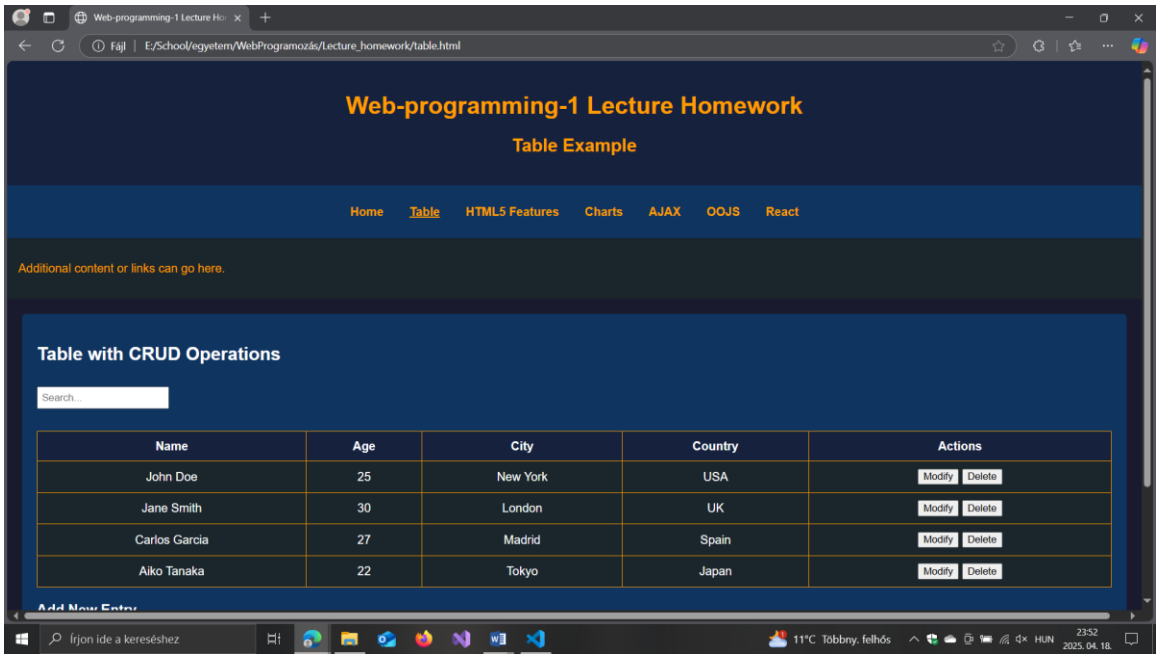
Sorting and searching functionality is also included:

- Clicking on column headers sorts the table alphabetically/numerically.
- A search field filters visible rows based on user input.

The layout ensures all form controls remain visible above the footer, avoiding overlap and improving user experience.

Features:

- Full Create/Read/Update/Delete support
- Row modification with inline editing
- Duplicate entry warning
- Search and sort by column
- Accessible layout with responsive design



HTML5 Page

The HTML5 Page showcases the use of semantic HTML elements, enhanced form input types, and browser-based data storage using `localStorage`.

The page is structured using HTML5 semantic tags like `<header>`, `<nav>`, `<section>`, `<article>`, and `<footer>`, demonstrating proper content organization and accessibility best practices.

A sample form is included that features modern HTML5 input types:

- **Email input** for basic validation
- **Date picker** for selecting dates
- **Range slider** for numeric input

It includes semantic elements (`<article>`, `<aside>`), enhanced form inputs, and working examples of:

- Local Storage
- Canvas drawing
- SVG shapes
- Web Workers
- Geolocation
- Drag and Drop API

  The **Drag and Drop example** now features a smooth **back-and-forth** interaction, allowing the user to move an image between two zones repeatedly. This simulates a more realistic drag-and-drop interface by maintaining draggable behavior and styling consistently.

In addition, a **localStorage example** is implemented. Users can input a custom value into a text field, which will be stored in the browser's local storage. On reload, the stored value is automatically retrieved and displayed, proving data persistence without needing a server.

Features:

- Semantic HTML layout
- Input types: email, date, and range
- Persistent data using localStorage

- User-friendly, interactive elements

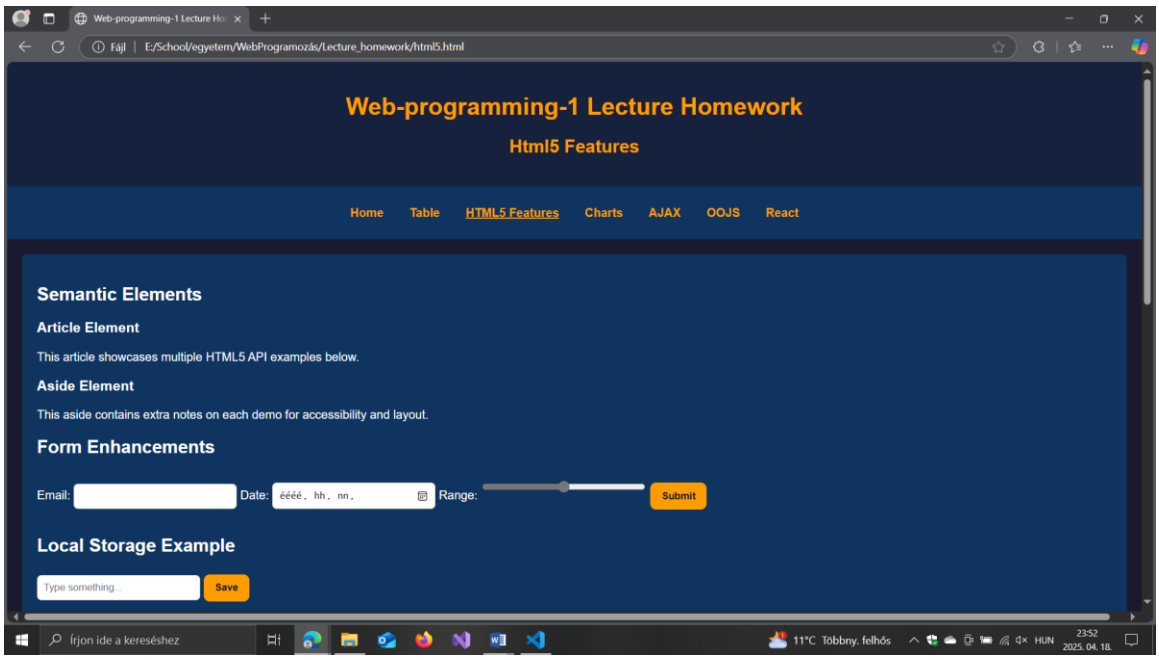


Chart Page

The Chart Page presents an interactive 5×5 table filled with random numeric values. This feature demonstrates how JavaScript and Chart.js can be used together to visualize data dynamically.

Each row in the table is clickable. When a user selects a row, the values from that row are extracted and rendered in a **responsive line chart** directly below the table using **Chart.js**. The chart displays:

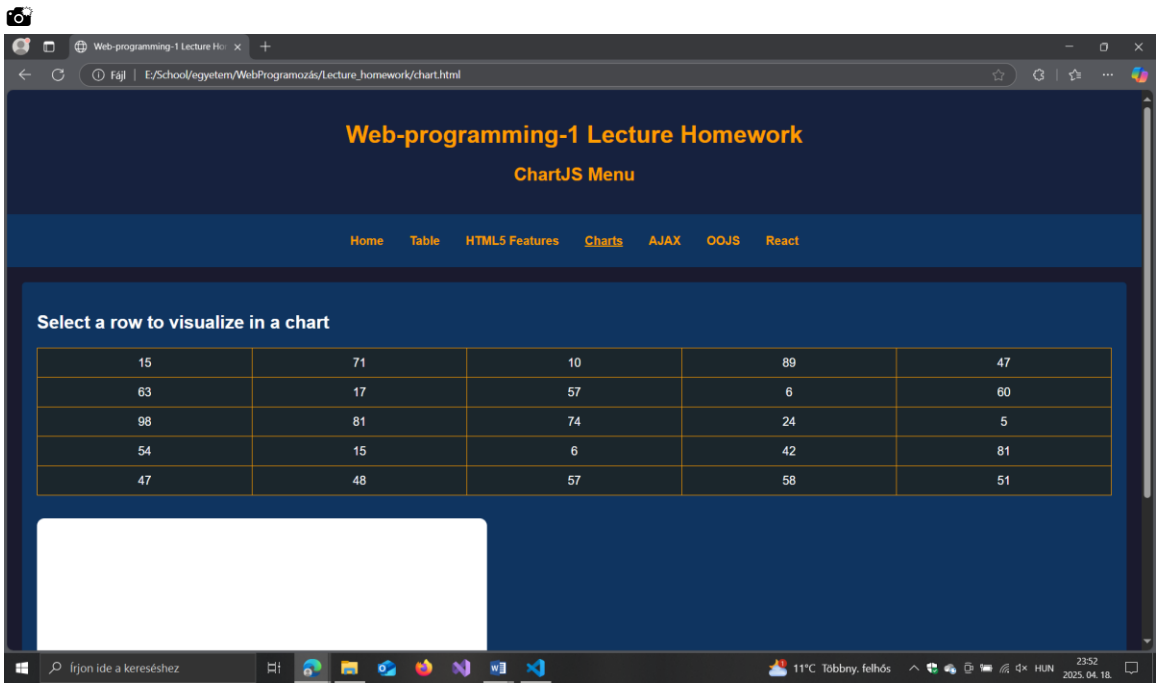
- Labels for each column (e.g., “Column 1”, “Column 2”, etc.)
- Smooth line connections
- Point markers and hover tooltips showing the exact values

The canvas area for the chart is styled to match the table’s width, providing visual balance and enhancing usability. This makes the chart feel like a natural extension of the table.

Features:

- Interactive 5×5 numeric table

- Dynamic chart rendering with Chart.js
- Data values displayed in tooltips
- Responsive layout with aligned canvas width



AJAX Page

The AJAX Page allows users to perform real-time data operations with a backend API using `fetch()` and the `POST` method. It includes a form with inputs for name, height, weight, and ID, and supports the full range of **CRUD operations**:

- **Create:** Add a new record to the database.
- **Read:** Fetch all records associated with the user's code and display a summary with total, average, and max height.
- **Update:** Modify a specific record by providing its ID and new values.
- **Delete:** Remove a record using its ID.

Input Validation:

To ensure data integrity, the app checks that:

- All fields are filled in
- Name is no longer than 30 characters

If any input is invalid, a friendly error message is shown before any request is sent.

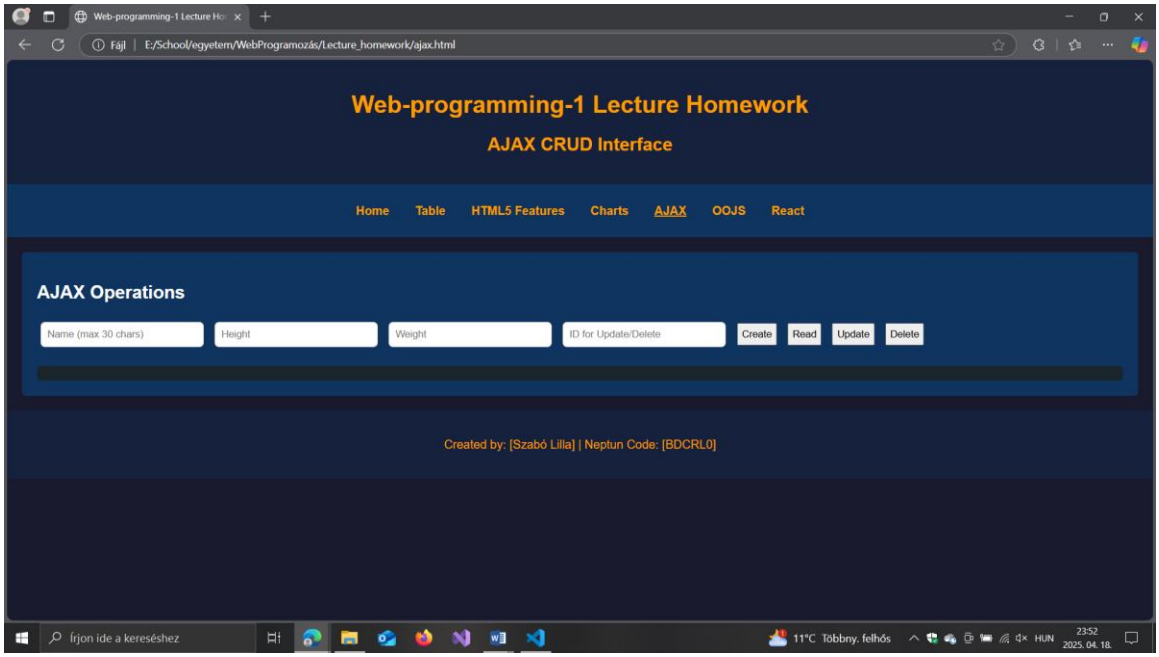
API Communication:

Each operation sends a properly formatted URL-encoded body with a unique `code` used for authentication (based on the user's Neptun code). API responses are displayed in a text area below the form.

Features:

- Full CRUD using AJAX with live feedback
- Form validation for clean data input
- Summary stats (count, sum, average, max)
- Uses `fetch()` and `application/x-www-form-urlencoded`

📸



OOJS Page

The Object-Oriented JavaScript (OOJS) Page demonstrates the use of `class`, `constructor`, and inheritance to dynamically create and manage interactive elements — specifically, **colored boxes** rendered on the page.

Each box is created from a class-based system, with a base `Box` class and a subclass `AnimatedBox` that adds mouse-hover animations. Clicking the **"Add Color Box"** button creates a new box with a random color, and unique ID.

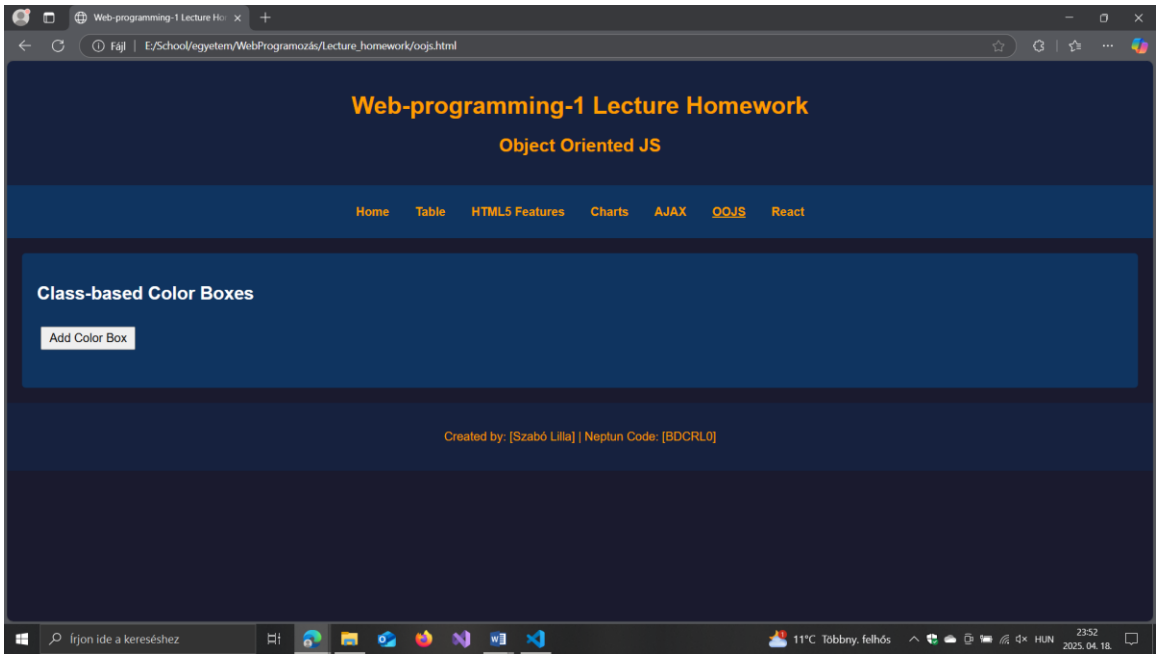
Boxes are arranged in a flexible grid layout using CSS Flexbox. They are fully interactive — users can simply **click a box to remove it**. This triggers a class method that updates the DOM in real time, showcasing encapsulation and DOM manipulation via object methods.

This page was gradually enhanced with features like:

- Auto-generated IDs
- Hover animations
- Click-to-remove
- Simplified layout to focus on OOP

Features:

- ES6 Classes with `constructor`, `extends`, and `super()`
- Dynamic DOM element creation (`appendChild`)
- Interactive animation using inline event handlers
- Clean UI with auto-reflow when a box is removed



React SPA

The React SPA (Single Page Application) demonstrates the use of dynamic, component-based interfaces within a single HTML page. This section is fully interactive, built using **React 17 (via CDN)** and **Babel** to enable JSX.

The React Single Page Application (SPA) includes three mini tools: a Calculator, a Counter, and a fully functional **Tic Tac Toe** game. Each of these components demonstrates state management and interactive UI building using React.

Calculator

- Users input two numbers.
- The sum is calculated and displayed instantly using `useState`.

Counter

- Displays a number that the user can increment or decrement.
- Includes a **Reset** button to set the value back to 0.
- Uses `useState` for state management.
- Styled and centered for clarity and responsiveness.

Tic tac toe

- uses a 3x3 board
- keeps track of the current player
- visually updates the board on each move
- checks for a winner
- includes a restart button

This interactive game showcases conditional rendering and basic game logic handled entirely with React hooks.

The app is navigated through a simple tab-like menu, and only the active component is rendered. The React logic is embedded in a `<script type="text/babel">` block and powered by ReactDOM's `render()` function.

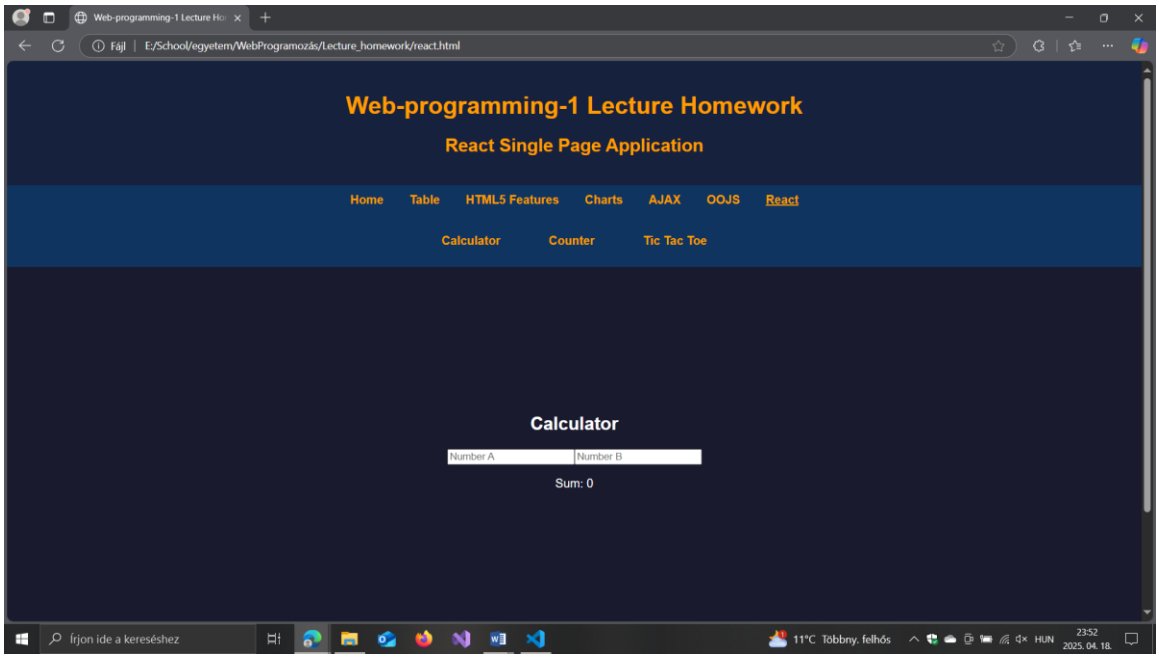
A link is also provided to return to the main site navigation from within the SPA, ensuring consistent user flow.

Features:

- Built with React and Babel (CDN-based)

- Functional components using React hooks (`useState`)
- SPA-like navigation between views
- Clean layout and modern styling
- Accessible return to the main menu

Find the codes of the Calculator and Counter at the end of the PDF.



Version Control and Collaboration

This project was managed using **Git** and hosted on **GitHub** for version control. Throughout the development process, Git was used to track changes, manage file structure, and commit progress incrementally.

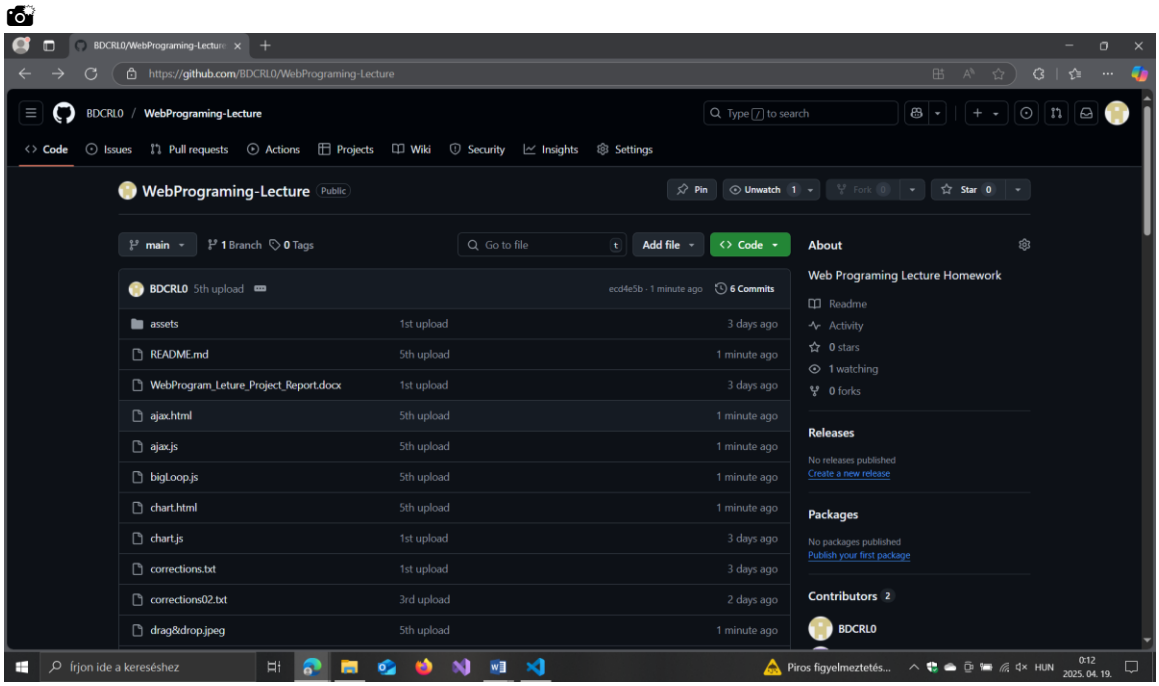
Each functional feature — including AJAX integration, the React SPA, dynamic charts, and object-oriented JS — was implemented in dedicated HTML and JS files and committed with clear, descriptive messages. This ensured a clean and traceable development history.

Although the project was completed individually, GitHub was used as if in a collaborative environment:

- Regular commits documented each major milestone and feature addition
- The repository structure was kept modular and organized

- All files were maintained in a consistent coding style and naming convention

This workflow ensured better project structure, ease of review, and readiness for deployment.



API Testing

To verify the correctness and reliability of the AJAX-based CRUD functionality, the project's API endpoints were tested using both the **web interface** and external tools like **Postman**.

Postman was used to manually construct and send HTTP POST requests to the following endpoint: http://gamf.nhely.hu/ajax2/

Each request included:

- Operation (`op`) as a form field: `create`, `read`, `update`, or `delete`
- User-provided data: `name`, `height`, `weight`, and `id` (when needed)
- Unique `code` associated with the project instance

Tests Performed:

- Created multiple records with valid and invalid data

- Read records and verified the response content
- Updated records using specific IDs
- Deleted records and confirmed successful deletion
- Tested numeric validation for height and weight
- Confirmed API responses displayed correctly in the browser

These tests helped ensure that all validation logic, form handling, and fetch interactions in the JavaScript code function as intended.

Project Links

GitHub Repository: BDCRL0/WebPrograming-Lecture: Web Programing Lecture Homework

Live Demo: http://webprog10.nhely.hu

Calculator code:

```
<script type="text/babel">
  const { useState } = React;

  function Calculator() {
    const [a, setA] = useState(0);
    const [b, setB] = useState(0);
    return (
      <div style={{ textAlign: 'center' }}>
        <h2>Calculator</h2>
        <input type="number" onChange={e => setA(+e.target.value)} placeholder="Number A" />
        <input type="number" onChange={e => setB(+e.target.value)} placeholder="Number B" />
        <p>Sum: {a + b}</p>
      </div>
    );
```

Counter code:

```
155
156     function Counter() {
157       const [count, setCount] = useState(0);
158       return (
159         <div className="counter">
160           <h2>Counter</h2>
161           <p>{count}</p>
162           <button onClick={() => setCount(count + 1)}>+</button>
163           <button onClick={() => setCount(count - 1)}>-</button>
164           <button onClick={() => setCount(0)}>Reset</button>
165         </div>
166       );
167     }
```