UNIWERSYTET RZESZOWSKI

WYDZIAŁ NAUK ŚCISŁYCH I TECHNICZNYCH

INSTYTUT INFORMATYKI

Krystian Filipek

134907

Informatyka

Technologie Internetowe - JavaScript

Praca projektowa

Praca wykonana pod kierunkiem
dr. Katarzyna Garwol

Rzeszów 2025

Spis treści

1. Wprowadzenie

Strona internetowa **"VEXILLARIUS"** stworzona została na potrzeby drugiego projektu (Zakres **JavaScript**) z przedmiotu **Technologie Internetowe** na 2 roku Informatyki na **Uniwersytecie Rzeszowskim**. Strona jako projekt posiada skrypty napisane w języku **JavaScript**, które nadają witrynie pewne funkcjonalności. Całość spełnia podstawowe standardy w tworzeniu stron internetowych, którymi między innymi są **responsywność**, intuicyjny układ, jak i również płynne działanie umieszczonych na stronie skryptów.

The website **"VEXILLARIUS"** was created for the second project (JavaScript) in the **Internet Technologies** course for second-year Computer Science students at the **University of Rzeszów**. The project includes scripts written in **JavaScript**, which provide the site with certain functionalities. The entire project meets basic standards for website development, including **responsiveness**, an intuitive layout, and the smooth operation of scripts placed on the site.

2. Opis założeń projektu

Celem projektu było stworzenie wielofunkcyjnej strony internetowej o nazwie **Vexillarius**, będącej praktycznym sprawdzianem podstawowych umiejętności głównie z zakresu technologii **JavaScript**. Strona została zaprojektowana w sposób spójny wizualnie z myślą o zaimplementowanych skryptach.

Głównym założeniem projektu było przejście ze statycznego dokumentu HTML/CSS do interaktywnej witryny poprzez użycie skryptów. W ramach prac zaimplementowano ich kilka, gdzie część z nich to narzędzia takie jak kalkulator oraz lista zadań, które pozwalają na interakcję z użytkownikiem.

Rozwiązanie problemu projektowego przebiegło w następujących krokach:

1. Przygotowanie struktury dokumentu `HTML5`.
2. Stworzenie stylów `CSS3` wraz z tymi odpowiedzialnymi za podstawową responsywność witryny - `Media Queries`.
3. Implementacja skryptów w języku `JavaScript` odpowiedzialnych za warstwę użytkową strony.

2.1. Wymagania funkcjonalne

Projekt realizuje następujące funkcjonalności, które pozwalają użytkownikowi na interakcję z witryną:

Przełącznik trybu light/dark mode: Możliwość zmiany motywu strony (jasny/ciemny) za pomocą jednego przycisku zlokalizowanego w prawym górnym rogu paska nawigacyjnego.

Interaktywna lista zadań (To-Do): Lista pozwala użytkownikowi na wpisywanie i dodawanie zadań, jak i ich usuwanie za pomocą kliknięcia. Lista zadań wyświetlana jest za pomocą dynamicznej listy na ekranie.

Kalkulator liczb - arabskie/rzymskie: Kalkulator przeliczający wpisane liczby arabskie na ich rzymski odpowiednik, prezentujący wynik w czasie rzeczywistym.

Obsługa formularza kontaktowego: Sekcja z walidacją pól (imię, email, treść wiadomości). Skrypt sprawdza poprawność danych i informuje użytkownika o błędach lub powodzeniu wysłania wiadomości za pomocą autorskich komunikatów - *custom alerts*.

Dynamiczny pasek nawigacji: Nagłówek strony podczas scrollowania w dół zmienia swój wygląd poprzez pojawienie się dolnego obramowania oddzielającego go od reszty strony.

2.2. Wymagania niefunkcjonalne

Środowisko testowe

Testy projektu odbywały się lokalnie na komputerze o przedstawionej poniżej specyfikacji.

- **System operacyjny:** Microsoft Windows 11 Pro
- **Model:** Lenovo Yoga 7 2-in-1 14IML9
- **Procesor:** Intel® Core™ Ultra 5-125H 4.5 GHz
- **Pamięć RAM:** 16GB DDR4
- **Karta graficzna:** Intel® Arc™ graphics (Zintegrowana)

- **Dysk:** Samsung MZAL81T0HDLB-00BL2 1TB
- **Rozdzielczość ekranu:** 2880 x 1800
- **IDE:** Visual Studio Code

Technologie: Strona zgodnie z przyjętymi wytycznymi skupiać powinna się na wykorzystaniu skryptów w języku **JavaScript**. Dozwolone jest natomiast wykorzystanie ich w formie strony internetowej, gdzie wykorzystane będą również inne technologie.

Wydajność: Strona powinna charakteryzować się krótkim czasem ładowania poprzez brak ciężkich bibliotek.

Kompatybilność: Witryna powinna wyświetlać się poprawnie zarówno na przeglądarkach wyposażonych w silnik `Chromium` jak i `Gecko`.

Responsywność (RWD): Strona poprawnie wyświetla się na różnych szerokościach ekranu - od monitorów aż po urządzenia mobilne. Układ elementów dostosowuje się tak, aby zachować czytelność i wygodę klikania.

Spójność wizualna: Wykorzystanie zmiennych globalnych w kaskadowych arkuszach stylów w celu zarządzania kolorystyką oraz typografią nadaje stronie estetycznego, jednolitego wyglądu.

3. Opis struktury projektu

3.1. Użyte technologie

Strona internetowa **Vexillarius** została wykonana przy użyciu natywnych technologii webowych. Zgodnie z założeniami projektowymi, zrezygnowano z zewnętrznych frameworków na rzecz czystego kodu.

`HTML5` wykorzystano do stworzenia podstawowego szkieletu witryny. Zastosowanie znaczników takich jak `<header>`, `<nav>`, `<section>` oraz `<footer>` zapewnia poprawną strukturę dokumentu i wspiera jego dostępność (accessibility).

`CSS` wykorzystano do stworzenia warstwy wizualnej opartej na nowoczesnych standardach, w tym Flexbox do układów oraz zmienne do prostego zarządzania kolorami i typografią.

`JavaScript` wykorzystano do stworzenia skryptów odpowiadających za funkcjonalność całej strony. Zastosowano DOM (Document Object Model) oraz obsługę zdarzeń w czasie rzeczywistym.

3.2. Struktura katalogów

Projekt został podzielony na katalogi, co zapewnia separację sturktury, stylów oraz skryptów. Ułatwia to konserwację kodu oraz jego ewentualną rozbudowę.

```
ti-project-2/
├── css/ ..... Arkusze stylów
│   └── style.css ..... Główny arkusz stylów witryny
├── documentation/ ..... Pliki źródłowe dokumentacji
├── html/ ..... Struktura dokumentów
│   └── index.html ..... Główny plik strony internetowej
└── javascript/ ..... Logika aplikacji (skrypty)
    ├── animation.js .....Obsługa animacji i efektów scrolla
    ├── calc.js .....Logika kalkulatora rzymskiego
    ├── form.js .....Obsługa formularza i walidacji
    ├── theme.js .....Skrypt zmiany motywu kolorystycznego
    └── todo.js .....Obsługa interaktywnej listy zadań
```

3.3. Konwencje kodowania i organizacja stylów

W arkuszach stylów zastosowano konwencje zwiększające czytelność i uniwersalność kodu:

- **Zmienne CSS (:root)**: Definicje kolorów (tła, tekstu, akcentów) zostały wraz z typografią zamknięte w zmiennych. Pozwala to na błyskawiczną zmianę globalnej palety barw oraz umożliwiło Implementację trybu `dark-mode` poprzez nadpisywanie wartości zmiennych w klasie nadrzędnej.

- **Skalowalne jednostki REM**: Bazowa wielkość czcionki została ustawiona na `50%` (8px). Dzięki temu projekt jest łatwiej skalowalny, a jednostki `REM` pozwalają na zachowanie proporcji między elementami.

- **Box-siging**: Globalne ustawienie `border-box` zapobiega problemom z obliczaniem szerokości elementów przy dodawaniu marginesów wewnętrznych (padding).

3.4. Implementacja logiki w JavaScript

Skrypty projektu zostały zaprojektowane w sposób modułowy. Każda funkcjonalność znajduje się w osobnym pliku, co zapobiega konfliktom nazw i ułatwia ewentualne debugowanie:

- **Manipulacja DOM**: Skrypty dynamicznie tworzą (np. lista zadań) lub modyfikują (np. alerty) elementy drzewa dokumentu.
- **Event Listeners**: Interakcja z użytkownikiem opiera się na nasłuchiwaniu zdarzeń takich jak `click`, `scroll` czy `keypress`.
- **Powiadomienia**: Zastosowano funkcje czasu (`setTimeout`), aby kontrolować czas wyświetlania komunikatów błędów i sukcesów w przypadku formularza kontaktowego.

3.5. Responsywność (RWD)

Strona dostosowuje się do urządxeń mobilnych poprzez *Media Queries*. Zastosowano podejście które zakłada płynne przejścia między szerokościami tak aby strona wyglądała czytelnie na urządzeniach węższych.

- **Typografia:** Wykorzystano funkcję `clamp()`, która automatycznie dobiera wielkość czcionki w zależności od szerokości okna przeglądarki.
- **Pasek nawigacji:** Pasek nawigacji na telefonach zmienia swój układ na kolumnowy.
- **Flex-column:** Na mniejszych ekranach kontenery formularza i narzędzi (kalkulator, lista) zmieniają orientacje z poziomej na pionową.

3.6. Prezentacja najważniejszych skryptów i ich fragmentów

W niniejszej sekcji przedstawiono implementację oraz krótką analizę naważniejszych skryptów **JavaScript**. Każdy fragment kodu został opatrzony krótkim komentarzem technicznym wyjaśniającym zasadę jego działania.

3.6.1. Zarządzanie motywem (theme.js)

Skrypt ten odpowiada za interakcję z użytkownikiem w zakresie wyboru preferowanego motywu wizualnego (Dark Mode / Light Mode).

Listing 3.1. Przełącznik motywów Dark Mode / Light Mode

```
var themeButton = document.getElementById('theme-button');
themeButton.onclick = function() {
var pageBody = document.body;
pageBody.classList.toggle('dark-mode');
};
```

Wyjaśnienie: Funkcja przypisana do zdarzenia `onclick` przycisku wywołuje metodę `toggle()` na liście klas dokumentu. Mechanizm automatycznie dodaje klasę `.dark-mode` przy jej braku lub usuwa ją, gdy jest już ona obecna. Powoduje to natycmiastową aktualizację wartości zmiennych CSS.

3.6.2. Animacja powitalna sekcji Hero (animation.js)

Skrypt odpowiedzialny za pojawienie się tytułu strony wykorzystuje funkcję czasu do płynnej zmiany właściwości CSS.

Listing 3.2. Implementacja animacji "fade-in-up"

```
var title = document.querySelector('.hero-content h1');
var position = 50;
var opacity = 0;
title.style.marginTop = position + "px";
title.style.opacity = opacity;
var timer = setInterval(function() {
position = position - 1;
opacity = opacity + 0.02;
title.style.marginTop = position + "px";
title.style.opacity = opacity;

if (position <= 0) {
    clearInterval(timer);
    title.style.marginTop = "0px";
    title.style.opacity = "1";
}
}, 15);
```

Wyjaśnienie: Skrypt ręcznie steruje animacją przy użyciu metody `setInterval`, która wykonuje blok co 15 milisekund. W każdym kroku wartość zmiennej `position` maleje powodując ruch tekstu w górę, zaś `opacity` rośnie zwiększając widoczność. Po osiągnięciu docelowej pozycji, interwał jest czyszczony przez `clearInterval`, co zatrzymuje proces.

3.6.3. Pasek nawigacji (animation.js)

Skrypt ten zapewnia wizualne oddzielenie paska nawigacji od treści strony podczas jej przewijania.

Listing 3.3. Zmiana stylu nagłówka przy przewijaniu

```
var header = document.querySelector('header');
window.addEventListener('scroll', () => {
if (window.scrollY > 0) {
header.classList.add('scrolled');
} else {
header.classList.remove('scrolled');
}
});
```

Wyjaśnienie: Skrypt rejestruje słuchacza zdarzeń (`Event Listener`) dla obiektu `window`. Gdy wartość `scrollY` przekracza 0, do nagłówka dodawana jest klasa `.scrolled`, która w CSS odpowiada za wyświetlenie dolnego obramowania `border-bottom`.

3.6.4. Konwerter liczb rzymskich (calc.js)

Skrypt implementuje algorytm przeliczania liczb arabskich na liczby systemu rzymskiego. Z oczywistych względów nie wylistowano obsługi przycisków i deklaracji zmiennych.

Listing 3.4. Logika przeliczania liczb arabskich na rzymskie

```
function convert() {
var val = parseInt(numberInput.value);
if (isNaN(val) || val <= 0 || val > 3999) {
    out.innerText = "Error";
    return;
}

var numbers = [1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1];
var roman = ["M", "CM", "D", "CD", "C", "XC", "L", "XL", "X", "IX", "
    V", "IV", "I"];
var finalStr = "";

for (var i = 0; i < numbers.length; i++) {
    while (val >= numbers[i]) {
        finalStr = finalStr + roman[i];
        val = val - numbers[i];
    }
}
out.innerText = finalStr;
}
```

Wyjaśnienie: Algorytm sprawdza poprawność danych wejściowych (Zakres 1 - 3999), a następnie wykonuje pętle `for` do iteracji po tablicy zdefiniowanych progów liczbowych. Wewnętrzna pętla `while` odejmuje najwyższą możliwą wartość od liczby wejściowej i dopisuje odpowiadający jej znak rzymski do wyniku.

3.6.5. Interaktywna lista zadań (todo.js)

Skrypt demonstruje działanie listy zadań.

Listing 3.5. Dynamiczne dodawanie i usuwanie zadań

```
function addNewTask() {
var text = inputField.value;
if (text != "") {
    var newItem = document.createElement('li');
    newItem.className = 'todo-item';
    newItem.innerHTML = "<span>" + text + "</span>";

    newItem.onclick = function() {
        this.remove();
    };

    myList.appendChild(newItem);
    inputField.value = "";
}
}
```

Wyjaśnienie: Po aktywacji funkcji, skrypt tworzy w pamięci nowy element `li`, nadaje mu klasę stylu i wstrzykuje tekst pobrany z inputa. Każdemu elementowi przypisywana jest również funkcja `remove()`, wyzwalana po kliknięciu w dodany wcześniej element, co umożliwia selektywne usuwanie z listy.

3.6.6. Obsługa formularza i system alertów (form.js)

Skrypt odpowiada za weryfikację danych w formularzu oraz za wysyłanie powiadomień w przypadku błędnych danych lub poprawnie przesłanego formularza.

Listing 3.6. Walidacja w formularzu i generowanie powiadomień

```
function myAlert(msg, mode) {
    var old = document.querySelector('.custom-alert');
    if (old) { old.remove(); }

    var box = document.createElement('div');
    box.className = 'custom-alert ' + mode;
    box.innerHTML = "<span>" + (mode === 'success' ? 'EMOTKA1' : '
    EMOTKA2') + "</span> " + msg;
    document.body.appendChild(box);

    setTimeout(function() { box.classList.add('show'); }, 50);
    setTimeout(function() { box.classList.remove('show'); }, 3000);
}

var contactForm = document.getElementById('contact-form');

if (contactForm) {
    contactForm.onsubmit = function(e) {
        e.preventDefault();
        var userEmail = document.getElementById('email').value;

        if (userEmail.indexOf("@") == -1 || userEmail.length < 5) {
            myAlert("Adres email jest nieprawidłowy!", "error");
            return;
        }
        myAlert("Wiadomość została przesłana.", "success");
        contactForm.reset();
    };
}
```

Wyjaśnienie: Funkcja `myAlert` dynamicznie buduje okno powiadomienia w drzewie DOM, wykorzystując funkcję `setTimeout` do sterowania animacją pojawiania się i znikania klasy stylu. Skrypt do walidacji danych formularza blokuje domyślną wysyłkę formularza przy użyciu `e.preventDefault()` i przeprowadza test poprawności adres e-mail, imienia, wiadomości wywołując alert z odpowiednim statusem sukcesu albo błędu.

4. Harmonogram realizacji projektu

4.1. Etapy realizacji projektu

Proces tworzenia witryny był ciągiem mniejszych bądź większych etapów, takich jak:

1. Analiza wymagań i ułożenie wstępnego schematu.
2. Wstępne stworzenie układu strony.
3. Implementacja rozwiązania przy użyciu technologii `HTML5`.
4. Wprowadzenie kaskadowych arkuszy stylów.
5. Dodanie skryptów `JavaScript`
6. Drobne zmiany wizualne takie jak dodanie drobnego skryptu dla paska nawigacyjnego.
7. Stworzenie dokumentacji technicznej.

4.2. System kontroli wersji

- Podczas tworzenia projektu wykorzystano system kontroli **Git** poprzez aplikację **GitHub Desktop**. Do przechowywania kodu źródłowego wykorzystano repozytorium na platformie **GitHub**. Repozytorium dostępne jest pod adresem:

  https://github.com/Avsh11/web-technologies/tree/main/ti-project-2

- Repozytorium zawiera pełną wersję projektu po ostatecznych poprawkach (w chwili oddania) i dostępne będzie w trybie publicznym ustalonym przez wykładowcę prowadzącego przedmiot.

5. Prezentacja warstwy użytkowej projektu

6. Instrukcja uruchomieniowa

Aby poprawnie uruchomić witrynę w celach testowych, należy wykonać następujące kroki uprzednio przygotowując środowisko uruchomieniowe:

Wymagane środowisko

- **Proponowany system operacyjny:** Microsoft Windows / Linux / Apple MacOS
- **IDE:** Visual Studio Code
- **Rozszerzenia:** Live Server autorstwa: **Ritwicka Deya**
- **Repozytorium GitHub:** *https://github.com/Avsh11/web-technologies*
- **Proponowany system kontroli wersji:** GitHub Desktop App

Krok 1. - Konfiguracja środowiska testowego

1. Pobierz edytor `Visual Studio Code`.
2. Pobierz aplikację `GitHub Desktop`.
3. Uruchom apliakcję `GitHub Desktop` i wybierz opcję sklonowania repozytorium w zakładce **File**.
4. Sklonowane repozytorium umieszczone ma zostać w wybranym, stworzonym przez użytkownika katalogu.

Krok 2. - Uruchomienie strony

1. Otwórz uprzednio skonfigurowaną aplikację `Visual Studio Code`.
2. Upewnij się, że poprawnie zainstalowałeś rozszerzenie `Live Server`.
3. Otwórz katalog, w którym sklonowałeś repozytorium.
4. Wejdź w plik `index.html`.
5. W prawym dolnym rogu **IDE** kliknij w przycisk `Go Live` jak na obrazku poniżej - **rys. 6.1**

Rys. 6.1. Przycisk `Go Live` rozszerzenia `Live Server`

7. Podsumowanie

Celem projektu było stworzenie responsywnej i wizualnie estetycznej witryny zawierającej skrypty zapewniające funkcjonalności dla strony. Umożliwić ma ona użytkownikowi łatwą nawigację przez zrozumiale rozmieszczone treści. Dzięki zastosowaniu kaskadowych arkuszy stylów i języka znaczników `HTML5` jak i również języka `JavaScript` udało się stworzyć witrynę spełniającą powyższe kryteria.

Mimo pomyślnego zrealizowania większości najważniejszych funkcjonalności wedle etapów rozwoju witryna wciąż posiada miejsce na wiele innych równie ważnych funkcjonalności, o które można w przyszłości rozbudować ową witrynę. Oto niektóre z nich:

1. Umożliwić działanie formularza poprzez skrypty oraz implementacje języka `PHP` wraz z wybraną bazą danych.

2. Bardziej zaawansowane animacje na stronie.

Bibliografia

[1] ChatGPT, 2025. Wykorzystano do wygenerowania dummy text. *https://openai.com/chatgpt*.

[2] Coolors, 2025. *https://coolors.co/*.

[3] Freepik, 2025. *https://www.freepik.com/*.

[4] Google Fonts, 2025. *https://fonts.google.com/*.

[5] Pexels, 2025. *https://www.pexels.com/*.

[6] Stack Overflow, 2025. *https://stackoverflow.com/*.

[7] Unsplash, 2025. *https://unsplash.com/*.

[8] W3schools, 2025. *https://www.w3schools.com/*.

Spis rysunków