ROZDZIAŁ 2

BIBLIOTEKA JQUERY

2 Biblioteka jQuery

2.1 Czym jest jQuery?

JQuery to popularna biblioteka JavaScript, która została stworzona w celu ułatwienia manipulowania dokumentem HTML, obsługi zdarzeń, animacji oraz komunikacji asynchronicznej w przeglądarkach internetowych. Została stworzona przez Johna Resiga i po raz pierwszy wydana w 2006 roku. jQuery jest napisana w czystym JavaScript i zapewnia wiele ułatwień dla programistów, dzięki czemu kod jest bardziej czytelny i łatwiejszy do utrzymania.

Jej motto „Write less, do more" (z ang. „Pisz mniej, osiągaj więcej") doskonale odzwierciedla założenia tej biblioteki. jQuery pozwala programistom efektywnie manipulować elementami DOM, obsługiwać zdarzenia, wykonywać animacje oraz realizować zapytania AJAX w sposób prosty i intuicyjny

Rysunek 2.1 Logo jQuery

Obecnie, mimo rosnącej popularności nowoczesnych frameworków takich jak React, Vue.js czy Angular, jQuery wciąż znajduje zastosowanie w wielu projektach, szczególnie w starszych aplikacjach i prostszych projektach webowych.

JQuery zyskało popularność z kilku powodów. Jednym z nich jest **uproszczona manipulacja DOM**, co oznacza, że jQuery wprowadza intuicyjny system selektorów, inspirowany CSS, który pozwala na szybkie i łatwe wybieranie oraz modyfikowanie elementów DOM. Biblioteka jQuery ułatwia też przypisywanie zdarzeń do elementów, czyli tak zwaną „obsługę zdarzeń", eliminując problemy z kompatybilnością między przeglądarkami.

Biblioteka ta upraszcza również pracę z technologią AJAX, umożliwiając łatwe wysyłanie i odbieranie danych bez przeładowywania strony (**zapytania AJAX)**. JQuery pozwala też na tworzenie **animacji i efektów wizualnych** w prosty sposób. Aby rozpocząć pracę z jQuery, należy włączyć bibliotekę do projektu,

czyli pliku z rozszerzeniem *.*html*. Można to zrobić na dwa sposoby, pierwszy – najprostszy – **korzystając z sieciowego CDN:**

```
<script src="https://code.jquery.com/jquery-
3.7.1.min.js"></script>
```

Listing 2.1 Zaimportowanie blibliteki jQuery z sieciowego CDN

Drugi – lokalnie – pobierając plik jQuery z oficjalnej strony (*https://jquery.com/download/*) i importując go:

```
<script src="js/jquery-3.7.1.min.js"></script>
```

Listing 2.2 Zaimportowanie biblioteki jQuery z zasobu lokalnego (folderu js) znajdującego się w projekcie.

Obie metody mają swoje mocne i słabe strony. Wybór między nimi zależy od specyfiki projektu oraz priorytetów zespołu programistycznego. Korzystanie z CDN ma swoje zalety, takie jak szybsze ładowanie dzięki globalnej pamięci podręcznej przeglądarek czy zmniejszenie obciążenia serwera lokalnego. Jednak wadą tego podejścia jest zależność od zewnętrznych serwerów — w razie ich niedostępności aplikacja może nie działać prawidłowo.

Używanie lokalnej kopii biblioteki pozwala na większą kontrolę nad wersją oraz niezależność od dostępności zewnętrznych serwerów. Z drugiej strony może jednak zwiększać obciążenie serwera i wymaga ręcznego zarządzania aktualizacjami pliku. Same programy możemy pisać w zwykłym notatniku lub innym edytorze, a pliki *.html otwierać za pomocą dowolnej przeglądarki Internetowej np. Chrome czy Safari.

2.2 Funkcje $(document).ready() i $(window).on("load")

Aby móc bezpiecznie manipulować elementami strony, dokument musi być w pełni przygotowany. JQuery automatycznie wykrywa ten stan gotowości, co pozwala na uruchomienie kodu we właściwym momencie. Kod umieszczony wewnątrz funkcji `$(document).ready()` zostanie wykonany w chwili, gdy struktura DOM (Document Object Model) strony zostanie wczytana i stanie się dostępna do modyfikacji przez JavaScript.

Z kolei kod w funkcji `$(window).on("load", function() { ... })` zostanie uruchomiony dopiero po pełnym załadowaniu całej strony, włącznie z zasobami takimi jak obrazy, style CSS, czy osadzone elementy iframe.

```
$(document).ready(function() {
    console.log("Dokument gotowy!");
});
```

Listing 2.3 Przykład użycia metody $(document).ready()

Doświadczeni programiści często stosują bardziej zwięzły zapis, który jest skrótem pełnej funkcji `$(document).ready()`.

```
$(function() {
    console.log("Dokument gotowy!");
});
```

Listing 2.4 Przykład skróconego użycia metody $(document).ready(), ujętej w znak $

Mimo że skrócona forma działa identycznie jak pełna, zaleca się używanie pełnego zapisu, szczególnie jeśli kod ma być czytelny dla mniej doświadczonych programistów lub osób uczących się jQuery. Zamiast korzystać z anonimowej funkcji, możemy też przekazać funkcję nazwaną, co często poprawia czytelność i ułatwia ponowne wykorzystanie kodu.

```
function readyFunction() {
    // Kod, który zostanie wykonany po załadowaniu
    DOM
    console.log("DOM jest gotowy do manipulacji.");
}

$(document).ready(readyFunction);
$(window).on("load", readyFunction);
```

Listing 2.5 Użycie $(document).ready() i $(window).on("load") wraz z funkcją nazwaną readyFunction

Poniższy przykład pokazuje oba zdarzenia w akcji. W kodzie ładowany jest zewnętrzny adres URL w elemencie `<iframe>`. Funkcja `$(document).ready()` rejestruje moment, gdy DOM jest gotowy, natomiast funkcja `$(window).on("load")` rejestruje załadowanie wszystkich zasobów strony.

```
<!DOCTYPE html>
<html lang="pl">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-
    width, initial-scale=1.0">
    <title>Przykład zdarzeń ładowania</title>
    <script src="https://code.jquery.com/jquery-
    3.7.1.min.js"></script>
    <script>
        $(document).ready(function() {
            console.log("DOM wczytany i gotowy do
            manipulacji.");
        });

        $(window).on("load", function() {
            console.log("Cała strona załadowana
            (łącznie z obrazami i iframe).");
        });
    </script>
</head>
<body>
    <h1>Przykład: $(document).ready() i
    $(window).on("load")</h1>
    <iframe src="http://example.com" width="600"
    height="400"></iframe>
</body>
</html>
```

Listing 2.6 Działanie metod `$(document).ready()` i `$(window).on(„load”)`

Rodzi się pytanie, kiedy stosować `$(document).ready()` i `$(window.on("load")`? Najkorzystniej stosować metodę `$(document).ready()`, gdy chcemy manipulować elementami DOM, dodawać zdarzenia lub dynamicznie zmieniać zawartość strony natychmiast po jej załadowaniu, niezależnie od stanu innych zasobów (np. obrazów).

Natomiast funkcję `$(window).on("load")` stosujemy, gdy potrzebujemy pewności, że wszystkie zasoby strony (np. obrazy, osadzone wideo) zostały w pełni załadowane.

2.3 Selektory

Manipulacja DOM (Document Object Model) jest kluczowym aspektem pracy z nowoczesnymi stronami internetowymi. JQuery upraszcza tę czynność, oferując intuicyjny sposób wybierania i modyfikowania elementów strony. Podstawowym narzędziem do pracy z elementami w jQuery są selektory.

Dzięki nim możemy precyzyjnie wskazywać elementy na stronie, bazując na ich identyfikatorach, klasach, tagach HTML, a nawet bardziej zaawansowanych kryteriach, takich jak atrybuty czy relacje w strukturze DOM. Działają one w oparciu o składnię CSS, co sprawia, że są intuicyjne i proste w użyciu dla osób zaznajomionych z arkuszami stylów.

Selektor to nic innego, jak ciąg znaków używany do wskazywania elementów w strukturze DOM. Na przykład selektor identyfikatora (`#id`) wybiera element o danym identyfikatorze, selektor klasy (`.class`) pozwala wskazać elementy o określonej klasie, a selektory tagów (`tag`) umożliwiają pracę ze wszystkimi elementami danego typu. Taka składnia pozwala efektywnie manipulować elementami strony i tworzyć dynamiczne interakcje, na których chcemy wykonać operacje. Możemy wybierać elementy na podstawie ich identyfikatorów, klas, tagów, atrybutów lub ich relacji z innymi elementami.

Podstawowa zasada działania selektorów jest bardzo prosta - najpierw określamy kryterium wyboru (np. identyfikator, klasa, tag), a następnie jQuery przeszukuje strukturę DOM w poszukiwaniu elementów pasujących do podanego kryterium. Na wybranych elementach można wykonywać różne operacje, takie jak zmiana tekstu, stylów, atrybutów czy obsługa zdarzeń. W poniższym przykładzie zastosujemy selektor ID. Kod znajdziemy w folderze **Aplikacja1**.

```
<!DOCTYPE html>
<html lang="pl">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-
    width, initial-scale=1.0">
    <title>Przykład jQuery</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            text-align: center;
            margin: 50px;
```

```
        }
        #exampleId {
            display: inline-block;
            padding: 10px 20px;
            font-size: 18px;
            background-color: lightblue;
            border: 2px solid blue;
            border-radius: 5px;
            cursor: pointer;
        }
    </style>
    <script src="https://code.jquery.com/jquery-
    3.7.1.min.js"></script>
</head>
<body>
    <div id="exampleId">Kliknij, aby zmienić
    tekst</div>
    <script>
        $(document).ready(function() {
            $("#exampleId").click(function() {
                $(this).text("Nowy tekst w elemencie
                z ID exampleId");
            });
        });
    </script>
</body>
</html>
```

Listing 2.7 Wybór elementu o id #exampleId za pomocą selektora, następnie dzięki metodzie .text() ustawiany jest tekst tego elementu.

Kod przedstawiony powyżej jest prostym przykładem wykorzystania biblioteki jQuery do dynamicznej manipulacji treścią strony internetowej. Struktura tego rozwiązania opiera się na trzech głównych elementach: kodzie HTML odpowiedzialnym za strukturę strony, arkuszu stylów CSS nadającym jej estetyczny wygląd oraz skrypcie JavaScript, który obsługuje interakcję użytkownika.

Strona internetowa zaczyna się od podstawowego dokumentu HTML. W sekcji `<head>` znajduje się deklaracja kodowania znaków UTF-8, co zapewnia poprawne wyświetlanie polskich znaków, oraz znacznik `<meta`

`name="viewport">`, który sprawia, że strona będzie prawidłowo skalowana na różnych urządzeniach, w tym mobilnych. W tej samej sekcji został również załączony arkusz stylów CSS, który wpływa na sposób wyświetlania elementów na stronie.

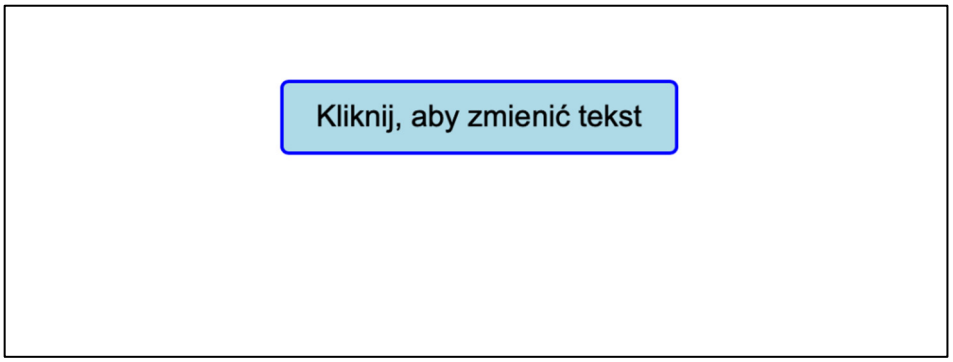
Kod HTML zawiera główny element interaktywny – `<div>` o identyfikatorze `exampleId`, w którym domyślnie znajduje się tekst „Kliknij, aby zmienić tekst". Jest to kluczowy element, na którym będą wykonywane operacje jQuery.

Stylizacja została zaimplementowana wewnątrz sekcji `<style>`. Zastosowano tutaj kilka podstawowych właściwości, które wpływają na wygląd strony oraz interakcyjnego elementu. Cały tekst na stronie został wyśrodkowany, co osiągnięto poprzez ustawienie `text-align: center`.

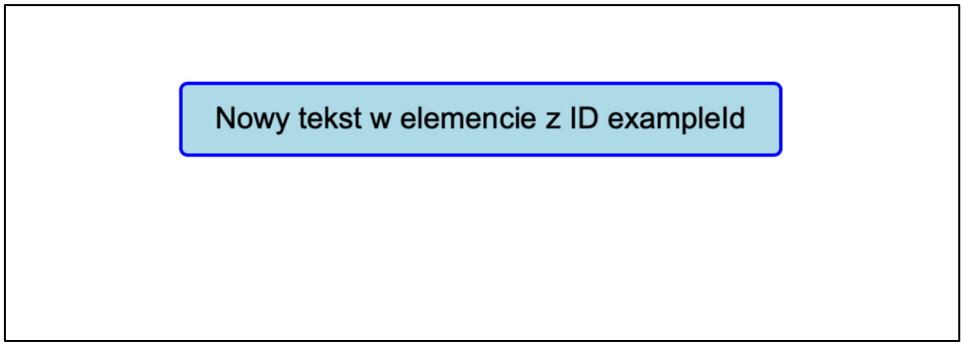
Aby wizualnie wyróżnić interaktywny element, nadano mu jasnoniebieskie tło oraz obramowanie o nieco ciemniejszym odcieniu. Dodatkowo, poprzez zastosowanie `cursor: pointer`, użytkownik ma jasny sygnał, że element jest klikalny (aktywny), co poprawia intuicyjność interakcji.

Najistotniejsza część funkcjonalności została zaimplementowana w skrypcie JavaScript umieszczonym na końcu dokumentu. W pierwszej kolejności do strony została dołączona biblioteka jQuery, co umożliwia korzystanie z jej funkcji. Następnie kod JavaScript został zamknięty w konstrukcji `$(document).ready(function() {...})`, co oznacza, że jego wykonanie nastąpi dopiero po pełnym załadowaniu struktury strony. Jest to ważne, ponieważ zapewnia, że skrypt nie będzie próbował odwoływać się do elementów, które jeszcze nie istnieją w strukturze DOM.

Wewnątrz funkcji obsługującej zdarzenie kliknięcia znajduje się instrukcja `$(this).text("Nowy tekst w elemencie z ID exampleId");`. Oznacza ona, że po kliknięciu na `div` jego zawartość tekstowa zostanie zastąpiona nową wartością. Dzięki wykorzystaniu `$(this)`, skrypt odnosi się bezpośrednio do klikniętego elementu, co czyni kod bardziej uniwersalnym i skalowalnym. Działanie aplikacji w przeglądarce prezentuje się następująco:



Rysunek 2.2 Wygląd aplikacji po uruchomieniu jej w przeglądarce



Rysunek 2.3 Wygląd aplikacji po interakcji z obiektem – ustawiono nowy tekst w divie

Aby wybrać wszystkie elementy, które posiadają określoną klasę CSS, można skorzystać z selektora klasy. Rozważmy przykład, w którym wszystkie elementy posiadające klasę `exampleClass` zmienią kolor tekstu na czerwony. Kod aplikacji znajdziemy w folderze **Aplikacja2**.

```
<!DOCTYPE html>
<html lang="pl">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-
    width, initial-scale=1.0">
    <title>jQuery - Zmiana koloru klasy</title>
    <style>
        .exampleClass {
            font-size: 20px;
            font-weight: bold;
```

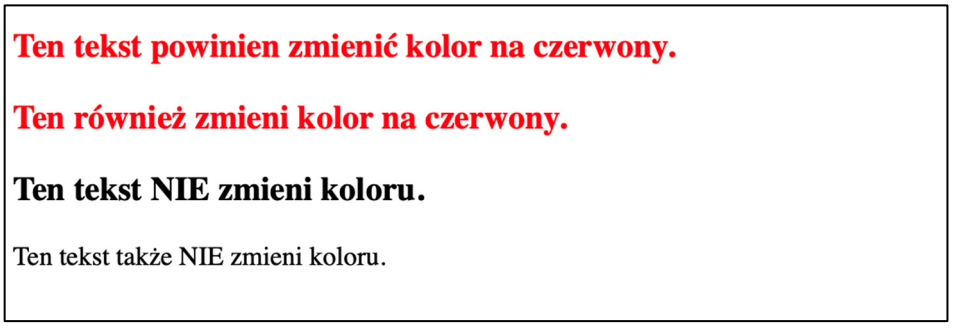
```
        }
        .otherClass {
            font-size: 20px;
            font-weight: bold;
        }
    </style>
    <script src="https://code.jquery.com/jquery-
    3.7.1.min.js"></script>
</head>
<body>
    <p class="exampleClass">Ten tekst powinien
    zmienić kolor na czerwony.</p>
    <p class="exampleClass">Ten również zmieni kolor
    na czerwony.</p>
    <p class="otherClass">Ten tekst NIE zmieni
    koloru.</p>
    <p>Ten tekst także NIE zmieni koloru.</p>
    <script>
        $(document).ready(function() {
            let elements = $(".exampleClass");
            elements.css("color", "red");
        });
    </script>
</body>
</html>
```

Listing 2.8 Wybór elementów o klasie .exampleClass za pomocą selektora, następnie dzięki metodzie .css () ustawiany kolor tekstu w tych elementach na czerwony

Za pomocą tagu `<style>` zdefiniowane są klasy CSS: `.exampleClass` i `.otherClass`, które określają wygląd tekstu, ustawiając rozmiar czcionki na `20px` oraz pogrubienie. Wartość `font-size: 20px` ustawia wielkość czcionki, a `font-weight: bold` zapewnia jej pogrubienie.

W sekcji `<body>` znajdują się cztery akapity `<p>`, z których dwa mają klasę `exampleClass`, jeden klasę `otherClass`, a czwarty nie ma przypisanej klasy. Skrypt jQuery wykonuje operację zmiany koloru tekstu w elementach, które mają klasę `exampleClass`. Po załadowaniu strony (co zapewnia funkcja `$(document).ready()`), wszystkie akapity z klasą `exampleClass` mają zmieniony kolor tekstu na czerwony za pomocą metody `.css("color",`

"red"). Elementy z klasą otherClass oraz akapit bez klasy pozostają bez zmian, co oznacza, że ich kolor tekstu nie zostaje zmodyfikowany. Dzięki temu, tylko tekst w akapitach oznaczonych klasą exampleClass zmienia swój kolor na czerwony.



Rysunek 2.4 Elementy posiadające klasę exampleClass zmienią kolor tekstu na czerwony, w pozostałych przypadkach kolor pozostaje bez zmian.

Możemy również wybrać wszystkie elementy danego typu, np. wszystkie na stronie:

```
let elements = $("div");
```

Listing 2.9 Wybór wszystkich div-ów za pomocą selektora

Ten kod zwraca kolekcję wszystkich elementów , co pozwala na ich dalszą modyfikację, np. zmianę tła czy dodanie efektów animacji. Czasami konieczne jest wybranie różnych typów elementów jednocześnie. Możemy to osiągnąć, łącząc selektory klas, identyfikatorów i tagów:

```
let elements = $(".myClass, #myElement, div");
```

Listing 2.10 Łączenie selektorów – w tym przypadku elementy posiadające klasę myClass, element o identyfikatorze myElement oraz wszystkie divy

W tym przypadku wybierane są, wszystkie elementy posiadające klasę myClass, pojedynczy element o identyfikatorze myElement, wszystkie elementy na stronie. Dzięki temu możemy manipulować wieloma różnymi elementami naraz, bez konieczności powielania kodu.

Nie zawsze chcemy manipulować wszystkimi elementami danego typu. Czasem istotne jest odnalezienie pierwszego lub ostatniego dziecka w danym kontenerze. Rozważmy wybór pierwszego dziecka. Kod programu znajduje się w folderze **Aplikacja3**.

```
<!DOCTYPE html>
<html lang="pl">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-
    width, initial-scale=1.0">
    <title>jQuery - Manipulacja Pierwszym Dzieckiem
    Listy</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            padding: 20px;
            background-color: #f0f0f0;
        }

        ul {
            list-style-type: none;
            padding: 0;
        }

        li {
            font-size: 18px;
            padding: 10px;
            margin-bottom: 5px;
            background-color: #fff;
            border: 1px solid #ccc;
            transition: all 0.3s ease;
        }

        li:first-child {
            font-weight: bold;
            color: blue;
        }

        .highlight {
            background-color: yellow;
            color: red;
            font-weight: bold;
```

```
        }

        .change-font {
            font-family: 'Courier New', Courier,
            monospace;
        }
    </style>
    <script src="https://code.jquery.com/jquery-
    3.7.1.min.js"></script>
</head>
<body>
    <h1>Lista Zakupów</h1>
    <ul>
        <li>Chleb</li>
        <li>Masło</li>
        <li>Jajka</li>
        <li>Ser</li>
        <li>Pomidory</li>
    </ul>

    <script>
        $(document).ready(function() {
            let element = $("ul li:first-child");
            element.css("color", "green");

            element.addClass("highlight");
            element.addClass("change-font");

            element.click(function() {
                $(this).css("background-color",
                "orange");
            });

            $("ul li:not(:first-
            child)").hover(function() {
                $(this).css("background-color",
                "#f4f4f4");
            }, function() {
                $(this).css("background-color",
                "#fff");
```

```
            });
        });
    </script>
</body>
</html>
```

Listing 2.11 Wybór i manipulacja pierwszym elementem w liście nienumerowanej

Listę `<ul>` ustawiono bez domyślnych punktów za pomocą `list-style-type: none`, a każdy element `<li>` otrzymał określone właściwości, takie jak rozmiar czcionki, marginesy oraz tło. Dodatkowo zastosowano animację przejścia, dzięki czemu zmiana właściwości takich jak tło czy kolor odbywa się płynnie.

Zdefiniowano także specjalne style dla pierwszego elementu listy za pomocą selektora `li:first-child`, który zmienia kolor tekstu na niebieski oraz nadaje mu pogrubienie. Kolejne style są zdefiniowane w klasach `.highlight` i `.change-font`, które po dodaniu do elementu zmieniają tło na żółte, tekst na czerwony, a także czcionkę na monospace.

Po załadowaniu strony, skrypt wyszukuje pierwszy element `<li>` na liście za pomocą selektora `$("ul li:first-child")`. Wybiera go do dalszej manipulacji, zmieniając jego kolor na zielony, dodając klasy CSS, które modyfikują jego wygląd (tło na żółte i kolor tekstu na czerwony) oraz zmieniając czcionkę na monospace. Dodatkowo, dodano funkcjonalność, która pozwala na interakcję z tym elementem. Po kliknięciu na pierwszy element zmienia się jego tło na pomarańczowe.

Skrypt zawiera także interakcję z pozostałymi elementami listy. Wykorzystując selektor `$("ul li:not(:first-child)")`, skrypt ustawia efekt, który zmienia tło wszystkich elementów listy, które nie są pierwszymi dziećmi na szare podczas najechania na nie kursorem. Po opuszczeniu kursora tło wraca do pierwotnego koloru.

Rysunek 2.5 Wygląd aplikacji – pierwszy element w liście ma zmieniony kolor tekstu i tła dzięki selektorowi li:first-child

Rysunek 2.6 Wygląd aplikacji – po kliknięciu w pierwszy element listy, kolor tła zmienia się na pomarańczowy.

Podobnie możemy znaleźć ostatnie dziecko listy. Kod programu znajdziemy w folderze **Aplikacja4**.

```
<!DOCTYPE html>
<html lang="pl">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
  initial-scale=1.0">
  <title>Przykład jQuery - Gry komputerowe</title>
  <link rel="stylesheet" href="styles.css">
  <script src="https://code.jquery.com/jquery-
  3.7.1.min.js"></script>
</head>
<body>


  <h1>Lista gier komputerowych</h1>
  <ul id="game-list">
    <li>The Witcher 3: Wild Hunt</li>
    <li>Cyberpunk 2077</li>
    <li>Red Dead Redemption 2</li>
    <li>Hades</li>
  </ul>


  <button id="changeColor">Zmień kolor ostatniej
  gry</button>
  <button id="hideLast">Ukryj ostatnią grę</button>


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

Listing 2.12 Kod HTML aplikacji – plik index.html

```
body {
    font-family: Arial, sans-serif;
    background-color: #f4f4f9;
    text-align: center;
  }


  ul {
    list-style-type: none;
    padding: 0;
  }
```

```
li {
  padding: 10px;
  font-size: 18px;
  margin: 5px 0;
  background-color: #ddd;
}

li:last-child {
  font-weight: bold;
}

button {
  padding: 10px 20px;
  margin-top: 20px;
  font-size: 16px;
  cursor: pointer;
  background-color: #4CAF50;
  color: white;
  border: none;
  border-radius: 5px;
}

button:hover {
  background-color: #45a049;
}
```

Listing 2.13 Styl aplikacji – plik styles.css

```
$(document).ready(function() {
    $("#changeColor").click(function() {
      $("ul li:last-child").css("background-color",
      "#ffcc00");
    });

    $("#hideLast").click(function() {
      $("ul li:last-child").fadeOut(500);
    });
  });
```

Listing 2.14 Funkcjonalność aplikacji z jQuery – plik script.js

W sekcji HTML znajduje się nagłówek oraz lista gier, w której wymienione są tytuły takie jak *The Witcher 3: Wild Hunt*, *Cyberpunk 2077*, *Red Dead Redemption 2* i *Hades*. Pod listą umieszczono dwa przyciski: jeden zmienia kolor tła ostatniego elementu listy, a drugi ukrywa go stopniowo, wykorzystując efekt animacji. Stylizacja strony została zaimplementowana za pomocą CSS.

Lista gier ma prostą, estetyczną formę – każdy element ma jednolite tło, a ostatni element listy jest dodatkowo pogrubiony, aby wyróżniał się wizualnie. Przyciski posiadają zielone tło i efekt podświetlenia po najechaniu kursorem, co zwiększa ich czytelność i ułatwia interakcję. W kodzie JavaScript, korzystając z jQuery, zaimplementowano obsługę zdarzeń kliknięcia dla przycisków.

Pierwszy przycisk zmienia kolor ostatniej gry na żółty, co sprawia, że wyróżnia się na tle pozostałych. Drugi przycisk stopniowo ukrywa ostatnią pozycję z listy, używając efektu `fadeOut`, co powoduje płynne zanikanie elementu. Wszystkie akcje są wykonywane dynamicznie, dzięki czemu użytkownik może na bieżąco modyfikować wygląd listy gier.

Rysunek 2.7 Wygląd aplikacji po uruchomieniu.

Rysunek 2.8 Wygląd aplikacji po uruchomieniu funkcji „Zmień kolor ostatniej gry”.

Rysunek 2.9 Wygląd aplikacji po uruchomieniu funkcji „Ukryj ostatnią grę”.