

Web Programming I

Lecture Assignment Documentation

Prepared by:

Name: Szabó Lilla

Neptun Code: BDCRL0

GitHub Project URL: [BDCRL0/WebPrograming-Lecture: Web Programing Lecture Homework](https://github.com/BDCRL0/WebPrograming-Lecture: Web Programing Lecture Homework)

Access data for verification

- Website URL address: <http://webprog10.nhely.hu>
-

Task 1

Introduction

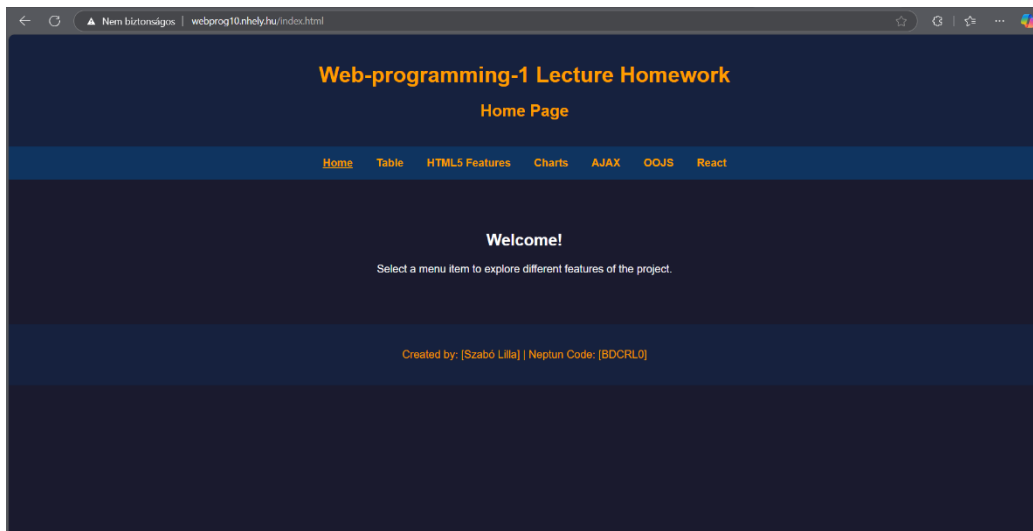
This documentation presents the *Web Programming I Lecture Assignment* project. The goal of the assignment was to develop a multi-page web application using various frontend technologies. The system includes CRUD functionality in a table, HTML5 API demonstrations, data visualization using Chart.js, AJAX communication, object-oriented JavaScript (OOJS), and a React-based Single Page Application (SPA).

Page Structure and Home Page

The project uses a consistent structure across all pages. The individual HTML sections are as follows:

- `<header>` – Contains the site title
- `<nav>` – Horizontal navigation bar
- `<aside>` – Informational sidebar on selected pages
- `<div class="content">` – Main content section (varies by page)
- `<footer>` – Displays author name and Neptun code

The main page (index.html) contains the page title and a welcome text.

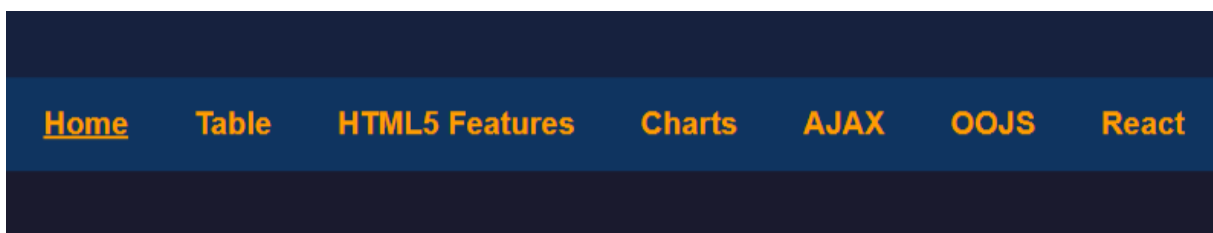


Navigation Menu

The navigation is displayed horizontally under the `nav` element. The menu items point to the following pages:

- Home (index.html)
- Table (table.html)
- HTML5 Features (html5.html)
- Charts (chart.html)
- AJAX (ajax.html)
- OOJS (oojs.html)
- React (react.html)

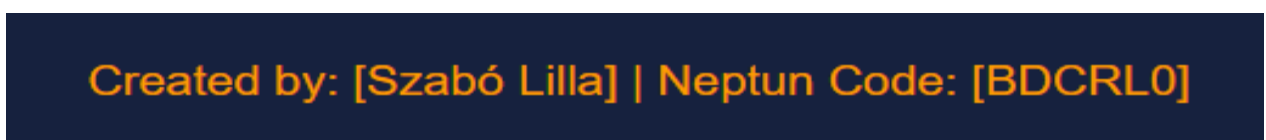
The active page is highlighted using the `class="active"`, which is highlighted by CSS styling.



Footer

The footer appears on every page and contains the creator's information:

„Created by: [Szabó Lilla] | Neptun Code: [BDCRL0]”



Styling

The CSS (embedded in each HTML page) defines the visual appearance of the project. It handles the following in particular:

- Menu style, active element highlighting
 - Layout responsiveness
 - Header, aside, content and footer layout
 - Fonts, margins, paddings
 - The theme features dark backgrounds with an accent color of orange (#ff9d00).
-

CRUD Table (table.html + table.js)

I implemented the CRUD (Create, Read, Update, Delete) functions on a table of at least 4x4 in the table.html and table.js files. The following functions were implemented:

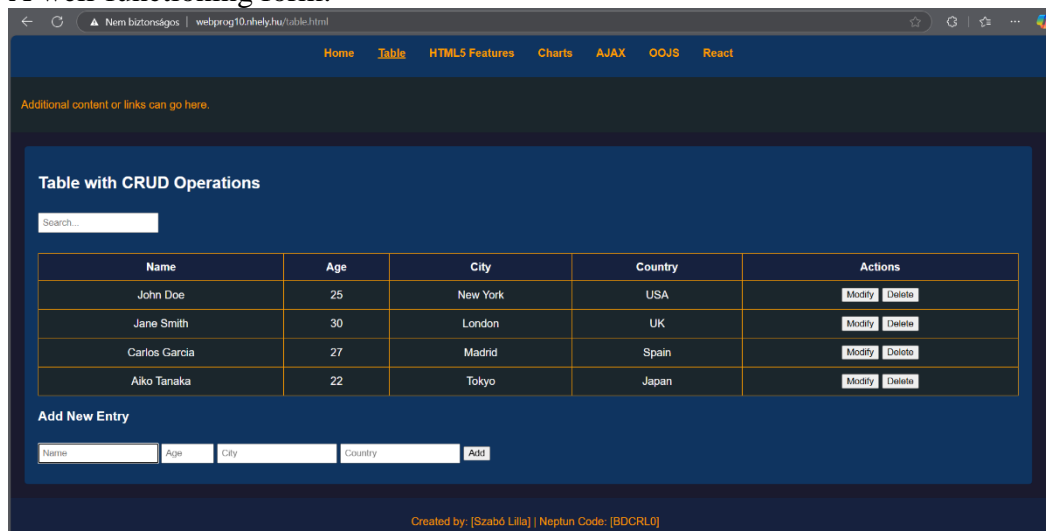
1. Form for creating new data

The form field contains four input fields: Name, Age, City, Country. With the "Add" button a new row can be created in the table.

The program will throw error messages if the filling does not meet the following conditions:

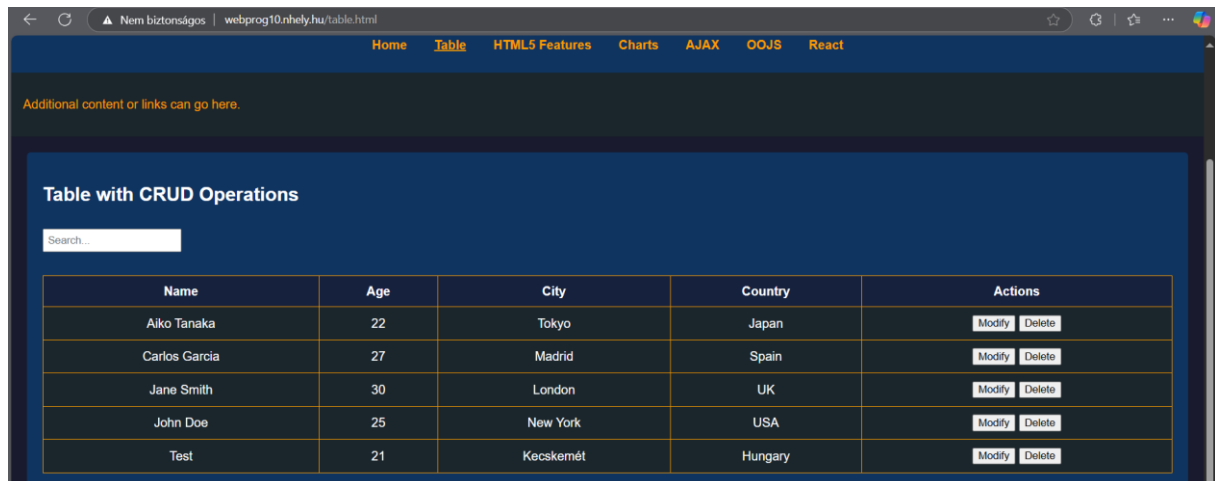
- Required fields
- Minimum: 2 characters
- Maximum: 30 characters
- The data is validated by JavaScript (table.js)

A well-functioning form:



Minimum character error:

The entered data is displayed in a dynamic table. Each row has two action buttons: "Modify" and "Delete".



Additional content or links can go here.

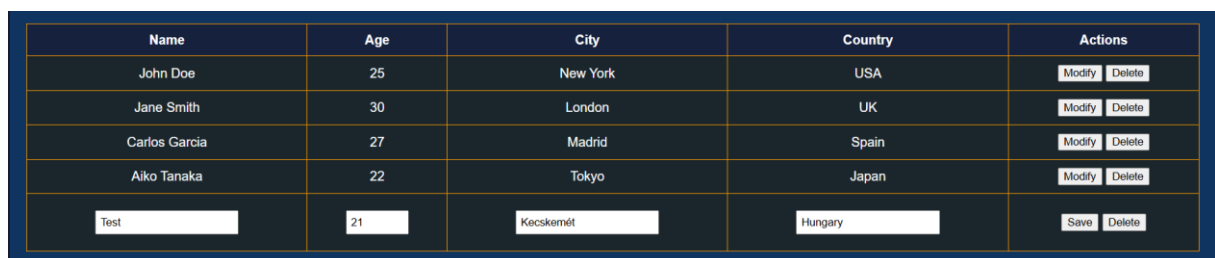
Table with CRUD Operations

Search...

| Name | Age | City | Country | Actions |
|---------------|-----|-----------|---------|---|
| Aiko Tanaka | 22 | Tokyo | Japan | <button>Modify</button> <button>Delete</button> |
| Carlos Garcia | 27 | Madrid | Spain | <button>Modify</button> <button>Delete</button> |
| Jane Smith | 30 | London | UK | <button>Modify</button> <button>Delete</button> |
| John Doe | 25 | New York | USA | <button>Modify</button> <button>Delete</button> |
| Test | 21 | Kecskemét | Hungary | <button>Modify</button> <button>Delete</button> |

3. Modify data (Update)

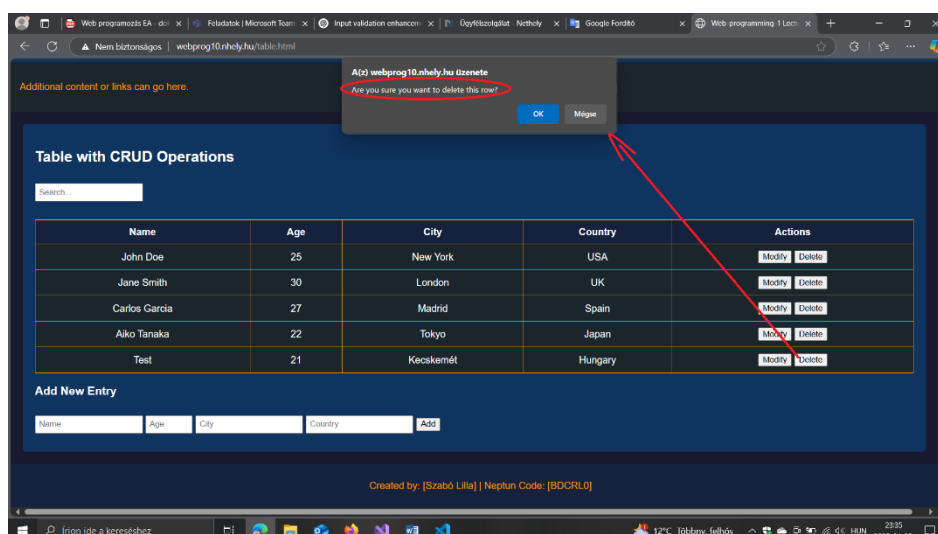
By clicking the "Edit" button, the data of the given row becomes editable. The save button updates the given row.



| Name | Age | City | Country | Actions |
|-----------------------------------|---------------------------------|--|--------------------------------------|---|
| John Doe | 25 | New York | USA | <button>Modify</button> <button>Delete</button> |
| Jane Smith | 30 | London | UK | <button>Modify</button> <button>Delete</button> |
| Carlos Garcia | 27 | Madrid | Spain | <button>Modify</button> <button>Delete</button> |
| Aiko Tanaka | 22 | Tokyo | Japan | <button>Modify</button> <button>Delete</button> |
| <input type="text" value="Test"/> | <input type="text" value="21"/> | <input type="text" value="Kecskemét"/> | <input type="text" value="Hungary"/> | <button>Save</button> <button>Delete</button> |

4. Delete row (Delete)

By clicking the "Delete" button, the user will receive a confirmation. If accepted, the row will be deleted from the table.



5. Search / Filter

The search field above the table allows you to filter the results based on the value of any column.

Table with CRUD Operations

| Name | Age | City | Country | Actions |
|------------|-----|--------|---------|---|
| Jane Smith | 30 | London | UK | <button>Modify</button> <button>Delete</button> |

6. Sort by columns

The column headers are clickable, so you can sort the table by any column. Sorting is done by textual comparison (`localeCompare`).

Table with CRUD Operations

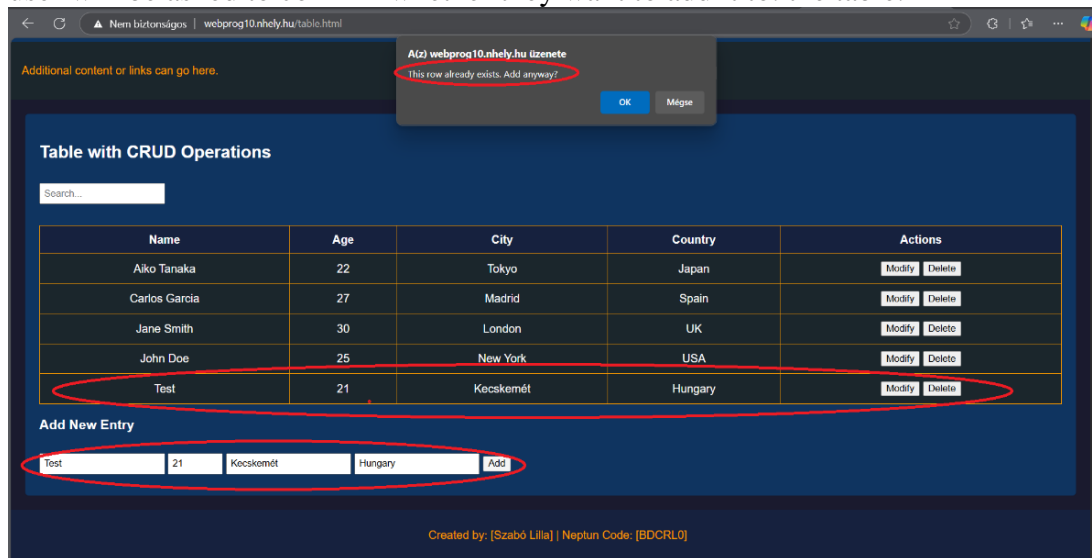
| Name | Age | City | Country | Actions |
|---------------|-----|----------|---------|---|
| Aiko Tanaka | 22 | Tokyo | Japan | <button>Modify</button> <button>Delete</button> |
| John Doe | 25 | New York | USA | <button>Modify</button> <button>Delete</button> |
| Carlos Garcia | 27 | Madrid | Spain | <button>Modify</button> <button>Delete</button> |
| Jane Smith | 30 | London | UK | <button>Modify</button> <button>Delete</button> |

Table with CRUD Operations

| Name | Age | City | Country | Actions |
|---------------|-----|----------|---------|---|
| Aiko Tanaka | 22 | Tokyo | Japan | <button>Modify</button> <button>Delete</button> |
| Carlos Garcia | 27 | Madrid | Spain | <button>Modify</button> <button>Delete</button> |
| Jane Smith | 30 | London | UK | <button>Modify</button> <button>Delete</button> |
| John Doe | 25 | New York | USA | <button>Modify</button> <button>Delete</button> |

7. Duplicate check

If every element of a form the user wants to add matches every part of an existing row, the user will be asked to confirm whether they want to add it to the table.



Technologies implemented in this section:

HTML: Structured structure, use of forms, tables

CSS: for uniform appearance

JavaScript: Full functionality:

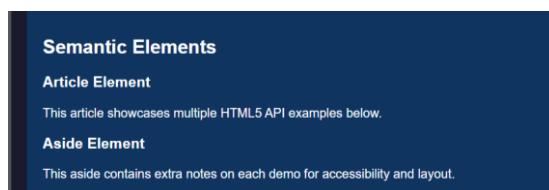
- Validation
- CRUD operations
- Search
- Sorting
- Dynamic DOM handling (appendChild, innerHTML)

HTML5 API Demonstrations (html5.html + html5.js)

Under the HTML5 API menu item, I demonstrated various modern web functions that help me illustrate the possibilities offered by HTML5. The sub-chapters show in detail how I implemented these functions.

1. Article & Aside

This page contains the `article` and the `aside` element.



2. Form Enhancements

Modern input types (email, date, range)

Form Enhancements

Email: Date: Range:

3. Local Storage

This feature allows the page to store data locally in the user's browser. Using the `localStorage` object, we can store a text input, which we can display even after reloading.

Local Storage Example

Saved: TEST

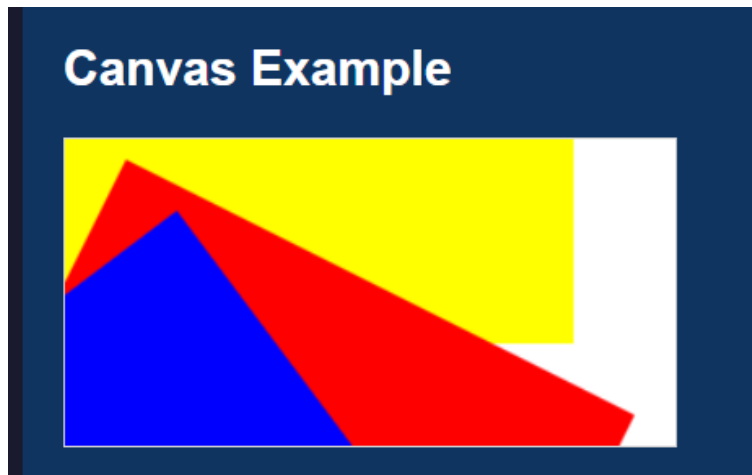
4. Drag and Drop API

In a simple example, a user can drag an image to a designated target area and then back



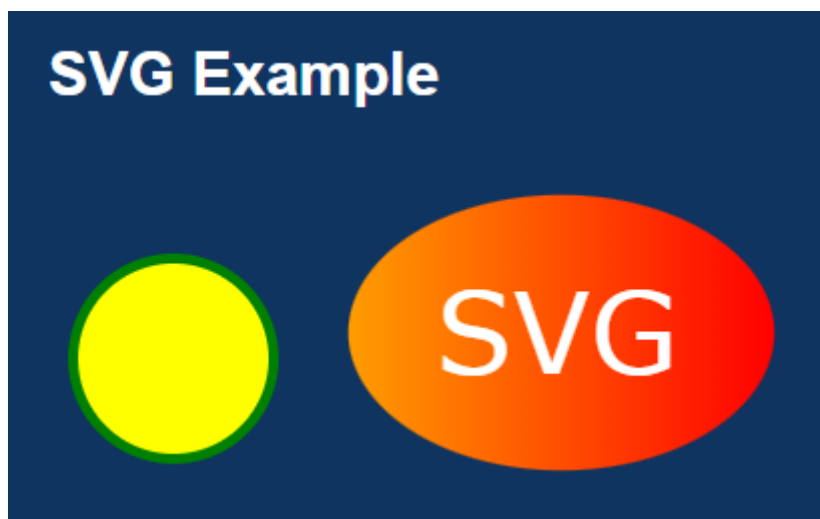
5. Canvas

The canvas element allows you to draw visual content. Here I have displayed an image consisting of several rotated rectangles using JavaScript.



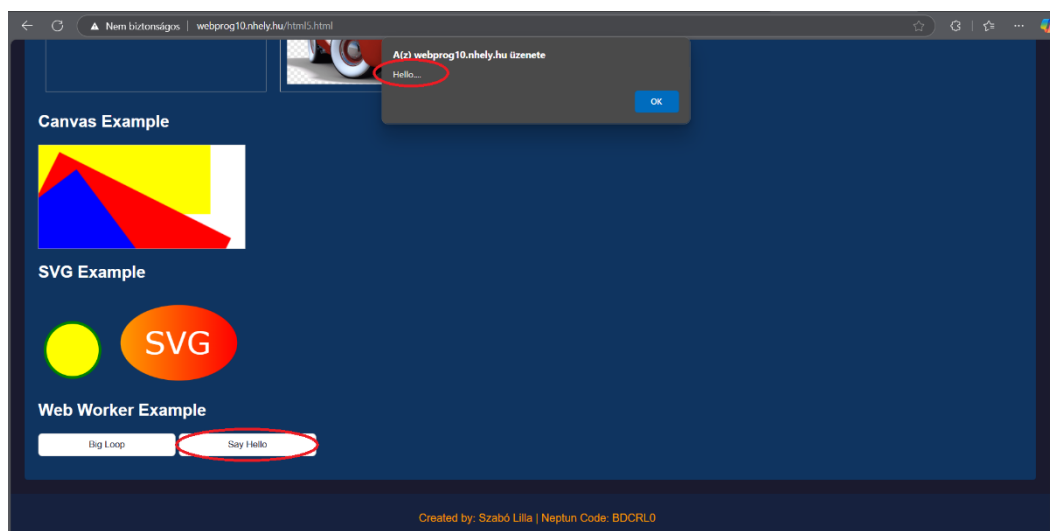
6. SVG

The SVG element displays vector graphics. In this case, I drew a yellow circle and an ellipse with a transition from yellow to red, with the svg label in the middle.



7. Web Worker

With the Web Worker, we can perform parallel calculations without interrupting the main thread.



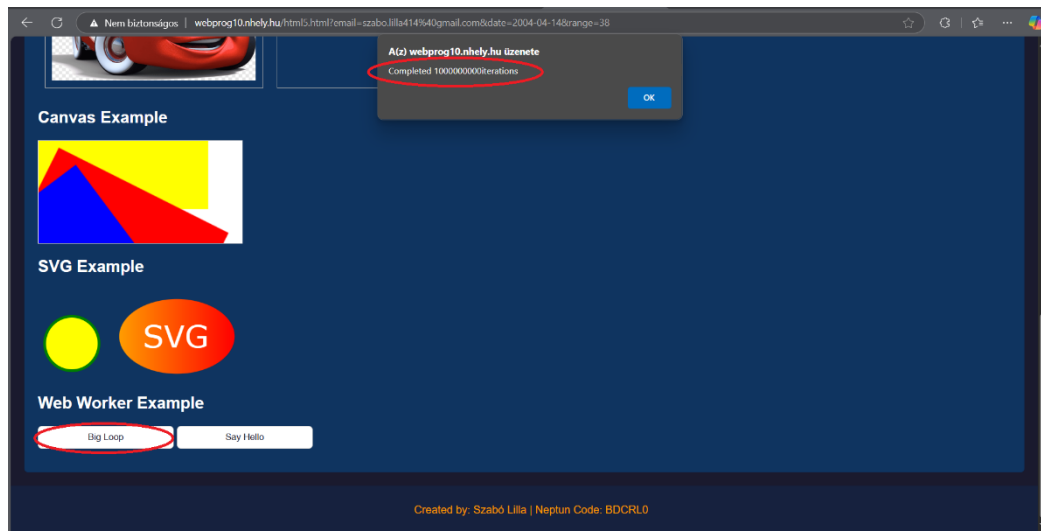


Chart.js Integration (chart.html + chart.js)

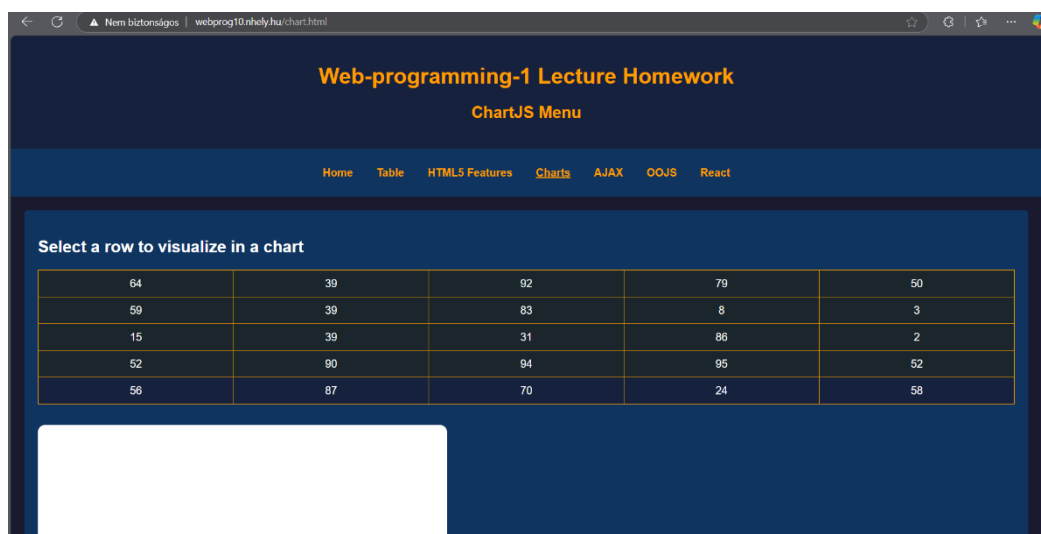
In this menu item, I displayed a 5×5 table from which the user can select any row. I display the values of the selected row as a line chart using the `Chart.js` library.

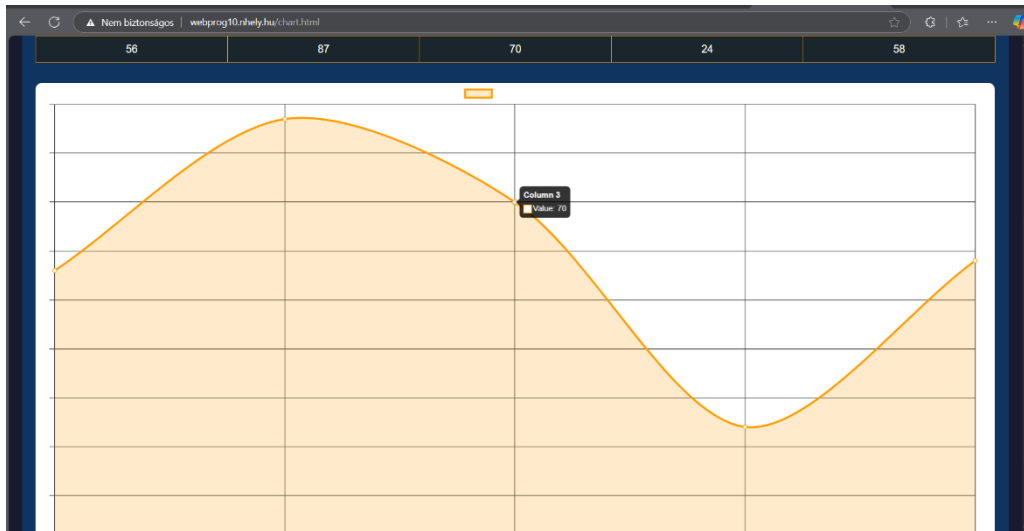
Technologies used:

- **Chart.js:** a modern JavaScript library that creates spectacular, responsive charts.
- **HTML table:** To display static data
- **Event handling in JavaScript:** To implement clickability of the rows and dynamic chart updates.
- **Tooltip:** the user can see the current value by moving the mouse to the appropriate point on the chart

Description of operation:

By clicking on each row in the table, a new line chart is generated based on the row's values. In the `chart.js` file, the `drawChart()` function creates the corresponding chart using the data of the selected row.





AJAX CRUD Operations (ajax.html + ajax.js)

In this menu item, I implemented a complete CRUD application using the `fetch()` function, connected to the API of the server `http://gamf.nhely.hu/ajax2/`. The data represents a fictitious database, where name, height and weight fields can be handled.

Technology used:

- **AJAX** (fetch API): communication with the server in the background
- **JavaScript DOM operations**: data retrieval, form handling
- **HTML form elements**: for implementing inputs and interactions

Technical note:

- AJAX operations are done with POST requests
- Responses arrive in JSON format and dynamically update the DOM.
- We use the code parameter to identify our own data on the backend page

Functions:

1.READ – Retrieve data

- Queries the user's data at the press of a button
- Displays them and also calculates statistics: total records

2.CREATE

- A new entry can be created based on name, height, and weight
- Validation is also performed (e.g. name length, empty fields)
- If successful, the list is reloaded

AJAX Operations

Lilla 200 60 ID for Update/Delete **Create** Read Update Delete

Create Response: 1

Nem biztonságos | webprog10.nhely.hu/ajax.html

**Web-
network**

Home Table HTML5 Features Charts **AJAX** OOJS React

AJAX Operations

John 100 Weight ID for Update/Delete Create Read Update Delete

Create Response: 1

3. UPDATE

- Queries the server for existing data based on a given ID and fills in the form
- After modification, sends the new data to the server
- If no match is found, reports an error

AJAX Operations

Name (max 30 chars) Height Weight ID for Update/Delete Create Read Update Delete

ID: 983, Name: test, Height: test, Weight: test ID: 1001, Name: TEST, Height: t, Weight: t ID: 1441, Name: Lilla, Height: 165, Weight: 65 ID: 1488, Name: Lilla, Height: 200, Weight: 60 Total Records: 4

AJAX Operations

TEST test test 1001 Create Read **Update** Delete

ID: 983, Name: test, Height: test, Weight: test ID: 1001, Name: TEST, Height: t, Weight: t ID: 1441, Name: Lilla, Height: 165, Weight: 65 ID: 1488, Name: Lilla, Height: 200, Weight: 60 Total Records: 4

AJAX Operations

TEST test test 1001 Create Read Update Delete

Update Response: 1

AJAX Operations

TEST test test 1001 Create Read Update Delete

ID: 983, Name: test, Height: test, Weight: test ID: 1001, Name: TEST, Height: test, Weight: test ID: 1441, Name: Lilla, Height: 165, Weight: 65 ID: 1488, Name: Lilla, Height: 200, Weight: 60 Total Records: 4

Error response:

AJAX Operations

ErrorTest 100 100 2000 Create Read **Update** Delete

Update Response: 0

4.DELETE

- The selected record can be deleted based on an ID
- The page immediately refreshes the list after a successful deletion

The screenshot shows a dark blue header with the text "AJAX Operations". Below the header is a form with four input fields: "Name (max 30 chars)", "Height", "Weight", and "ID". The "ID" field contains the value "1001". To the right of the input fields are four buttons: "Create", "Read", "Update", and "Delete". The "Delete" button is circled in red. Below the form is a dark blue bar with the text "Delete Response: 1", which is also circled in red.

OOJS – Object-Oriented JavaScript (oojs.html + oojs.js)

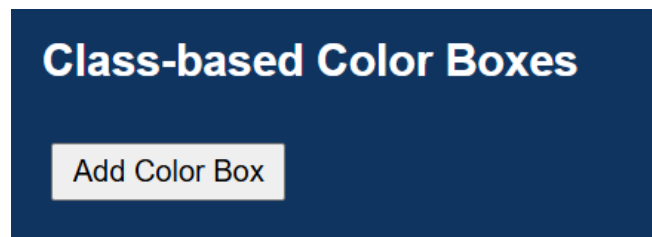
In this menu item, I will demonstrate the application of object-oriented programming principles in JavaScript through a Class-based Color Boxes example.

Technologies used:

- JavaScript classes – class, constructor, extends, super
- DOM manipulation – adding new elements
- Event handling – creating and displaying instances by clicking buttons

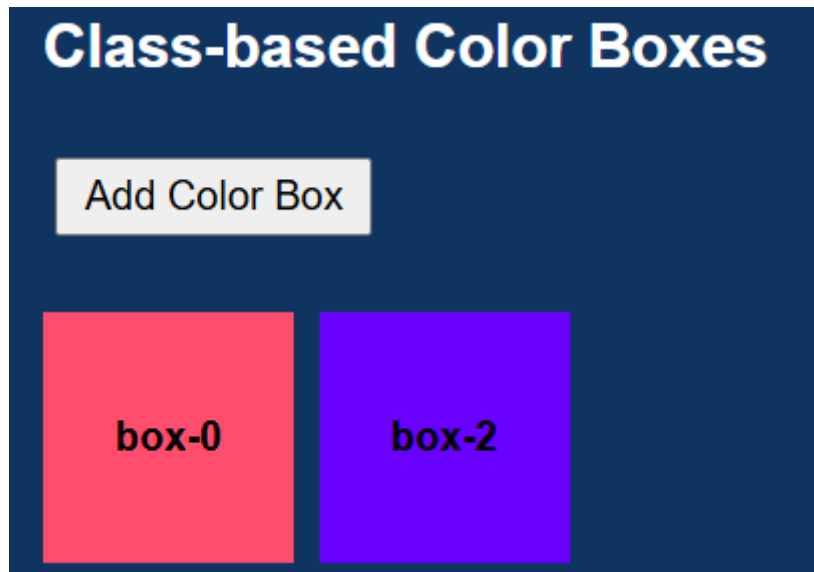
Example in operation:

Users can add colorful boxes by clicking a button. Each box is an instance of a class, using inheritance and animation effects. Then users can remove any of the boxes just by clicking on it.



The cursor is standing on box-1:





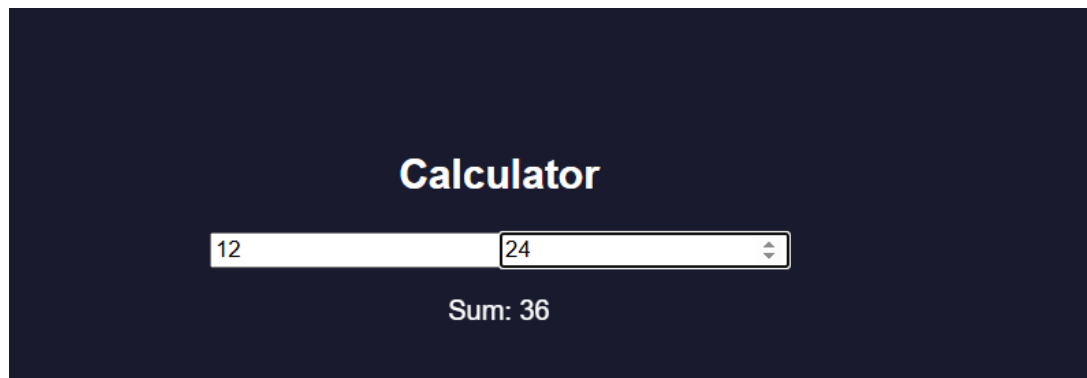
React SPA (react.html)

The task was to create a React-based single-page application (SPA) that contains at least two separate games as menu items. The app uses `useState` hooks, a component-based approach, and JavaScript-based page switching.

The React-based single page app includes 3 games.

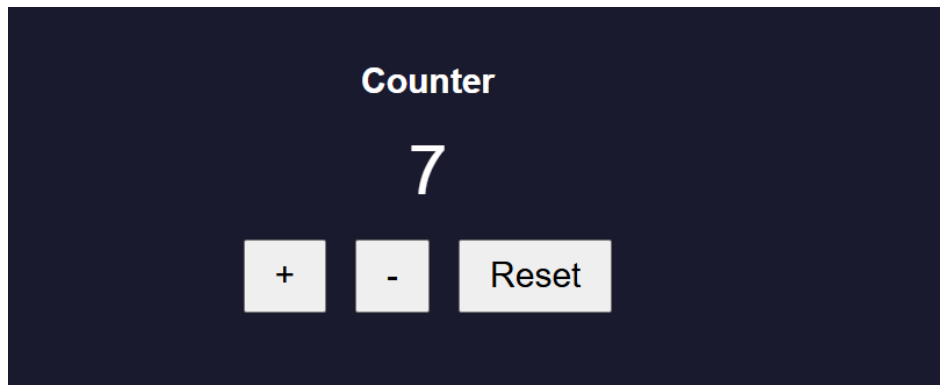
Calculator

As soon as we enter the react page, the calculator appears. Here we have the option to add two numbers (it can be negative or positive).



Counter

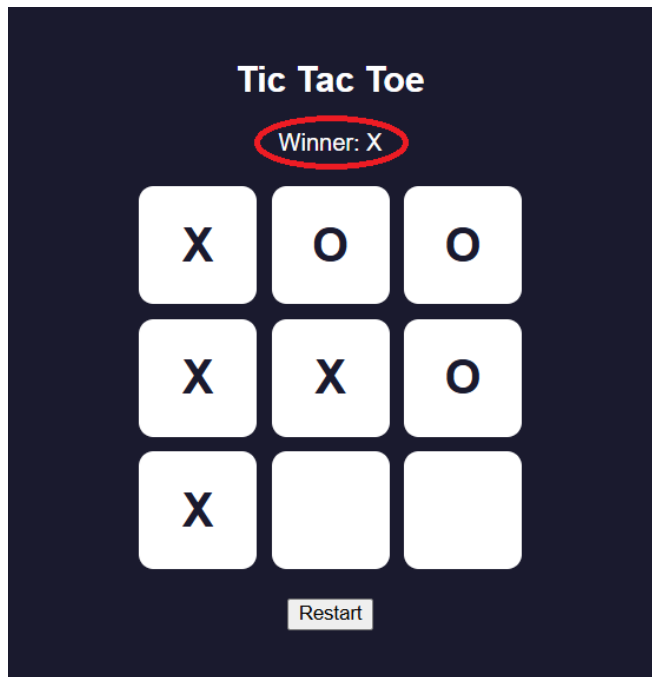
Clicking the + and - buttons to add or subtract 1 from the counter. In addition, there is a third button, `reset`, which can be used to reset the counter to its default settings.



Tic Tac Toe

A Fully interactive game, it always displays which player is next. At the end of the game it displays who the winner is, and in case of a tie it displays „it's a tie”.





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.

