



dr inż. Michał Malinowski
michal.malinowski@warszawa.merito.pl



[https://www.linkedin.com/in/
michał-malinowski-malkon/](https://www.linkedin.com/in/michał-malinowski-malkon/)

JavaScript po stronie klienta: Obsługa zdarzeń i Formularze

Wykład 06

JĘZYKI PROGRAMOWANIA
STUDIUM PRZYPADKU

Podział języków programowania

Kryterium	Kategoria	Przykłady języków
Poziom abstrakcji	Języki niskiego poziomu	Asembler, język maszynowy
	Języki wysokiego poziomu	C, Java, Python
Paradygmaty programowania	Proceduralne	C, Fortran
	Obiektowe	Java, C++, Python
	Funkcyjne	Haskell, Lisp
	Logiki	Prolog
	Skryptowe	JavaScript, Python, Ruby
Sposób wykonywania	Kompilowane	C, C++, Rust
	Interpretowane	Python, Ruby, JavaScript
	JIT (Just-In-Time)	Java (JVM), C# (CLR)
Zastosowanie	Ogólnego przeznaczenia	Python, Java, C++
	Specjalistyczne	MATLAB, SQL, R
Składnia	C-like	C++, Java, C#
	Pascal-like	Delphi, Ada
	ML-like	Haskell, F#
Inne kryteria	Deklaratywne	SQL, HTML
	Imperatywne	C, Java
	Statycznie typowane	Java, C++
	Dynamicznie typowane	Python, JavaScript

Formularze kontaktowe

Zastosowanie:

Walidacja danych wejściowych na stronie klienta przed wysłaniem formularza do serwera.

Przypadek użycia:

Sprawdzanie, czy użytkownik wpisał prawidłowy adres email, czy wszystkie wymagane pola zostały wypełnione, i czy wprowadzone hasło spełnia określone kryteria bezpieczeństwa.

```
<html lang="pl">
<head>
  <title>Walidacja Formularza</title>
</head>
<body>
  <form id="contactForm">
    <label for="email">Email:</label>
    <input type="email" id="email" required>
    <br>
    <label for="password">Hasło:</label>
    <input type="password" id="password" required>
    <br>
    <button type="submit">Wyślij</button>
  </form>

  <script src="validate.js"></script>
</body>
</html>
```

Formularze kontaktowe

Opis działania

addEventListener('submit', ...) – Nasłuchuje na zdarzenie submit na formularzu.

event.preventDefault() – Zapobiega domyślnej akcji przeglądarki, czyli wysłaniu formularza, co pozwala na jego walidację w JavaScript.

Sprawdzanie formatu e-mail – Używając prostej metody `includes('@')` sprawdzamy, czy e-mail zawiera symbol @.

Walidacja hasła – Sprawdzamy, czy hasło ma co najmniej 8 znaków, przynajmniej jedną wielką literę i jedną cyfrę, korzystając z wyrażeń regularnych (`match`).

Wyświetlanie błędów – Jeśli są jakieś błędy, zostaną one wyświetlone za pomocą `alert()`.

Wysyłanie formularza – Gdy wszystkie dane są prawidłowe, formularz może być teoretycznie wysłany.

```
document.getElementById('contactForm').addEventListener('submit', function(event) {  
    event.preventDefault(); // Zapobiega domyślnemu zachowaniu formularza  
  
    var email = document.getElementById('email').value;  
    var password = document.getElementById('password').value;  
    var errorMessage = '';  
  
    if (!email.includes('@')) {  
        errorMessage += 'Adres email musi zawierać @.\n';  
    }  
  
    if (password.length < 8) {  
        errorMessage += 'Hasło musi mieć co najmniej 8 znaków.\n';  
    } else if (!password.match(/[A-Z]/)) {  
        errorMessage += 'Hasło musi zawierać przynajmniej jedną wielką literę.\n';  
    } else if (!password.match(/[0-9]/)) {  
        errorMessage += 'Hasło musi zawierać przynajmniej jedną cyfrę.\n';  
    }  
  
    if (errorMessage) {  
        alert(errorMessage);  
    } else {  
        // Jeśli wszystko jest w porządku, formularz może być wysłany  
        // Tutaj można dodać logikę wysyłania formularza  
        alert('Formularz wysłany pomyślnie!');  
    }  
});
```

Autouzupełnianie formularzy

Zastosowanie:

Ulepszenie interaktywności formularzy przez sugerowanie możliwych odpowiedzi na podstawie wcześniej wprowadzonych danych.

Przypadek użycia:

Gdy użytkownik zaczyna wpisywać adres, skrypt może sugerować pełne adresy zapisane wcześniej lub popularne adresy z bazy danych.

```
<!DOCTYPE html>
<html lang="pl">
<head>
  <title>Autouzupełnianie Adresu</title>
</head>
<body>
  <label for="addressInput">Adres:</label>
  <input type="text" id="addressInput" placeholder="Zacznij wpisywać adres...">
  <ul id="suggestions" style="list-style-type: none; padding: 0;"></ul>

  <script src="autocomplete.js"></script>
</body>
</html>
```

Autouzupełnianie formularzy

Opis działania

addEventListener('input', ...) – Nasłuchuje na zdarzenie input na polu adresu, co oznacza, że każde wpisanie znaku wyzwała funkcję.

Czyszczenie listy sugestii – Na początku każdej operacji czyszczone są wcześniejsze sugestie.

Filtrowanie adresów – Używając metody filter, skrypt wyszukuje adresy, które zawierają wpisany przez użytkownika ciąg znaków.

Wyświetlanie sugestii – Dla każdego pasującego adresu tworzony jest element listy li, który jest dodawany do listy sugestii pod polem input.

Wybór sugestii – Kliknięcie na którykolwiek z elementów listy wpisuje jego treść w pole input i czyści listę sugestii.

```
document.getElementById('addressInput').addEventListener('input', function() {
    var input = this.value;
    var suggestions = document.getElementById('suggestions');
    suggestions.innerHTML = ''; // Czyści listę sugestii

    if (input.length > 2) { // Sugeruj adresy tylko, gdy wpisane są co najmniej 3 z
        var hintData = [
            'Warszawa, ul. Marszałkowska 1',
            'Warszawa, ul. Mokotowska 15',
            'Wrocław, ul. Rуска 22',
            'Kraków, ul. Floriańska 5',
            'Gdańsk, ul. Długa 8'
        ]; // Przykładowe adresy

        // Filtruj adresy zawierające wpisany tekst
        var filteredHints = hintData.filter(function(item) {
            return item.toLowerCase().includes(input.toLowerCase());
        });

        // Wyświetl pasujące sugestie
        filteredHints.forEach(function(item) {
            var listItem = document.createElement('li');
            listItem.textContent = item;
            listItem.style.cursor = 'pointer';
            listItem.onclick = function() {
                document.getElementById('addressInput').value = this.textContent; //
                suggestions.innerHTML = ''; // Czyść sugestie po wyborze
            };
            suggestions.appendChild(listItem);
        });
    }
});
```

Dynamiczne formularze

Zastosowanie:

Modyfikacja formularzy w czasie rzeczywistym w odpowiedzi na akcje użytkownika.

Przypadek użycia:

Dodawanie lub usuwanie pól formularza w zależności od wybranych opcji, np. jeśli użytkownik zaznaczy, że jest przedsiębiorcą, pojawiają się dodatkowe pola do wypełnienia dotyczące firmy.

```
<html lang="pl">
<head>
  <title>Dynamiczny Formularz</title>
</head>
<body>
  <form id="registrationForm">
    <label for="userType">Jestem:</label>
    <select id="userType">
      <option value="private">Prywatnie</option>
      <option value="business">Przedsiębiorcą</option>
    </select>
    <div id="businessFields" style="display: none;">
      <label for="companyName">Nazwa Firmy:</label>
      <input type="text" id="companyName">
      <br>
      <label for="taxNumber">NIP:</label>
      <input type="text" id="taxNumber">
    </div>
    <button type="submit">Zarejestruj</button>
  </form>

  <script src="dynamicForm.js"></script>
</body>
</html>
```

Dynamiczne formularze

Opis działania

`addEventListener('change', ...)` – Nasłuchuje na zmiany w selektorze `#userType`, aby odpowiedzieć na wybór użytkownika.

Determinowanie wyświetlania pól – Zależnie od wartości wybranej w selektorze, pola dotyczące informacji biznesowych są wyświetlane lub ukrywane. Użycie `style.display` kontroluje, czy element jest widoczny na stronie.

```
document.getElementById('userType').addEventListener('change', function() {  
    var selectedType = this.value;  
    var businessFields = document.getElementById('businessFields');  
  
    if (selectedType === 'business') {  
        businessFields.style.display = 'block'; // Pokaż dodatkowe pola  
    } else {  
        businessFields.style.display = 'none'; // Ukryj dodatkowe pola  
    }  
});
```

Interaktywne ankiety i quizy

Zastosowanie:

Stworzenie interaktywnych ankiet i quizów, które reagują na odpowiedzi użytkownika.

Przypadek użycia:

Pokazywanie kolejnych pytań na podstawie wcześniej udzielonych odpowiedzi lub ocenianie odpowiedzi w czasie rzeczywistym i przedstawianie wyników na końcu.

```
<!DOCTYPE html>
<html lang="pl">
<head>
  <title>Interaktywny Quiz</title>
</head>
<body>
  <h1>Quiz Wiedzy</h1>
  <div id="quizContainer">
    <p id="question"></p>
    <button class="answerBtn" style="display:none;">Odpowiedź 1</button>
    <button class="answerBtn" style="display:none;">Odpowiedź 2</button>
    <button class="answerBtn" style="display:none;">Odpowiedź 3</button>
    <button id="nextQuestion" style="display:none;">Następne pytanie</button>
  </div>
  <div id="result" style="display:none;"></div>

  <script src="quiz.js"></script>
</body>
</html>
```

Interaktywne ankiety i quizy

Opis działania

Struktura pytań - Pytania zdefiniowane są jako obiekty w tablicy `questions`, każde z polami: `question` (treść pytania), `answers` (możliwe odpowiedzi) oraz `correct` (indeks prawidłowej odpowiedzi).

Wyświetlanie pytania - Funkcja `displayQuestion` aktualizuje treść DOM, prezentując użytkownikowi aktualne pytanie oraz przyciski z możliwymi odpowiedziami. Każdy przycisk jest powiązany z funkcją `handleAnswer` odpowiadającą za obsługę kliknięcia.

Obsługa odpowiedzi - Funkcja `handleAnswer` porównuje indeks wybranej odpowiedzi z indeksem prawidłowej odpowiedzi danego pytania. W zależności od wyniku porównania, aktualizuje wynik quizu i wyświetla odpowiedni komunikat (poprawna lub błędna odpowiedź).

Przechodzenie do następnego pytania - Po udzieleniu odpowiedzi, użytkownik może kliknąć przycisk "Następne pytanie" aby przejść do kolejnego pytania lub, w przypadku ostatniego pytania, zakończyć quiz. Funkcja `finishQuiz` jest wywoływana na końcu, aby wyświetlić wynik quizu i ukryć formularz pytań.

```
const questions = [
  { question: "Stolica Polski to:", answers: ["Warszawa", "Kraków", "Wrocław"], correct: 0 },
  { question: "2 + 2 * 2 = ?", answers: ["6", "8", "4"], correct: 0 },
  { question: "Najdłuższa rzeka w Polsce to:", answers: ["Wisła", "Odra", "Warta"], correct: 0 }
];
let currentQuestion = 0;
let score = 0;

function displayQuestion() {
  const question = questions[currentQuestion];
  document.getElementById('question').textContent = question.question;
  document.querySelectorAll('.answerBtn').forEach((btn, index) => {
    btn.textContent = question.answers[index];
    btn.style.display = 'block';
    btn.onclick = () => handleAnswer(index);
  });
  document.getElementById('nextQuestion').style.display = 'none';
}

function handleAnswer(selected) {
  if (selected === questions[currentQuestion].correct) {
    score++;
    alert("Dobra odpowiedź!");
  } else {
    alert("Zła odpowiedź!");
  }
  document.getElementById('nextQuestion').style.display = 'block';
}

document.getElementById('nextQuestion').addEventListener('click', () => {
  if (++currentQuestion < questions.length) {
    displayQuestion();
  } else {
    finishQuiz();
  }
});

function finishQuiz() {
  document.getElementById('quizContainer').style.display = 'none';
  document.getElementById('result').textContent = `Twój wynik: ${score}/${questions.length}`;
  document.getElementById('result').style.display = 'block';
}

displayQuestion();
```

Obsługa zdarzeń w czasie rzeczywistym

Zastosowanie:

Reagowanie na zdarzenia w czasie rzeczywistym, takie jak ruch myszy, kliknięcia, przewijanie strony.

Przypadek użycia:

Zmiana wyglądu strony lub wyświetlanie informacji kontekstowych gdy użytkownik najedzie kursorem na określone elementy.

```
<html lang="pl">
<head>
  <title>Obsługa Zdarzeń w Czasie Rzeczywistym</title>
  <style>
    .highlight {
      color: red;
      cursor: pointer;
    }
  </style>
</head>
<body>
  <div id="infoBox" style="display: none; position: absolute; padding: 10px; background: yellow; border: 1px solid black;"></div>
  <p class="interactive" data-info="To jest paragraf pierwszy.">Paragraf 1</p>
  <p class="interactive" data-info="To jest paragraf drugi.">Paragraf 2</p>
  <p class="interactive" data-info="To jest paragraf trzeci.">Paragraf 3</p>

  <script src="realtimeEvents.js"></script>
</body>
</html>
```

Obsługa zdarzeń w czasie rzeczywistym

Opis działania

Wybór elementów i zdarzenia: Skrypt najpierw wybiera wszystkie elementy z klasą `interactive`. Dla każdego z nich dodaje zdarzenia `mouseenter` i `mouseleave`.

Funkcje obsługi zdarzeń: Gdy użytkownik najedzie kursorem na element, dodawana jest klasa `highlight`, a funkcja `showInfo` wyświetla dodatkowe informacje pobrane z atrybutu `data-info` danego elementu. Informacje są wyświetlane w `div` `infoBox`, który jest pozycjonowany bezpośrednio pod elementem.

Ukrywanie informacji: Gdy kursor opuści element, klasa `highlight` jest usuwana, a `infoBox` jest ukrywany przez funkcję `hideInfo`.

```
document.querySelectorAll('.interactive').forEach(item => {
  item.addEventListener('mouseenter', function() {
    this.classList.add('highlight');
    showInfo(this, this.getAttribute('data-info'));
  });

  item.addEventListener('mouseleave', function() {
    this.classList.remove('highlight');
    hideInfo();
  });
});

function showInfo(element, info) {
  var infoBox = document.getElementById('infoBox');
  infoBox.style.display = 'block';
  infoBox.textContent = info;
  infoBox.style.top = element.offsetTop + element.offsetHeight + 'px';
  infoBox.style.left = element.offsetLeft + 'px';
}

function hideInfo() {
  var infoBox = document.getElementById('infoBox');
  infoBox.style.display = 'none';
}
```

Złożone walidacje formularzy

Zastosowanie:

Wykonywanie złożonych walidacji formularzy, które wymagają interakcji z różnymi elementami formularza.

Przypadek użycia:

Sprawdzanie, czy data zakończenia jest późniejsza niż data rozpoczęcia, czy suma wartości wprowadzonych w kilku polach nie przekracza określonego limitu.

```
<html lang="pl">
<head>
  <title>Złożona Walidacja Formularza</title>
</head>
<body>
  <form id="complexForm">
    <label for="startDate">Data rozpoczęcia:</label>
    <input type="date" id="startDate" required>
    <br>
    <label for="endDate">Data zakończenia:</label>
    <input type="date" id="endDate" required>
    <br>
    <label for="value1">Wartość 1:</label>
    <input type="number" id="value1" required>
    <br>
    <label for="value2">Wartość 2:</label>
    <input type="number" id="value2" required>
    <br>
    <label for="value3">Wartość 3:</label>
    <input type="number" id="value3" required>
    <br>
    <button type="submit">Sprawdź</button>
  </form>

  <script src="complexValidation.js"></script>
</body>
</html>
```

Złożone walidacje formularzy

Opis działania

addEventListener('submit', ...) – Nasłuchuje na zdarzenie submit w formularzu.

Przetwarzanie dat i wartości – Konwertuje wartości z pól formularza na odpowiednie typy danych, takie jak daty i liczby całkowite.

Walidacja dat – Sprawdza, czy data zakończenia jest później niż data rozpoczęcia.

Walidacja sumy wartości – Sumuje wartości z trzech pól i sprawdza, czy suma nie przekracza ustalonego limitu.

Komunikaty o błędach – Wyświetla komunikaty o błędach, jeśli takie występują, lub informuje o poprawności danych.

```
document.getElementById('complexForm').addEventListener('submit', function(event) {  
    event.preventDefault(); // Zapobiega domyślnemu wysłaniu formularza  
  
    const startDate = new Date(document.getElementById('startDate').value);  
    const endDate = new Date(document.getElementById('endDate').value);  
    const value1 = parseInt(document.getElementById('value1').value, 10);  
    const value2 = parseInt(document.getElementById('value2').value, 10);  
    const value3 = parseInt(document.getElementById('value3').value, 10);  
  
    let errors = '';  
    const totalValue = value1 + value2 + value3;  
    const maxLimit = 100; // maksymalny dozwolony limit dla sumy wartości  
  
    if (endDate <= startDate) {  
        errors += 'Data zakończenia musi być późniejsza niż data rozpoczęcia.\n';  
    }  
  
    if (totalValue > maxLimit) {  
        errors += `Suma wartości przekracza dozwolony limit ${maxLimit}. Aktualna suma: ${totalValue}\n`;  
    }  
  
    if (errors) {  
        alert(errors);  
    } else {  
        alert('Wszystkie dane są prawidłowe!');  
    }  
});
```

Evolution of Memory Storage

