POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Wydział Informatyki

Katedra Inżynierii Oprogramowania
Inżynieria oprogramowania, procesów biznesowych i baz danych

Dominik Nowicki
Nr albumu s28835

Elastyczny system tworzenia animacji dla portali internetowych

Praca magisterska
napisana pod kierunkiem:

dr inż. Mariusz Trzaska

Warszawa, lipiec, 2024

Streszczenie

Celem pracy magisterskiej jest analiza nowoczesnych technik umieszczania animacji na portalach internetowych oraz ocena ich skuteczności. Praca skupia się na animacjach przyciągających uwagę użytkownika, takich jak banery reklamowe, które muszą być łatwe do modyfikacji bez specjalistycznej wiedzy technicznej. W wyniku analizy obecnych technologii i dostrzeżonych niedostatków, autor opracował własne rozwiązanie – „Elastyczny system do generowania animacji dla portali internetowych". System składa się z trzech elementów: komponentu *webowego* do odtwarzania animacji, zestawu wtyczek dedykowanych animacji, który zapewnia generyczność rozwiązania oraz edytora animacji, który umożliwia tworzenie i podgląd animacji w czasie rzeczywistym użytkownikom nieposiadającym umiejętności programowania. Praca odzwierciedla doświadczenia autora zdobyte podczas pracy zawodowej, podczas której napotkał problemy związane z umieszczaniem i modyfikacją animowanych treści na stronach internetowych.

Słowa kluczowe

Animacja, Web Component, Canvas

Spis treści

Wstęp

Rozdział ten opisuje kontekst pracy, zawiera krótkie omówienie problematyki umieszczania animacji na portalach internetowych. Wyjaśnia motywacje, które skłoniły autora do zajęcia się tym problemem, cel stawiany przed niniejszą pracą oraz opisuje jej rezultaty i organizację.

1.1 Kontekst

Animacje na portalach internetowych pełnią różnorakie funkcje. Ich zadaniem może być wyłącznie uatrakcyjnienie strony pod kątem wizualnym, ale mogą pełnić też inne role, np. ułatwić użytkownikowi zrozumienie sytuacji w jakiej się znajduje lub wytłumaczyć sposób użytkowania strony. Bardzo często służą przyciąganiu uwagi lub wyświetlaniu reklam.

Nie ma jednej oficjalnej kategoryzacji animacji ze względów technicznych. Można dokonać tego na wiele różnych sposobów. Jednym z nich jest podział przeprowadzony na podstawie tego, co jest animowane. Da się wyróżnić następujące kategorie, przy czym należy zaznaczyć, że poniższa lista pewnością nie wyczerpuje wszystkich możliwości [1]:

- animacje pasków postępu (ang. *progres bar* lub *loader*), informujące użytkownika np. o postępie procesu ładownia strony lub pobierania lub wysyłania pliku;
- animacje poszczególnych kontrolek towarzyszące interakcjom użytkownika np. wciśnięcie przycisku, przestawienie przełącznika, otwieranie i zamykanie menu;
- animacje poszczególnych sekcji zawierających treści strony np. przełączanie się zakładek, otwieranie i zamykanie tzw. akordeonu;
- animacje tekstów, obejmujące pojawianie i znikanie poszczególnych liter lub większych fragmentów tekstów;
- animacje dodatkowych obiektów zachęcających użytkownika do działania np. obiekty nałożone na pierwszy plan strony wskazujące, gdzie kierować wzrok albo stanowiące kompletne samouczki do korzystania z portalu internetowego;
- animacje dodatkowych treści, które mogą, ale nie muszą być związane bezpośrednio z portalem internetowym np. animowane banery reklamowe;
- wyświetlanie animacji wideo na pierwszym planie lub animacje tła strony, mające zwykle na celu wizualne uatrakcyjnienie strony lub wywołania określonego odczucia u użytkownika;
- animacje generowane przez dedykowaną aplikację związaną ze specjalistyczną funkcjonalnością np. wizualizacja projektowanego przez użytkownika obiektu lub gry osadzone na stronie internetowej. Aplikacje takie mogą generować animacje rysując poszczególne klatki na elemencie *canvas* lub stanowić byt oddzielny od strony, wytworzony przykładowo przy użyciu technologii *WebAssembly* lub *Unity*.

Warto dopasować odpowiednią technologię wykonania animacji do konkretnego przypadku. Przykładowo, animacje kontrolek można wykonać wyłącznie przy użyciu arkusza stylu CSS, przewodniki po stronie i samouczki wymagają zwykle użycia dodatkowych elementów HTML i manipulacji za pomocą kodu napisanego w języku *JavaScript*, natomiast animacje pełnoekranowe odgrywane są najczęściej z zewnętrznego pliku wideo w postaci rastrowej.

Niniejsza praca koncentruje się na animacjach, które mają za zadanie skupić uwagę użytkownika i przekazać mu pewną treść. W większości przypadków, są to animowane banery reklamowe, przyjmujące postać zamkniętego prostokątnego obszaru ekranu, na którym przedstawiane są zarówno obrazy jak i teksty. W takich przypadkach należy zapewnić możliwość bezproblemowej podmiany

treści, niewymagającej specjalistycznej wiedzy technicznej z zakresu programowania ani ingerencji w kod samego portalu.

1.2 Cel pracy

Celem niniejszej pracy magisterskiej jest:

- przeprowadzenie analizy współczesnych technik umieszczania animacji na portalach internetowych, w tym identyfikacja mocnych i słabych stron poszczególnych rozwiązań oraz ocena ich efektywność w praktycznym zastosowaniu;
- przedstawienie, w oparciu o wyniki analizy, własnej koncepcji wprowadzającej usprawnienia w stosunku do istniejących rozwiązań;
- ułatwienie procesu umieszczania i aktualizacji animacji na stronach internetowych osobom, które nie posiadają wystarczających kompetencji technicznych z obszaru programowania – implementacja prototypu „Elastycznego systemu tworzenia animacji dla portali internetowych".

1.3 Rozwiązania przyjęte w pracy

Przy realizacji części analitycznej pracy wykorzystano dokumentacje techniczne współczesnych rozwiązań pozwalających na umieszczanie animacji na portalach internetowych oraz wiedzę i doświadczenie autora.

Przy realizacji prototypu wykorzystuje się nowoczesne, popularne i ogólnodostępne technologie:

- języki programowania *TypeScript* i *JavaScript* w aktualnych specyfikacjach;
- popularne i darmowe *frameworki*, takie jak: *Angular*, *Stencil.js*;
- współczesne technologie *frontendowe*, takie jak: *Web Components i Canvas API;*
- uznane narzędzia wspomagające pracę programistów, takie jak środowisko uruchomieniowe *Node.js* wraz z menadżerem pakietów NPM, system kontroli wersji *Git*, czy zintegrowane środowisko developerskie *IntelliJ Webstorm.*

Dokumentacje do wszystkich wykorzystanych technologii są ogólnodostępne w Internecie, zarówno w języku polskim jak i angielskim. Szczegółowe informacje na temat zastosowanych narzędzi i technologii znajdują się w rozdziale czwartym.

1.4 Rezultaty pracy

Rezultatem pracy jest analiza istniejących rozwiązań pozwalających na umieszczanie animacji na portalach internetowych, ich ocena, a następnie przedstawienie własnej koncepcji wprowadzającej usprawnienia w stosunku do istniejących rozwiązań. Określenie wymagań co do nowego rozwiązania problemu oparte jest o wyniki uprzedniej analizy obecnych technologii.

Aby osiągnąć założony cel pracy tj. ułatwić proces umieszczania i aktualizacji animacji na stronach internetowych osobom nie posiadającym wystarczających kompetencji technicznych z obszaru programowania, w ramach pracy stworzono prototyp „Elastycznego systemu do generowania animacji dla portali internetowych". Rozwiązanie składa się z trzech elementów:

- *Web Componentu* umieszczanego w prosty sposób na dowolnej stronie internetowej. Komponent ten generuje animacje przy wykorzystaniu dedykowanych wtyczek animacji według przekazanego do komponentu „scenariusza animacji";

- dedykowanych wtyczek animacji wykorzystywanych przez komponent do generowania animacji. System wtyczek zapewnia generyczność rozwiązania i możliwość rozszerzenia go o nowe efekty;
- edytora animacji, który służy do tworzenia i podglądu animacji o zadanych parametrach w czasie rzeczywistym. Edytor wyświetla instrukcje implementacji w formie gotowej do skopiowania oraz umożliwia eksport „scenariusza animacji" w postaci kodu gotowego do użycia. Jego obsługa nie wymaga specjalistycznej wiedzy.

1.5 Organizacja pracy

Praca magisterska jest podzielona na siedem rozdziałów, które odzwierciedlają kolejne etapy realizacji celu pracy:

- pierwszy wprowadza w kontekst pracy, zawiera krótkie omówienie problematyki umieszczania animacji na stronach internetowych, określa cel stawiany przed niniejszą pracą oraz opisuje jej rezultaty i organizację, wreszcie wyjaśnia motywacje, które skłoniły autora do zajęcia się tym problemem;
- drugi ma celu przeprowadzenie analizy istniejących technik umieszczania animacji na portalach internetowych, określenie ich mocnych i słabych stron oraz wyciągnięcie wniosków służących jako podstawa do zaproponowania nowej jakości, którą należy stworzyć;
- trzeci zawiera omówienie koncepcji rozwiązania problemów zidentyfikowanych w rozdziale drugim oraz określenie wymagań stawianych wobec nowego rozwiązania;
- czwarty opisuje zastosowane narzędzia i technologie;
- piąty opisuje szczegółowo sposób implementacji nowego rozwiązania;
- szósty opisuje przykładowy, kompletny scenariusz użycia całego rozwiązania, w tym także sposób korzystania z edytora animacji i ustawiania parametrów animacji składowych;
- siódmy to podsumowanie opisujące mocne i słabe strony zaproponowanego rozwiązania, wnioski wypływające z pracy oraz możliwe kierunki rozwoju prototypu.

1.6 Motywacja

Wybór tematu pracy związany jest z zainteresowaniami autora i jest zbieżny z doświadczeniami zdobytymi podczas jego pracy zawodowej. Autor pracuje jako programista aplikacji działających w środowisku przeglądarki internetowej i spotkał się z problemem umieszczania animowanych treści na stronach internetowych, a następnie modyfikacji ich przez osoby nie posiadające wystarczających kompetencji technicznych z obszaru programowania.

Jak wynika z analizy przeprowadzonej w rozdziale drugim, istniejące na rynku technologie nie wyczerpują wszystkich możliwości zagadnienia umieszczania animacji na portalach internetowych. Wyciągnięte wnioski służą jako podstawa do zaproponowania nowej metody rozwiązania problemu.

2. Analiza istniejących rozwiązań

Jak zauważono w podrozdziale 1.1, animacje są wykorzystywane na stronach internetowych w różnoraki sposób. Niniejsza praca koncentruje się na animacjach, które mają za zadanie skupić uwagę użytkownika i przekazać mu pewną treść. Są to najczęściej animowane banery reklamowe, przyjmujące postać zamkniętego prostokątnego obszaru ekranu, na którym użytkownikowi przedstawiane są zarówno obrazy jak i teksty. Rozdział ten ma celu przeprowadzenie analizy istniejących technologii pozwalających na umieszczanie tego typu animacji na portalach internetowych, określenie mocnych i słabych stron poszczególnych rozwiązań oraz wyciągnięcie wniosków służących jako podstawa do zaproponowania nowej jakości, którą należy stworzyć.

2.1 Istniejące rozwiązania

Animacje przedstawiające użytkownikowi obrazy i teksty na zamkniętym, prostokątnym obszarze ekranu można uzyskać przy pomocy różnych technik [1]:

- umieszczenia na stronie internetowej gotowych plików animacji;
- animacji wektorowych generowanych przy użyciu elementu HTML *svg;*
- animacji elementów DOM przy użyciu arkusza stylu CSS;
- animacji elementów DOM przy użyciu kodu *JavaScript*;
- animacji generowanych przy użyciu elementu HTML *canvas*.

Na potrzeby umieszczania animowanych treści na stronach internetowych szeroko wykorzystywana była również technologia *Flash*, dostarczana przez firmę *Adobe.* Rozwiązanie polegało na umieszczeniu na stornie internetowej animacji w dedykowanym formacie, wymagało jednak to od użytkownika zainstalowania odpowiedniej wtyczki do przeglądarki internetowej. Warto dodać, że animacje powstające przy użyciu tej technologii mogły być również interaktywne. Jednak technologia ta ustąpiła nowszym, bardziej optymalnym rozwiązaniom zgodnym ze standardem HTML5 [2]. Projekt został zamknięty z końcem roku 2020 [3].

Na potrzeby wyświetlania banerów reklamowych często wykorzystywana jest również technologia *iframe*. Polega ona na umieszczeniu na portalu internetowym elementu HTML *iframe,* wewnątrz którego wyświetla jest zawartość innej strony internetowej, zawierającej docelowe treści. Treści te mogą być animowane przy wykorzystaniu dowolnej z wymienionych technik, mogą to być przykładowo obrazy lub animacje elementów DOM przy jego użyciu kodu *JavaScript*, tak więc technologia *iframe* nie ma na celu zastąpienia wyszczególnionych technik animacji, a jedynie je uzupełnia [4]. Sposób ten wykorzystuje m.in. rozwiązanie *Google Ads* [5].

2.1.1 Umieszczenie na stronie internetowej gotowych plików animacji

Najbardziej oczywistym i najprostszym sposobem umieszczenia animowanych treści na stronie internetowej jest wyświetlenie wcześniej przygotowanej animacji, zapisanej w oddzielnym pliku. Standard HTML umożliwia odegranie animacji w formacie zarówno wektorowym, jak i rastrowym.

Wyświetlenie na stronie internetowej animacji wektorowej, zapisanej w pliku w formacie SVG, odbywa się przy użyciu elementu HTML *image.* Wymaga podania źródła pliku jako wartości atrybutu *src,* umożliwia także określenie wysokości i szerokości elementu oraz podania atrybutu *alt,* czyli informacji pojawiającej się, jeżeli zasób nie zostanie pobrany [6]. Użycie zostało zademonstrowane na listingu 2.1.

Istnieje również możliwość generowania animacji wektorowej z wykorzystaniem elementu *svg*. Taki sposób nie prowadzi do wyświetlenia animacji z gotowego pliku, wykracza zatem poza opisywaną tutaj technikę i został omówiony oddzielnie w punkcie 2.1.2.

Listing 2.1. Przykładowy sposób wyświetlenia animacji zapisanej w formacie SVG przy użyciu elementu HTML *image*. Źródło: opracowanie własne

```
<img width="200" height="100" src="/assets/animation.svg" alt="…" />
```

Użycie elementu HTML *image*, umożliwia także wyświetlenie animowanych plików w formatach rastrowych GIF, APNG lub WebP [6]. Użycie jest analogiczne jak w przypadku wyświetlenia pliku w formacie SVG pokazanego na listingu 2.1.

Wraz ze specyfikacją HTML5 udostępniony został element HTML *video,* który jest w stanie odtworzyć animacje zapisane w różnych popularnych formatach takich jak: WebM, MP4 czy OGG [7]. Element ten działa na podobnej zasadzie co element *image* i wyświetla zawartość wskazanego pliku, ale może być bardziej rozbudowany. Poza atrybutami odpowiadającymi za rozmiar elementu udostępnia również takie odpowiadające za włączenie lub wyłączenie opcji pokazywania panelu kontrolującego przebieg animacji, oraz atrybutami pozwalającymi na zapętlenie animacji lub wyciszenie dźwięków pochodzących z nagrania. Budowa elementu umożliwia też podanie „zapasowych" lokalizacji plików. Użycie elementu HTML *video* zaprezentowano na listingu 2.2.

Listing 2.2. Przykładowy sposób wyświetlenia pliku wideo.
Źródło: opracowanie własne za [7]

```
<video width="200" controls loop muted>
  <source src="/assets/wideo.webm" type="video/webm"/>
  <source src="/assets/wideo.mp4"  type="video/mp4" />
</video>
```

Po przenalizowaniu możliwości i przykładowych zastosowań tego rozwiązania, zidentyfikowano następujące jego zalety:

- poprawność wyświetlenia animacji na każdym urządzeniu;
- nieograniczone możliwości kreacji artystycznej i pełna kontrola grafika nad efektem końcowym;
- prostota implementacji.

Dostrzeżono też wady tej technologii:

- każda zmiana treści wymaga wygenerowania i umieszczenia na serwerze nowego pliku;
- duże rozmiary plików w przypadku animacji rastrowych;
- brak możliwości dostosowania treści do poszczególnego odbiorcy (grafika jest przygotowana wcześniej i nie może być generowana na bieżąco w oparciu o przekazane parametry np. ustawienia językowe);
- brak możliwości indeksowania treści grafiki rastrowej.

Przedstawione elementy HTML typu *image* czy *video* oczekują podanego źródła – adresu konkretnego pliku. W celu zmiany wyświetlanej treści, należy każdorazowo wygenerować nowy plik i zapisać go na serwerze w miejsce pliku istniejącego i wskazanego jako źródło albo umieścić na serwerze nowy plik i następnie zmienić wartość atrybutu wskazującego źródło. Pierwszy sposób nie wymaga zmiany kodu aplikacji, a jedynie dostępu do przestrzeni, w której znajduje się plik oraz odpowiednich uprawnień do jego podmiany. Drugi ze sposobów wymaga zmiany kodu źródłowego w aplikacji internetowej. Dla osoby nieposiadających wystarczających umiejętności informatycznych istnieje możliwość użycia systemu zarządzania treścią (ang. *Content Management System,* w skrócie CMS), który w łatwy i przyjazny sposób umożliwi zmianę treści wyświetlanych użytkownikowi.

2.1.2 Animacje wektorowe generowane przy użyciu elementu HTML *svg*

Jak wspomniano w pkt. 2.1.1, istnieje również możliwość generowania animacji wektorowej na bieżąco z poziomu kodu aplikacji internetowej. Odbywa się to z wykorzystaniem elementu *svg*, przy użyciu następujących interfejsów [8]:

- *SVGAnimateElement* – używany do zmiany wartości atrybutów wybranego obiektu w czasie np. zmiana jego rozmiarów, położenia, koloru lub przezierności;
- *SVGAnimateMotionElement* – używany do obrotu i przemieszczania wskazanego obiektu po zdefiniowanej ścieżce;
- *SVGAnimateTransformElement* – używany do przemieszczania, skalowania, obrotu lub zakrzywiania obiektu (ang. *skew*).

Każdy z nich oferuje inne możliwości i służy do uzyskania różnych efektów. Ich przykładowe wykorzystanie pokazano odpowiednio na listingach 2.3, 2.4 i 2.5. Kod generujący animacje może oczywiście zostać napisany przez programistę, ale zwykle jest opracowany przez grafika za pomocą dedykowanych aplikacji graficznych, takich jak na przykład *Adobe Express* albo *Svagator* i eksportowany jako plik lub kod gotowy do użycia.

Listing 2.3. Przykładowy sposób generowania animacji wektorowej przy użyciu interfejsu *SVGAnimateElement*. Źródło: [9]

```
<svg viewBox="0 0 10 10" xmlns="http://www.w3.org/2000/svg">
  <rect
    width="10"
    height="10">
    <animate
      attributeName="rx"
      values="0;5;0"
      dur="10s"
      repeatCount="indefinite" />
  </rect>
</svg>
```

Listing 2.4. Przykładowy sposób generowania animacji wektorowej przy użyciu interfejsu *SVGAnimateMotionElement*. Źródło: [10]

```
<svg viewBox="0 0 200 100" xmlns="http://www.w3.org/2000/svg">
  <path
    fill="none"
    stroke="lightgrey"
    d="M20,50 C20,-50 180,150 180,50 C180-50 20,150 20,50 z" />
  <circle r="5" fill="red">
    <animateMotion
      dur="10s"
      repeatCount="indefinite"
      path="M20,50 C20,-50 180,150 180,50 C180-50 20,150 20,50 z" />
  </circle>
</svg>
```

Listing 2.5. Przykładowy sposób generowania animacji wektorowej przy użyciu interfejsu *SVGAnimateTransformElement.* Źródło: [11]

```
<svg viewBox="0 0 120 120" xmlns="http://www.w3.org/2000/svg">
  <polygon points="60,30 90,90 30,90">
    <animateTransform
      attributeName="transform"
      attributeType="XML"
      type="rotate"
      from="0 60 70"
      to="360 60 70"
      dur="10s"
      repeatCount="indefinite" />
  </polygon>
</svg>
```

Po przenalizowaniu możliwości i przykładowych zastosowań tego rozwiązania, zidentyfikowano następujące jego zalety:

- poprawność wyświetlenia animacji na każdym urządzeniu;
- szerokie możliwości twórcze i pełna kontrola grafika nad ostatecznym efektem;
- prostota implementacji;
- rozwiązanie optymalne pod względem zużywania zasobów – animacje takie zajmują bardzo mało miejsca, a ich odgrywanie nie jest obciążające dla przeglądarki.

Największą dostrzeżoną wadą tego sposobu animacji jest fakt, że każda potrzeba zmiany treści wymaga wygenerowania nowego kodu i aktualizację aplikacji internetowej. Ta technologia najlepiej sprawdzi się w przypadkach, gdy zmiany treści nie będą musiały być przeprowadzane zbyt często ani też przez osoby nie posiadające odpowiednich kompetencji programistycznych.

2.1.3 Animacje elementów DOM przy użyciu arkusza stylu CSS

Każdy element DOM, czyli obiekt będący częścią strony internetowej, posiada szereg parametrów definiujących jego właściwości. Zestaw właściwości zależy od typu elementu HTML, który obiekt reprezentuje. Przykładowo może być to kolor i wielkość tekstu, lub szerokość i wysokość obrazka. W określonych, ale nie wszystkich, przypadkach może to być też pozycja obiektu na ekranie. Właściwości, które odpowiadają za wygląd lub położenie obiektu można określać przy użyciu arkusza stylu CSS, a także definiować ich zmianę w miarę upływu czasu, osiągając w ten sposób efekt animacji.

Animacje generowane przy użyciu arkusza stylu CSS wykonuje się poprzez zdefiniowanie zestawu zmieniających się parametrów i przypisanie ich do obiektu. Definicja rozpoczyna się od słowa kluczowego *@keyframes*, po nim występuje unikalna nazwa zestawu oraz zestawy oczekiwanych wartości poszczególnych właściwości na różnych etapach animacji. Na koniec, do elementu HTML przypisywany jest zdefiniowany zestaw jako właściwość *„animation-name”*.

Za przykład niech posłuży animacja pokazana na listingu 2.6. Powoduje ona obrót znajdującego się na ekranie zielonego kwadratu o długości boku 20 pikseli. Za pomocą słowa kluczowego *@keyframes* oraz unikalnej nazwy, tutaj *„rotate-animation”*, określony zostaje zestaw zmieniających się parametrów. Na przedstawionym przykładzie jest to właściwość *„transform: rotate”* określająca obrót elementu, zmieniająca się z 0 (na początkowym etapie animacji, ang. *from*) na 90 stopni (na końcowym etapie animacji, ang. *to*). Następnie określony jest styl elementu HTML *div* o identyfikatorze *„green-square”*. Fragment *„animation-name: rotate-animation;”* oznacza, że przypisany zostaje mu zestaw oczekiwanych zmian, zdefiniowany pod nazwą *„rotate-animation”*. Zmiany te mają ukończyć się w czasie 1s – mówi o tym fragment *„animation-duration: 1s;”*.

Listing 2.6. Arkusz stylu CSS definiujący prostą animację obrotu zielonego kwadratu.
Źródło: opracowanie własne

```
@keyframes rotate-animation {
  from: {
    transform: rotate(0);
  }
  to {
    transform: rotate(90deg);
  }
}

div#green-square {
  background: green;
  display: block;
  width: 20px;
  height: 20px;
  animation-duration: 1s;
  animation-name: rotate-animation;
}
```

Animacje mogą obejmować zmianę wielu parametrów jednocześnie, a także definiować różne zmiany na poszczególnych etapach animacji, które podaje się w procentach – ilustruje to listing 2.7.

Listing 2.7. Arkusz stylu CSS definiujący animację zmiany kolorów i położenia jednocześnie.
Źródło: [12]

```
@keyframes animation {
  0%   {background-color:red; left:0px; top:0px;}
  25%  {background-color:yellow; left:200px; top:0px;}
  50%  {background-color:blue; left:200px; top:200px;}
  75%  {background-color:green; left:0px; top:200px;}
  100% {background-color:red; left:0px; top:0px;}
}
```

Animacje można opóźniać, ustawiając na animowanym obiekcie właściwość *„animation-delay”* oraz nadając jej wartość w sekundach, a także ustawiać, ile razy ma zostać odegrana – za pomocą właściwości *„animation-iteration-count”* lub wskazać kierunek animacji za pomocą *„animation-direction”* [12]. Istnieje też możliwość przypisaniu wielu animacji do jednego obiektu [13]. Sposób w jaki można to zrobić zaprezentowany został na listingu 2.8. Warto zauważyć, że do każdej z animacji można niezależnie przypisać różne czasy trwania, a także ilość odtworzeń.

Listing 2.8. Przypisanie dwóch animacji jednocześnie za pomocą arkusza stylu CSS.
Źródło: opracowanie własne za [13]

```
animation-name:            animation1, animation2;
animation-duration:        3,          5s;
animation-iteration-count: 2,          infinite;
```

Elementy DOM, do których przypisane są animacje zdefiniowane przy użyciu arkusza stylu CSS emitują zdarzenia (ang. *event*):

- rozpoczęcia się animacji (ang. *animationstart*);
- zakończenia się animacji (ang. *animationend*);
- wykonania się kolejnej iteracji animacji (ang. *animationiteration*).

Przy pomocy kodu napisanego w języku *JavaScript* można nasłuchiwać i reagować na tego typu zdarzenia.

Rozwiązanie polegające na umieszczeniu na stronie niezależnych elementów DOM np. tekstów i obrazków, a następnie animowanie ich przy użyciu arkusza stylu CSS ma wiele zalet:

- każdy z elementów DOM może obsługiwać własne zdarzenia (np. kliknięcie na tekst);
- możliwość zróżnicowana treści dla poszczególnych odbiorców lub w zależności od różnych sytuacji (przykładowo można podmienić treść tekstu w zależności od wybranej wersji językowej strony, nie zmieniając pozostałych elementów takich jak obrazy, pozostawiając także bez zmian zaplanowany przebieg animacji);
- możliwość szybkiej podmiany treści, także przez osoby nie posiadające kompetencji programistycznych np. poprzez dedykowany system CMS (jednak dodawanie nowych efektów animacji do istniejącej aplikacji internetowej nie jest już takie proste);
- wyższa jakość elementów takich jak teksty lub kształty wektorowe niż obrazów rastrowych;
- animacje przy użyciu arkusza styli CSS są optymalizowane przez silnik renderujący i obciążają przeglądarkę mniej niż animacje za pomocą kodu *JavaScript,* ponadto nie blokują wątku wykonywania logiki ani nie są wykonywane, gdy karta przeglądarki działa w tle [13];
- animacje elementów DOM mają dużo mniejszą tzw. „wagę” (liczoną w jednostkach pamięci jak np. kB), niż animacje rastrowe zawarte w plikach.

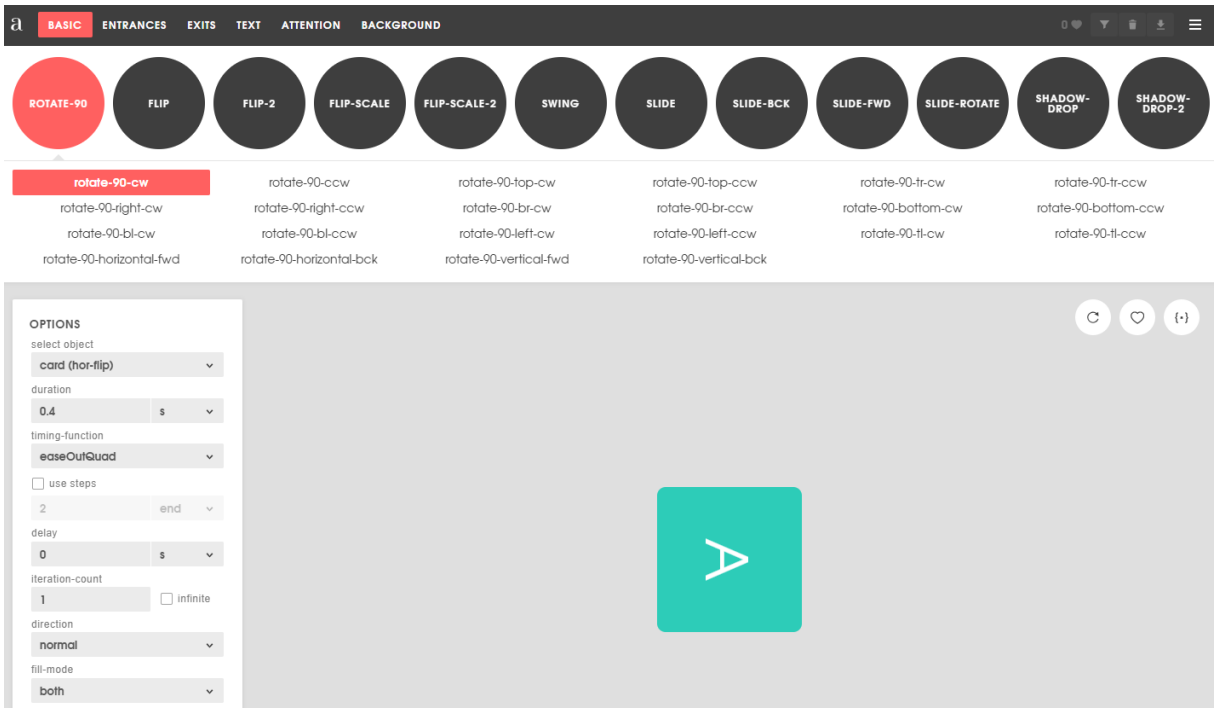
Technologia ta posiada również ograniczenia:

- ograniczone efekty jakie można uzyskać przy zastosowaniu tej technologii;
- zmiana sposobu animacji zapisanej w arkuszu styli CSS wymaga ingerencji w kod aplikacji internetowej – wprowadzenie zmian w tym zakresie przez osobę nieposiadającą kompetencji programistycznych może być trudne lub niemożliwe;
- trudność w przycinaniu ruchomych elementów do wskazanego obszaru (wada przy założeniu, że istotne jest zamknięcie animowanych elementów w wyznaczonym prostokątnym obszarze);
- programowanie animacji może wymagać współpracy projektanta strony i programisty.

Na rynku istnieje wiele bibliotek udostępniających przygotowane animacje wykorzystujące CSS. Mogą stanowić inspiracje dla projektantów stron i są jednocześnie zbiorem fragmentów kodu gotowego do użycia przez programistów aplikacji internetowych. Jednym z przykładów jest biblioteka *Animista* stworzona i udostępniona na licencji *FreeBSD* przez Anę Travas [14]. Strona internetowa prezentująca możliwości przygotowanych efektów znajduje się pod oficjalnym adresem projektu *Animista* [14]. Jej wygląd pokazano na rysunku 2.1. Górna część ekranu służy do odnalezienia właściwego efektu. Efekty podzielone są na kategorie, które użytkownik może wybrać poprzez wciśnięcie jednego z przycisków znajdujących się na pasku tytułowym aplikacji:

- podstawowe i ogólne (ang. *basic*);
- efekty wejścia (ang. *entrances*);
- efekty wyjścia (ang. *exits*);
- efekty dotyczące tekstu (ang. *text*);
- efekty przykuwające uwagę użytkownika (ang. *attention*);
- efekty tła (ang. *background*).

Poniżej kategorii, znajduje się lista typów efektów, przedstawiona jako szereg dużych okrągłych przycisków (na rysunku 2.1 wybrany został typ *„rotate-90”*), a pod nimi znajduje się lista konkretnych animacji. Po wybraniu jednego z efektów (na rysunku 2.1 wybrany został *„rotate-90-cw”*), element widoczny na środku ekranu (na rysunku jest to kwadrat z literką „A”) poddany jest wybranej animacji. Po lewej stronie ekranu znajduje się lista dodatkowych opcji.

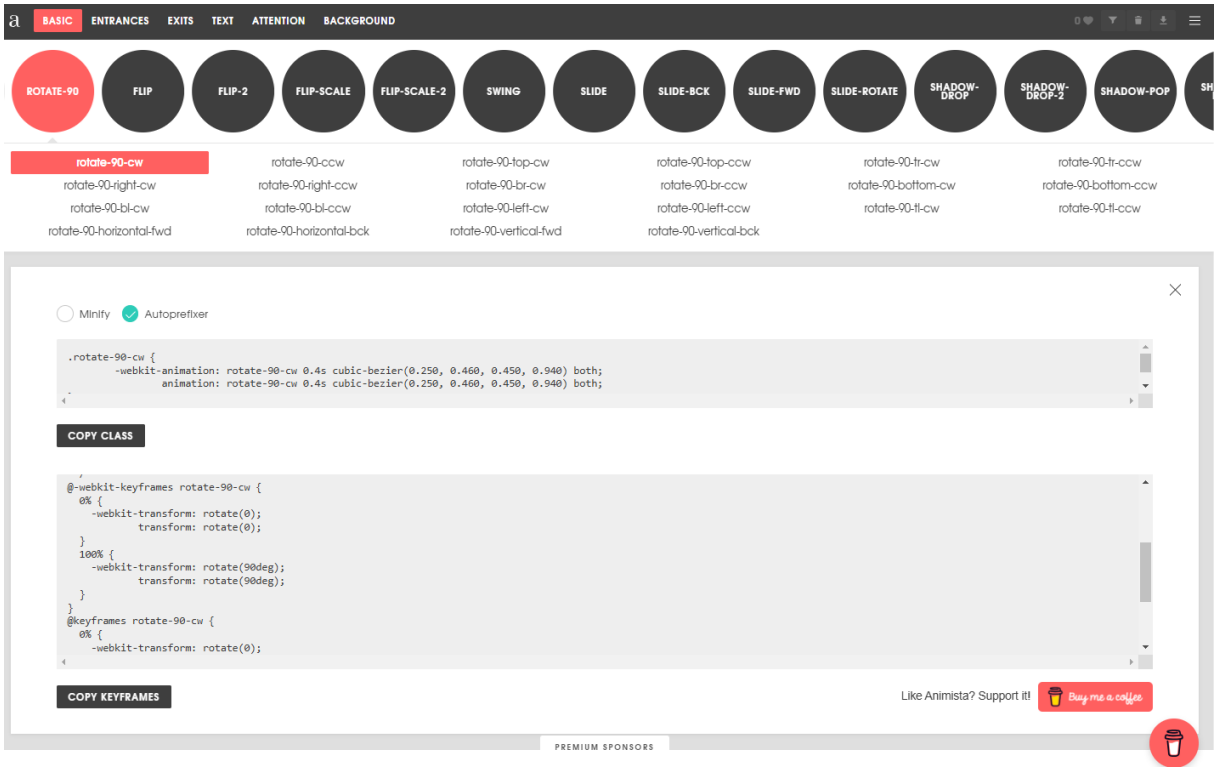


Rysunek 2.1.Widok aplikacji internetowej prezentującej efekty biblioteki *Animista.* Źródło: [14]

W celu wykorzystania wybranego efektu animacji należy nacisnąć przycisk znajdujący się w prawym górnym rogu szarego tła, oznaczony tekstem „{*}”. Przycisk ten pokazuje kod, jaki należy wykorzystać w aplikacji internetowej, aby uzyskać oczekiwany efekt. Na listingu 2.9 zaprezentowano kod dostarczany przez bibliotekę *Animista* w przypadku wyboru efektu obrotu o 90 stopni zgodnie z kierunkiem zegara tj. efektu *„rotate-90-cw”*. Na rysunku 2.2 zaprezentowano widok strony w trybie wyświetlania kodu gotowego do użycia.

Listing 2.9. Arkusz stylu CSS definiujący animację obrotu elementu o 90 stopni. Źródło: [14]

```
@keyframes rotate-90-cw {
  0% {
    transform: rotate(0);
  }
  100% {
    transform: rotate(90deg);
  }
}

.rotate-90-cw {
      animation: rotate-90-cw 0.4s cubic-bezier(0.250, 0.460, 0.450, 0.940) both;
}
```



Rysunek 2.2. Widok aplikacji internetowej prezentującej efekty biblioteki *Animista* w trybie wyświetlania kodu do użycia. Źródło: [14]

Użycie narzędzi tego typu zdecydowanie ułatwia pracę programisty oraz współpracę pomiędzy nim a projektantem strony, niestety nie jest w stanie rozwiązać największych problemów tej technologii:

- ograniczonego zakresu efektów jakie można uzyskać przy zastosowaniu tej technologii;
- zmiana sposobu animacji zapisanej w arkuszu styli CSS wymaga ingerencji w kod aplikacji internetowej – wprowadzenie zmian w tym zakresie przez osobę nieposiadającą kompetencji programistycznych może być trudne lub niemożliwe;
- trudność w przycinaniu ruchomych elementów do wskazanego obszaru (przy założeniu, że istotne jest zamknięcie animowanych elementów w wyznaczonym prostokątnym obszarze);
- programowanie animacji może wymagać współpracy projektanta strony i programisty.

2.1.4 Animacje elementów DOM przy użyciu kodu JavaScript

Bardziej zaawansowane animacje elementów strony internetowej można wykonać przy pomocy języka programowania *JavaScript*. Podobnie jak w przypadku zastosowania techniki animacji przy użyciu arkusza stylu CSS, także animacje przy użyciu kodu JS polegają na zmianie właściwości poszczególnego elementu strony internetowej. Każdym obiektem można manipulować niezależenie, na wiele różnych sposobów (np. przesuwać, skalować, zmieniać przezroczystość, kolor). Niepodważalną zaletą tej technologii jest możliwość reakcji na wszelkie zdarzenia (w tym także sieciowe) lub interakcje użytkownika (np. kliknięcia myszą lub dotknięcia ekranu, przewijanie ekranu), przy czym poszczególne elementy mogą reagować na różne zdarzenia niezależnie. Możliwości rozbudowania logiki animacji, ich wzajemnych zależności oraz kontroli programisty nad animacją są praktycznie nieograniczone.

Dla wyjaśnienia sposobu działania tej technologii można posłużyć się przykładem z poprzedniego punktu tj. wykonaniem animacji obrotu zielonego kwadratu znajdującego się na ekranie. Jedno z możliwych rozwiązań takiego zadania za pomocą kodu *JavaScript* zaprezentowano na listingu 2.10.

W celu animacji utworzona została pętla wykonywana we wskazanych odstępach czasu, za każdą jej kolejną iteracją zwiększana jest wartość właściwości *„transform: rotate”*, wobec czego element się obraca. Gdy wartość obrotu wynosi 90 pętla zostaje przerwana. Aby animacja trwała jedną sekundę, programista musi obliczyć co jaki czas powinna wykonywać się kolejna iteracja, tak aby pełny obrót o 90 stopni trwał właśnie jedną sekundę. Na podanym przykładzie zastosowano uproszczoną metodę polegającą na obliczeniu interwału wykonywania pętli z proporcji. W rzeczywistości sprawa jest jeszcze trudniejsza i precyzyjnie określenie upływającego czasu należałoby wykonać w oparciu o stały punkt na osi czasu, zwracanego przykładowo przez metodę *performance.now* [15]. Sposób ten wydaje się więc bardziej skomplikowany niż analogiczne rozwiązanie przedstawione w poprzednim punkcie.

Listing 2.10. Animacja obrotu elementu o 90 stopni wykonana za pomocą kodu *JavaScript*. Źródło: opracowanie własne za [16]

```
let object = document.getElementById('green-square');
let intervalId = null;
let rotation = 0;
intervalId = setInterval(nextFrame, (1000/90));
function nextFrame() {
  if (rotation === 90) {
    clearInterval(intervalId);
  } else {
    rotation++;
    object.style.transform = 'rotate(' + rotation + 'deg)';
  }
}
```

Rozwiązanie polegające na umieszczeniu na stronie niezależnych elementów DOM np. tekstów i obrazków, a następnie animowanie ich przy użyciu języka *JavaScript* ma wiele zalet podobnych do rozwiązania opartego na animacjach przy użyciu arkusza stylu CSS, opisanego w punkcie 2.1.3:

- każdy z elementów DOM może obsługiwać własne zdarzenia (np. kliknięcie na tekst);
- możliwość zróżnicowania treści dla poszczególnych odbiorców lub w zależności od różnych sytuacji (przykładowo można podmienić treść tekstu w zależności od wybranej wersji językowej strony, nie zmieniając pozostałych elementów takich jak obrazy, pozostawiając także bez zmian zaplanowany przebieg animacji);
- możliwość szybkiej podmiany treści, także przez osoby nie posiadające kompetencji programistycznych poprzez dedykowany system CMS (ale dodawanie nowych efektów animacji do działającej aplikacji internetowej nie jest już takie proste);
- wyższa jakość elementów takich jak teksty lub kształty wektorowe niż obrazów rastrowych;
- animacje elementów DOM mają dużo mniejszą tzw. „wagę” (liczoną w jednostkach pamięci jak np. kB), niż animacje rastrowe zawarte w plikach.

Technologia ta posiada również podobne ograniczenia:

- możliwość kreacji artystycznej jest bardziej ograniczona niż w przypadku animacji rastrowych, ale szersza niż w przypadku animacji obiektów DOM przy użyciu arkusza stylu CSS;
- zmiana sposobu animacji zapisanej w kodzie *JavaScript* wymaga zwykle ingerencji w kod aplikacji internetowej;
- trudność w przycinaniu ruchomych elementów do wskazanego obszaru (przy założeniu, że istotne jest zamknięcie animowanych elementów w wyznaczonym prostokątnym obszarze);
- programowanie animacji może wymagać współpracy projektanta strony i programisty;

- animacje przy użyciu kodu *JavaScript*, w przeciwieństwie do animacji za pomocą arkusza styli CSS, obciążają przeglądarkę, blokują wątek wykonywania logiki (co może skutkować zacinaniem się animacji), ponadto w większości wypadków logika działa także, gdy karta przeglądarki działa w tle.

Warto jeszcze raz podkreślić najważniejsze różnice pomiędzy technologią animacji przy użyciu arkusza stylu CSS, a animacji przy użyciu kodu *JavaScript*. Największą przewagą tej drugiej jest większa możliwość rozbudowania logiki animacji i jej kontroli, odbywa się to niestety kosztem wydajności i obciążenia wątku obsługującego także inne funkcjonalności strony internetowej (w uproszczeniu można przyjąć, że przeglądarki internetowe używają wyłącznie jednego wątku do wykonywania całości logiki napisanej w kodzie *JavaScript,* w tym także animacji).

Na rynku istnieje wiele bibliotek ułatwiających tworzenie animacji przy użyciu języka *JavaScript*. Od lat najpopularniejszym narzędziem wspomagającym tworzenie animacji przy użyciu kodu *JavaScript* jest biblioteka *jQuery* [17]. Jest ona udostępniana przez *OpenJS Foundation* w ramach licencji MIT [18], jest więc darmowa i może być używana także w projektach komercyjnych.

Przykład obrotu obiektu o 90 stopni przy użyciu biblioteki *jQuery* prezentuje listing 2.11. Kod jest wyraźnie mniej skomplikowany niż skrypt napisany od postaw zaprezentowany wcześniej na listingu 2.10. Działa jednak na tej samej zasadzie. Funkcja dostarczona przez bibliotekę wykonuje w pętli podmianę wartości dla wskazanych właściwości obiektu. Podobnie jak na wcześniejszych przykładach jest to właściwość *„transform: rotate”*, a wartości zmieniają się od 0 do 90. Przekazany jest także argument określający oczekiwana długość trwania animacji (ang. *duration*). W efekcie powoduje to oczekiwany obrót obiektu w czasie 1s.

Listing 2.11. Animacja obrotu elementu o 90 stopni wykonana przy użyciu biblioteki *jQuery*.
Źródło: opracowanie własne za [19]

```
$('#green-square').animate({
    i: 90
  }, {
    step: function(i) {
      $(this).css('transform','rotate('+i+'deg)');
    },
    duration: 1000
});
```

Właściwości, które przyjmują wartości liczbowe, takie jak np. pozycja lub rozmiary, animuje się jeszcze prościej. Na listingu 2.12 zaprezentowano animację wysokości kwadratu do 300px w czasie 1s, a następnie jej powrót do 20px w czasie kolejnej sekundy.

Listing 2.12. Zmiana wysokości elementu w czasie animowana przy użyciu biblioteki *jQuery*.
Źródło: opracowanie własne

```
$('#green-square')
  .animate({height: "300px"}, {duration: 1000})
  .animate({height: "20px"}, {duration: 1000});
```

Co istotne, i zostało już uwidocznione na listingach, programista używający biblioteki *jQuery* nie musi martwić się o przerywanie pętli w odpowiednim momencie, ani o konieczność przeliczenia oczekiwanego czasu animacji na opóźnienia wykonywania pętli jak to miało miejsce we wcześniejszym przypadku. Biblioteka robi to za niego. Wystarczy podać oczekiwany czas trwania animacji (ang.

duration). Mechanizm używany przez bibliotekę *jQuery* do obliczania postępu animacji pokazano na listingu 2.13. Ponadto, animacje korzystają z mechanizmu synchronizacji klatek z częstotliwością odświeżania ekranu, czyli metody *window.requestAnimationFrame* oferowanej przez współczesne przeglądarki [19].

Listing 2.13. Sposób obliczania postępu animacji wykorzystywany przez bibliotekę *jQuery*. Źródło: kod źródłowy biblioteki *jQuery* [20]

```
tick = function() {
  if (stopped) {
    return false;
  }
  var currentTime = fxNow || createFxNow(),
  remaining = Math.max(
    0, animation.startTime +  animation.duration    - currentTime),
  percent = 1 - (remaining / animation.duration || 0),
  index = 0,
  length = animation.tweens.length;

  for (; index < length; index++) {
    animation.tweens[index].run(percent);
  }

  deferred.notifyWith(elem, [animation, percent, remaining]);

  // If there's more to do, yield
  if (percent < 1 && length) {
    return remaining;
  }
  …
}
```

Listing 2.14. Fragment kodu bilbioteki *jQuery* odpowiadający za wybór metody powtarzania planowania kolejnych klatek animacji. Źródło: kod źródłowy biblioteki *jQuery* [20]

```
function schedule() {
  if (document.hidden === false && window.requestAnimationFrame) {
    window.requestAnimationFrame(schedule);
  } else {
    window.setTimeout(schedule, jQuery.fx.interval);
  }
  jQuery.fx.tick();
}
```

Niestety, *OpenJS Foundation*, twórca biblioteki *jQuery*, nie dostarcza żadnego dedykowanego narzędzia, za pomocą którego użytkownik mógłby przetestować parametry dostępnych animacji, a następnie użyć gotowego kodu. Jednak z racji dużej popularności tej biblioteki w Internecie dostępnych jest bardzo wiele przykładów i propozycji użycia.

Innym popularnym rozwiązaniem wspierającym tworzenie animacji za pomocą kodu *JavaScript* jest biblioteka GSAP [21]. Jest to rozwiązanie komercyjne, sprzedawane przez amerykańską firmę *GreenSock Inc.* Jest wysoko oceniane [22] i bardzo rozbudowane. Przy jej pomocy można animować

każdą z właściwości elementu HTML, to znaczy zmieniać wartości jego atrybutu w miarę upływu zadanego czasu. Mechanizm działania obejmuje każdy dowolny atrybut, nawet taki który dotychczas nie istniał na obiekcie i nie jest wykorzystywany do określania sposobu jego wyświetlania przez przeglądarkę. Pod oficjalnym adresem projektu [23] można zapoznać się z udostępnianymi efektami animacji oraz przykładowymi stronami internetowymi, które powstały z jej wykorzystaniem.

Użycie biblioteki GSAP do wykonania prostej animacji obrotu obiektu wygląda analogicznie do opisanego wcześniej przykładu użycia biblioteki *jQuery* jednak zapis jest jeszcze prostszy. Jak pokazano na listingu 2.15 do wewnętrznej metody obiektu *Gsap* przekazywany jest selektor elementu podlegającego animacji oraz zestaw właściwości, które służą jako opcje animacji. Jak zostało opisane w oficjalnej dokumentacji, właściwość *rotation* odpowiada właściwości elementu HTML *„transform: rotate”* [24], używanej także we wcześniejszych przykładach.

Listing 2.15. Animacja obrotu elementu o 90 stopni wykonana przy użyciu biblioteki *GSAP*. Źródło: opracowanie własne za [24]

```
Gsap.to('#green-square', {rotation: 90, duration: 1});
```

Mechanizm działania biblioteki GSAP jest analogiczny do działania *jQuery*. Metody wbudowane w obiekt *Gsap* wykonują w pętli podmianę wartości dla wskazanych właściwości obiektu. Pętla jest odpowiednio spowolniona tak, aby animacja trwała dokładnie tyle ile wskazano w atrybucie *duration.* W efekcie następuje oczekiwany obrót obiektu trwający dokładnie 1 sekundę.

Oczywiście, zamiast zaprezentowanego na przykładzie *„rotation”* do metody może być przekazany każdy inny istniejący atrybut obiektu HTML. Twórcy biblioteki twierdzą, że ich produkt jest w stanie obsłużyć zmiany każdego ze znanych atrybutów takich jak np. *opacity*, *font-size*, w tym także takich, które sprawiają wrażenie niemożliwych do animacji np. *color* czy *transform*. Jest to możliwe dzięki wbudowanemu mechanizmowi mapowania nazw właściwości sposobu zapisu ich wartości i tak np. *„rotation: 90”* tłumaczone jest na *„transform: rotate(90deg)”*, a *„scale: 2”* na *„transform: scale(2)”* [25]. W przypadku wskazania atrybutu, który nie istnieje, zostanie on na obiekcie utworzony, a jego wartość będzie zmieniać się w czasie. Do metody mogą być też przekazane atrybuty specjalne których jest ponad 30 [26], najważniejsze z nich to:

- duration – oczekiwany czas trwania animacji podawany w sekundach;
- delay – opóźnienie animacji podawane w sekundach;
- paused – flaga służąca do oznaczenia animacji jako zatrzymanej;
- repeat – ilość powtórzeń podawana jako liczba, przy czym wartość -1 oznacza nieskończoność;
- repeatDelay – opóźnienie kolejnego powtórzenia podawane w sekundach;
- onStart – służy do przekazania funkcji, która będzie wykonana w momencie rozpoczęcia animacji;
- onComplete – służy do przekazania funkcji, która będzie wykonana w momencie ukończenia animacji.

Istnieje również możliwość przekazania wielu atrybutów z wartościami, właściwości te będą animowane jednocześnie. Przykład pokazano na listingu 2.16.

Listing 2.16. Animacja jednoczesnego obrotu i przesunięcia elementu wykonana przy użyciu biblioteki *GSAP*. Źródło: opracowanie własne za [27]

```
let anim = Gsap.to('#green-square', {rotation: 360,  x: 200, duration: 2});
```

Istnieje możliwość kontrolowania animacji w czasie jej trwania. Interfejs jest bardzo rozbudowany i zawiera ponad 30 metod. Wystarczy do zmiennej przypisać referencję do utworzonej animacji, jak pokazano na listingu 2.16, a następnie wywołać jedną z dostępnych metod [26]. Najważniejsze z nich to:

- *play* – uruchamia animację;
- *pause* – zatrzymuje animację;
- *restart* – zaczyna animację od początku;
- *kill* – zatrzymuje i odłącza animację od animowanego obiektu, animacja jest usuwana przez garbage collector;
- *resume* – kontynuuje odtwarzanie animacji od momentu, w którym była zatrzymana.
- *seek* – przewija animację do wskazanego momentu;
- *timeScale* – zmienia czas trwania animacji;
- *then* – wykonuje przekazaną w argumencie funkcję zaraz po zakończeniu animacji.

Podsumowując, zastosowanie bibliotek udostępniających gotowe metody do animacji obiektów DOM za pomocą kodu *JavaScript* jest dla twórców stron dużym ułatwieniem. Znacznie przyspiesza pracę, ogranicza ilość niezbędnego kodu i możliwość popełnienia błędów. Jednakże, jeśli zajdzie potrzeba zmiany sposobu animacji, nadal będzie konieczna modyfikacja kodu aplikacji.

2.1.5 Animacje generowane przy użyciu elementu HTML canvas

Element HTML *canvas* to prostokątny obszar umożliwiający generowanie grafik przy użyciu języka *JavaScript*. Istnieją dwa niezależne interfejsy umożliwiające rysowanie na tym elemencie. Metody interfejsu *Canvas API* skoncentrowane są na rysowaniu grafiki 2D i umożliwiają m.in. [28]:

- rysowanie dowolnych kształtów wektorowych, które ulegają rasteryzacji na elemencie *canvas*;
- umieszczanie innych obrazów wektorowych i rastrowych z plików zewnętrznych;
- renderowanie tekstów na element canvas;
- manipulacje i transformacje całego obrazu obejmujące m.in. przesuwanie, obroty, skalowanie;
- manipulacje poszczególnymi pikselami obrazu wyświetlanego na elemencie *canvas*;
- zapisywanie wygenerowanej na obszarze elementu *canvas* grafiki do pliku;
- czyszczenie elementu lub jego fragmentu.

Istnieje również interfejs *WebGL API*, który skupia się na rysowaniu grafiki z wykorzystaniem akceleracji sprzętowej. Jest on znacznie bardziej rozbudowany, pozwala m.in. na ustawienie obiektów 3D, zdefiniowaniu ich tekstur, oświetlenia i manipulacji kamerą. W związku z jego możliwościami wykorzystywany jest w głównej mierze do tworzenia gier lub interaktywnych aplikacji działających w środowisku 3D [29].

Canvas jest umieszczany na stronie internetowej jak każdy inny element HTML, można dla niego także określić wartości atrybutów takich jak wysokość, szerokość czy tło [28]. Animacje z wykorzystaniem tego elementu HTML generuje się poprzez mechanizm rysowania kolejnych klatek animacji w pętli. Na listingu 2.17 pokazano przykład obrotu zielonego kwadratu przy użyciu *Canvas API*, znany z opisu poprzednich metod animacji. Warto zwrócić uwagę, że podobnie jak w przypadku animacji elementu DOM przy użyciu *JavaScript* opisanego w punkcie 2.1.4 na listingu 2.10, to programista musi zadbać o obliczenie opóźnienia wykonywania pętli, po to, aby animacja trwała oczekiwany czas (w tym przypadku jest to jedna sekunda), a także o przerwanie pętli w odpowiednim czasie (w tym przypadku należy przerwać pętle po wykonaniu obrotu o dokładnie 90 stopni) [32]. Kod niezbędny do wykonania animacji przy pomocy tej technologii wydaje się najbardziej skomplikowany ze wszystkich dotychczas przedstawionych.

Listing 2.17. Animacja obrotu zielonego kwadratu przy użyciu elementu HTML *canvas*.
Źródło: opracowanie własne

```
const canvas = document.getElementById('canvas');
const context = canvas.getContext('2d');
const squareSize = 20;
let rotation = 0;
let intervalId = null;

function drawOnCanvas(angle) {
  context.clearRect(0, 0, canvas.width, canvas.height);
  context.save();
  context.translate(canvas.width / 2, canvas.height / 2);
  context.rotate(angle * Math.PI / 180);
  context.fillStyle = 'green';
  context.fillRect(-squareSize / 2, -squareSize / 2, squareSize, squareSize);
  context.restore();
}

function nextFrame() {
  if (rotation === 90) {
    clearInterval(intervalId);
  } else {
    drawOnCanvas(rotation)
    rotation++;
  }
}

intervalId = setInterval(nextFrame, (1000/90));
```

Zaletami tej techniki są:

- możliwość rysowania prostych kształtów wektorowych, jak i skomplikowanych grafik, w tym umieszczania obrazów bitmapowych przy użyciu *Canvas API*;
- możliwość renderowania obiektów 3D (wraz z teksturami, oświetleniem i różnymi ustawieniami perspektywy) przy wykorzystaniu *WebGL API*;
- animacje generowane z kodu „ważą" dużo mniej niż pliki wideo;
- rysowanie grafiki na elemencie *canvas* polega na przekształcaniu jej w postać bitmapy, co daje możliwość manipulowania nią na poziomie pikseli i w efekcie całkowitą kontrolę nad generowanym obrazem. Pozwala to na tworzenie różnorodnych efektów graficznych i animacji;
- możliwość zróżnicowana treści generowanej z poziomu kodu dla poszczególnych odbiorów lub w zależności od różnych sytuacji;
- możliwość szybkiej podmiany treści, także przez osoby nie posiadające kompetencji programistycznych poprzez dedykowany system CMS (ale dodawanie nowych efektów animacji do działającej aplikacji internetowej nie jest już takie proste);
- możliwość animacji dużej ilości niezależnych elementów – rysowanie za pomocą *Canvas API* nie obciąża wątku w takim stopniu jak animacje niezależnych elementów DOM, ponadto dzięki wsparciu akceleracji sprzętowej, nawet animacje 3D mogą być płynne i wydajne;
- ruchome elementy są zawsze przycinane do prostokątnego obszaru płótna;
- interfejsy *Canvas API* oraz *WebGL API* są standardem wpisanym w HTML5 i są wspierane przez wszystkie współczesne przeglądarki.

Dostrzeżono także wady tej technologii:

- elementy narysowane na *canvas* podlegają rasteryzacji, nie stanowią oddzielnych obiektów modelu DOM i w związku z tym nie obsługują zdarzeń (ang. *event*), takich jak np. kliknięcia;
- obiekty narysowane na elemencie *canvas*, podobnie jak zawartość obrazów czy animacji rastrowych, nie stanowią oddzielnych bytów i nie są dostępne dla czytników ekranów (ang. *screen readers*), nie są także indeksowane;
- implementacja kodu generującego animacje na elemencie HTML *canvas* jest trudna i czasochłonna w porównaniu do innych technologii;
- obraz wygenerowany na elemencie *canvas* jest statyczny, animacja wymaga rysowanie każdej klatki od początku;
- programowanie animacji może wymagać stałej współpracy projektanta strony i programisty;
- mniejsza niż w przypadku wcześniej wymienionych technologii, ilość dostępnych informacji, istniejących przykładów oraz bibliotek gotowych efektów.

Podsumowując, animacje z wykorzystaniem HTML *canvas* oferują ogromne możliwości dla twórców stron internetowych, jednak ich efektywne wykorzystanie wymaga pewnej wiedzy i umiejętności programistycznych.

Rozwiązaniem tego problemu mogą być gotowe rozwiązania ułatwiające implementację animacji przy użyciu elementu HTML *canvas*. Dobrym przykładem może być biblioteka EaselJS wraz z modułem TweenJS udostępniania przez *gskinner.com INC* na licencji MIT [33]. Oferuje ona szereg gotowych mechanizmów definiowania obiektów i ich hierarchii, a następnie ich wyświetlania i animacji na elemencie HTML *canvas*. Co istotne, biblioteka umożliwia także wykrywanie zdarzeń kliknięcia na utworzonych przy jej pomocy elementach [34].

Przykład wykorzystania tej technologii do animacji obrotu kwadratu o boku 20px o 90 stopni zaprezentowano na listingu 2.18. Definicja animacji obrotu w tym przypadku to właściwie jedna linijka: *„Tween.get(greenSquare).to({rotation:90}, 1000);”*. Określa ona, że oczekiwana animacja dotyczy obiektu zielonego kwadratu (tutaj obiekt *greenSquare*), wykonany ma zostać obrót o 90 stopni (ang. *rotation*), a czas animacji to 1000ms, czyli jedna sekunda.

W przypadku wykorzystania tej biblioteki programista nie musi własnoręcznie wyliczać czasu trwania animacji, definiować pętli rysowania poszczególnych klatek ani przerywać jej w oczekiwanym momencie.

Listing 2.18. Animacja obrotu zielonego kwadratu przy użyciu biblioteki EaselJS z modułem TweenJS. Źródło: opracowanie własne za [35]

```
const canvas = document.getElementById('canvas');
const stage = new Stage(canvas);
const squareSize = 20;
const greenSquare = new Shape();
greenSquare.graphics.beginFill("green").drawRect(0, 0, squareSize, squareSize);
greenSquare.regX = squareSize/2;
greenSquare.regY = squareSize/2;
stage.addChild(greenSquare);

Tween.get(greenSquare).to({rotation:90}, 1000);

Ticker.setFPS(60);
Ticker.addEventListener("tick", stage);
```

2.2 Wnioski z analizy istniejących rozwiązań

Niniejsza praca skupia się na animacjach mających na celu przyciągnięcie uwagi użytkownika i przekazanie mu określonej treści. Najczęściej są to animowane banery reklamowe, które pojawiają się w zamkniętym, prostokątnym obszarze ekranu, prezentując zarówno obrazy, jak i teksty. W takich sytuacjach konieczne jest zapewnienie możliwości łatwej wymiany treści, w sposób który nie wymaga specjalistycznej wiedzy technicznej ani ingerencji w kod portalu. Innym kluczowym aspektem jest umożliwienie jak największej kontroli nad uzyskiwanym efektem animacji oraz szerokie możliwości kreacji artystycznej obrazu.

Wszystkie przeanalizowane technologie posiadają wady i zalety. Umieszczanie na stronie internetowej gotowych plików animacji oferuje nieograniczone możliwości kreacji artystycznej i pełną kontrolę grafika nad efektem końcowym. Nie wymaga implementacji skomplikowanego kodu ze strony programisty. Animacje takie nie są jednak elastyczne i każda potrzeba jakiejkolwiek zmiany lub personalizacji wymaga wygenerowania pliku od nowa i zapisania go na serwerze. Ponadto, animacja odgrywana z pliku nie oferuje możliwości interakcji. Dla odmiany animacje elementów DOM przy użyciu arkusza stylu CSS lub kodu napisanego w języku *JavaScript* są wysoce konfigurowalne, umożliwiają personalizacje i dostosowanie treści do poszczególnego użytkownika lub sytuacji. Ponadto mogą reagować na interakcje użytkownika, takie jak kliknięcia, najechanie kursorem czy przewijanie strony. Ich wadą jest jednak organiczny wachlarz możliwości oraz fakt, że pisanie kodu animacji jest czasochłonne i wymaga sporej wiedzy od programisty tworzącego stronę internetową.

Rozwiązania polegające na generowaniu animacji z użyciem elementu HTML *canvas* możemy sytuować pomiędzy rozwiązaniem odgrywającym animacje z pliku, a rozwiązaniem animującym elementy DOM za pomocą kodu (CSS/JS). Obiekty będące elementami animacji generuje się z kodu, w związku z tym istnieje możliwość ich konfiguracji czy personalizacji. Są wydajne i „ważą" mniej niż pliki rastrowe. W momencie umieszczenia ich na elemencie *canvas* podlegają rasteryzacji, nie stanowią oddzielnych obiektów modelu DOM i w związku z tym nie obsługują zdarzeń. Drugą największą wadą rozwiązania wykorzystującego Canvas API jest trudna implementacja animacji oraz stosunkowo niewielka ilość przykładów wykorzystania tego rozwiązania.

Zastosowanie gotowych bibliotek takich jak *jQuery*, GSAP lub *EaselJS*/*TweenJS* powoduje ułatwienie i przyspieszenie pracy programisty przy implementacji animacji generowanej z poziomu języka *JavaScript*. Nierozwiązanym problemem pozostaje konieczność ingerencji w kod aplikacji, gdy zachodzi potrzeba zmiany animowanych treści. Wobec czego użytkownicy bez umiejętności programistycznych mają ograniczoną możliwość łatwej podmiany treści. Na rynku brakuje rozwiązań oferujących intuicyjne tworzenie animacji wykorzystujących *Canvas API*, które można następnie w prosty sposób wykorzystać na stronie internetowej.

Przeprowadzona analiza skłania do dalszych poszukiwań alternatywnych rozwiązań i wskazuje na potrzebę stworzenia bardziej intuicyjnych narzędzi lub interfejsów, które umożliwią edycję animacji bez konieczności bezpośredniej pracy z kodem. Dzięki temu, nawet osoby bez specjalistycznej wiedzy mogłyby efektywnie zarządzać i modyfikować animowane treści w aplikacjach internetowych.

3. Propozycja nowego rozwiązania

Rozdział ten zawiera omówienie koncepcji rozwiązania problemów zidentyfikowanych podczas analizy przeprowadzonej w rozdziale drugim oraz określa wymagania stawiane przed projektowanym rozwiązaniem.

3.1 Zarys koncepcji nowego rozwiązania

Przeprowadzona analiza dowodzi, że dla animacji, których zadaniem jest skupienie uwagi użytkownika na wskazanym obszarze i przekazania mu pewnych treści, korzystne jest rozwiązanie wykorzystujące interfejs *Canvas API*. Jeżeli chodzi o dostępne efekty, jest ono sytuowane pomiędzy rozwiązaniami wyświetlającymi przygotowane pliki graficzne, a rozwiązaniami animującymi elementy DOM za pomocą kodu. Przewyższa konkurencję, jeżeli chodzi o elastyczność i możliwość konfiguracji. Nie jest jednak pozbawione wad. Animacje tego typu są trudne w implementacji. Nawet przy użyciu bibliotek oferujących gotowe efekty, nierozwiązanym problemem pozostaje konieczność ingerencji w kod aplikacji, gdy zachodzi potrzeba zmiany animowanych treści.

Na rynku brakuje rozwiązań oferujących intuicyjne tworzenie animacji wykorzystujących interfejs *Canvas API*, które można następnie w prosty sposób wykorzystać na stronie internetowej. W związku z tym użytkownicy, którzy nie posiadają umiejętności programistycznych mają ograniczoną możliwość aktualizacji animowanych treści wytworzonych przy użyciu tej technologii.

W celu poprawy obecnego stanu sztuki proponuje się nowe, kompleksowe rozwiązanie, które ułatwia proces tworzenia animacji oraz osadzenia jej na stronie internetowej. Istotnym aspektem rozwiązania jest umożliwienie aktualizacji treści animacji bez potrzeby ingerencji w kod portalu internetowego.

Aby osiągnąć założony cel, w ramach niniejszej pracy magisterskiej opracowuje się prototyp „Elastycznego systemu tworzenia animacji dla portali internetowych", składający się z trzech elementów:

- **edytora animacji**, który umożliwia generowanie animacji z uwzględnieniem zadanych parametrów oraz ich podgląd w czasie rzeczywistym. Edytor wyświetla instrukcje osadzenia animacji na stronie internetowej w formie gotowej do skopiowania oraz umożliwia eksport „scenariusza animacji" w postaci kodu gotowego do użycia, który jest następnie wykorzystywany przez element odgrywający animacje. Obsługa edytora nie wymaga specjalistycznej wiedzy;
- **komponentu odgrywającego animację** – elementu umieszczonego na dowolnej stronie internetowej w postaci *Web Componentu*, odgrywającego animacje według przekazanego „scenariusza animacji", utworzonego wcześniej za pomocą edytora animacji. Zmiana odgrywanej treści nie wymaga zmiany kodu aplikacji internetowej, a jedynie wczytania nowego, zaktualizowanego „scenariusza animacji". Aby zapewnić generyczność rozwiązania i możliwość poszerzenia listy dostępnych efektów, animacje odgrywane są przy wykorzystaniu dedykowanych wtyczek animacji;
- **dedykowanych wtyczek animacji** wykorzystywanych przez wyżej wymieniony komponent do generowania animacji. System wtyczek zapewnia generyczność rozwiązania i możliwość rozszerzenia go o nowe efekty.

Element odtwarzający animacje oraz dedykowane wtyczki animacji, które mają z nim współpracować, zostały zaprojektowane jako skrypty napisane w języku *JavaScript*. Dzięki temu są bezproblemowo wczytywane i używane na różnych stronach internetowych. Edytor animacji, który ma być łatwo dostępny dla użytkowników w postaci strony internetowej, również został stworzony w języku JavaScript. Zapewnia to spójność całego systemu i uławia integracje z technologiami aplikacji internetowych.

3.2 Wymagania stawiane przed projektowanym rozwiązaniem

Ten podrozdział przedstawia wymagania, co do projektowanego rozwiązania. Są one podzielone na trzy sekcje, każda z nich dedykowana jest jednemu z trzech projektowanych elementów prototypu.

3.2.1 Edytor animacji

Edytor animacji pozwala osobom bez umiejętności programistycznych tworzyć animacje z wykorzystaniem elementu HTML *canvas* i interfejsu *Canvas API*. Edytor spełnia wymagania:

- jest łatwy i intuicyjny w obsłudze i nie wymaga od użytkownika specjalistycznej wiedzy;
- umożliwia wybieranie efektów z listy dostępnych animacji oraz modyfikowanie ich parametrów za pomocą interfejsu graficznego (GUI);
- obsługuje system dedykowanych wtyczek animacji, zapewniający możliwość rozbudowy o kolejne efekty animacji;
- prezentuje animacje, reagując na zmiany parametrów w czasie rzeczywistym;
- udostępnia instrukcję użycia całego systemu w sposób zrozumiały i czytelny;
- umożliwia eksport „scenariusza animacji" w postaci fragmentu kodu gotowego do użycia na dowolnej stronie internetowej;
- działa w środowisku przeglądarki internetowej (aplikacja typu SPA).

3.2.2 Komponent odgrywający animacje

Komponent odgrywający animacje umieszczany jest w kodzie HTML dowolnej strony internetowej. Komunikuje się zarówno ze stroną, na której jest osadzony, jak i z dedykowanymi wtyczkami animacji. Dla elementu formułuje się następujące wymagania:

- prosty sposób osadzania elementu na dowolnej stronie internetowej – umożliwiony dzięki udostępnieniu go w formie *Web Componentu*;
- łatwość importu elementu na docelowe strony, na których ma być użyty – zapewniona poprzez jego publikację w ogólnodostępnym repozytorium pakietów Node (ang. *Node Package Manager Registry*) oraz sieci dostarczania zawartości (ang. *content delivery network*, CDN);
- prosty sposób komunikacji pomiędzy elementem odgrywającym animację, a stroną internetową na której jest on umieszczony – komponent odbiera treści i instrukcje animacji przekazywane za pomocą atrybutów HTML;
- zawiera element HMTL *canvas*, na którym generowane są animacje z wykorzystaniem interfejsu *Canvas API* i udostępnia referencję do niego dedykowanym wtyczkom animacji*;*
- komunikuje się z dedykowanymi wtyczkami animacji za pomocą zdarzeń DOM (ang. *event*);
- w celu optymalizacji animacji korzysta z wbudowanej metody *window.requestAnimationFrame.*

3.2.3 Dedykowane wtyczki animacji

System dedykowanych wtyczek animacji, umożliwia rozszerzenie systemu o nowe efekty animacji. Oczekuje się, że wtyczki będą spełniać następujące wymagania:

- zapewnienie generyczności całego systemu poprzez zastosowanie uniwersalnych metod komunikacji z pozostałymi jego elementami za pomocą zdarzeń DOM (ang. *event*) oraz implementacja mechanizmu parametrów animacji, który jest otwarty na rozszerzenia;
- prostota importowania i używania wtyczek na docelowych stronach korzystających z całego systemu, dzięki udostępnieniu wtyczek w postaci skryptów JavaScript;
- optymalizacja animacji – rysowanie na elemencie HTML *canvas* oraz wykorzystanie wbudowanej metody *window.requestAnimationFrame.*

4. Narzędzia i technologie zastosowane w pracy

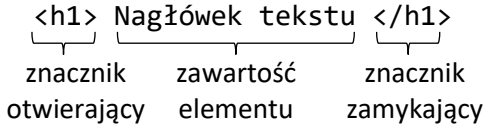
Rozdział ten zawiera krótkie omówienie technologii i narzędzi zastosowanych przy realizacji pracy. Koncepcja rozwiązania wykorzystuje nowoczesne, popularne i ogólnodostępne technologie.

4.1 HTML

HTML (ang. *HyperText Markup Language*) to sposób zapisu służący do definiowania struktury i zawartości stron internetowych. Plik formatu HTML jest odczytywany i parsowany przez każdą przeglądarkę internetową lub inny przystosowany do tego program (tzw. *user agent*). Zapisane w nim elementy przekształcane są w drzewo obiektów DOM (*Document Object Model*) [36].

Plik taki składa się z ciągu elementów, a poszczególny element składa się zwykle ze znacznika (ang. *tag*) otwierającego, zawartości i znacznika zamykającego. Od tej reguły istnieją wyjątki, ponieważ spotkać można też elementy składające się wyłącznie z pojedynczego znacznika. Budowę typowego elementu pokazano na rysunku 4.1. Elementy często bywają zagnieżdżone, to znaczy, że w zawartości jednego elementu znajduje się jeden lub więcej innych elementów [37].

Z punktu widzenia niniejszej pracy istotny jest fakt, że katalog znaczników domyślnie interpretowanych przez przeglądarki jest ustandaryzowany i zamknięty. Istnieje jednak możliwość zarejestrowania nowej, własnej definicji znacznika, a następnie użycia go w pliku HTML np. przy pomocy technologii *Web Components,* opisanej w podrozdziale 4.4.



Rysunek 4.1. Budowa typowego elementu HTML.
Źródło: opracowanie własne za [37]

Zgodnie ze standardem, plik HTML powinien mieć określoną strukturę. Pierwsza linia powinna zawierać deklarację typu. Następnie występuje znacznik otwierający element główny jakim jest *html*. W nim zagnieżdżone są elementy *head* i *body*. Plik powinien kończyć się znacznikiem zamykającym element *html* [38].

Element *head* nie zawiera wprost treści strony internetowej. Zawiera on metadane – elementy stanowiące dodatkowe informacje, jak na przykład tytuł strony (znacznik *title*) oraz bardzo często linki do innych zasobów, w tym elementy wczytujące skrypty napisane w języku *JavaScript* [39]. Jest to istotna funkcjonalność z punktu widzenia niniejszej pracy, ponieważ właśnie za pomocą elementów *script* wczytywane są i wykonywane pliki *JavaScript* [40], zawierające m.in. wtyczki animacji oraz definicję niestandardowego elementu jakim jest komponent *AnimatedBanner*.

Zawartość elementów zagnieżdżonych w elemencie *body* jest zwykle wyświetlana użytkownikowi, natomiast same znaczniki, wraz z atrybutami, są jedynie informacją na temat oczekiwanego sposobu wyświetlania zawartości i jego roli w strukturze dokumentu, czasem przekazują też informacje dotyczące semantyki tj. roli jaką odgrywa zawartość w kontekście strony internetowej [41]. Przykładowy plik formatu HMTL, nawiązujący do przedmiotu niniejszej pracy pokazanego na listingu 4.1.

Listing 4.1. Przykładowy plik formatu HTML wyświetlający komponent *AnimatedBanner*.
Źródło: opracowanie własne za [38]

```
<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
    <script type='module'>
      import { defineCustomElements }
      from 'https://unpkg.com/animated-banner/loader/index.es2017.js';
      defineCustomElements();
    </script>
    <script
      type='module'
      src='https://unpkg.com/animated-banner/animations/slide-text.js'>
    </script>
  </head>
  <body style="border: 1px solid black">
    <h1>This is a Heading</h1>
    <p style='red'>This is a <strong>paragraph</strong>text.</p>
    <div>
      <animated-banner-component
        id='animated-banner-1'
        width='500'
        height='250'
        match-parent='false'
        background='black'
     ></animated-banner-component>
    </div>
  </body
</html>
```

4.2 JavaScript

JavaScript (w skrócie JS) to język programowania który powstał w latach 90 XX wieku i wykorzystywany jest głównie w technologiach internetowych, w szczególności do manipulowania zawartością strony internetowej oraz obsługiwania interakcji użytkownika ze stroną. Jest to język:

- interpretowany – nie wymaga uprzedniej kompilacji, jest ona dokonywana w czasie trwania programu. Kompilacji dokonuje silnik wbudowany na przykład w przeglądarkę internetową;
- funkcyjny – czyli taki, w którym to funkcje, a nie obiekty są typami pierwszoklasowymi;
- wspierający programowanie obiektowe;
- typowany dynamicznie – czyli taki, w którym typy zmiennych przypisywane są w czasie trwania programu i mogą ulegać zmianie [42].

Jest on rozumiany natywnie przez wszystkie przeglądarki internetowe, chociaż występują różnice w kompatybilności niektórych poleceń, pojawiające się w związku z różnym tempem implementacji kolejnych standardów języka przez poszczególnych producentów przeglądarek internetowych lub ich własnymi innowacjami.

Wszystkie elementy niniejszej pracy napisane w języku *TypeScript* zostały skompilowane do języka *JavaScript* właśnie po to, aby były zrozumiałe przez przeglądarki internetowe.

4.3 TypeScript

Język programowania *TypeScript* (w skrócie TS) jest to nadzbiór języka *JavaScript*. Oznacza to, że rozszerza on możliwości języka JS. Forma zapisu zapewnia jednak, że każdy kod napisany w języku JS będzie poprawny z punktu widzenia języka TS. Natomiast, aby kod, który używa dodatkowych funkcjonalności języka TS, był zrozumiały przez interpreter języka JS, należy go wcześniej skompilować [43]. Został on udostępniony przez firmę *Microsoft* jako darmowa technologia otwartoźródłowa (ang. *open source*).

Wszystkie elementy składające się na prototyp rozwiązania zaproponowanego w niniejszej pracy, czyli komponent *AnimatedBanner*, dedykowane wtyczki animacji oraz edytor animacji, napisano przy użyciu języka *TypeScript*. Funkcjonalności języka *TypeScript*, których nie posiada język *JavaScript*, a które zostały wykorzystane przy tworzeniu niniejszej pracy to:

- typowanie statyczne, czyli definiowanie typów obiektów, zmiennych oraz argumentów i rezultatów funkcji na etapie pisania kodu;
- interfejsy, których użycie pozwala na określnie wymagań w stosunku do oczekiwanych obiektów;
- klasy i dziedziczenie klas, znane z języka takiego jak Java czy C#;
- dekoratory, czyli szczególne deklaracje dołączane do metody, pola lub definicji klasy, rozszerzające zachowanie tej metody, pola lub klasy. W rzeczywistości, dekorator to pewna funkcja, zmieniająca ten obiekt, to pole lub tę metodę w czasie trwania programu [44].

Wykorzystanie tego języka i jego funkcjonalności pomaga uniknąć niektórych błędów na etapie pisania kodu, szczególnie błędów związanych z typem danych. Ponadto systematyzuje i organizuje, a nawet w pewnym sensie opisuje kod.

Na listingu 4.2 pokazano przykładowy fragment kodu napisanego w języku *TypeScript* obrazujący m.in. wykorzystanie typowania statycznego, interfejsu, a także dziedziczenia klas. Obiekty utworzone na podstawie klasy *SlideText* i *SlideImage* implementują interfejs *AnimatorInterface* i w związku z tym, komponent *AnimatedBanner* ma pewność, że na tych obiektach, znajdujących się w tablicy *runningAnimation*, można wykonać metody *start* i *stop*.

Listing 4.2. Przykładowy plik formatu HTML.
Źródło: opracowanie własne

```
export interface AnimatorInterface {
  animate: Function
  stop: Function
}

export class SlideText implements AnimatorInterface { … }

export class SlideImage extends SlideText implements AnimatorInterface { … }

export class AnimatedBannerComponent {
  …
  runningAnimations: AnimatorInterface[] = []
  runningAnimations.push(new SlideText())
  runningAnimations.push(new SlideImage())
  runningAnimations.forEach((anim: AnimatorInterface) => anim.stop()
  …
}
```

4.4 Web Components

Web Components to technologia umożliwiająca tworzenie komponentów, fragmentów stron internetowych, które mogą być wielokrotnie używane w obrębie jednej jak i wielu różnych stron [45]. Komponentami są najczęściej powtarzalne elementy takie jak kontrolki formularzy lub wytworzony w ramach niniejszej pracy komponent *AnimatedBanner.*

Jak opisano w podrozdziale 4.1, katalog znaczników domyślnie interpretowanych przez przeglądarki internetowe jest ustandaryzowany i zamknięty, ale przy pomocy technologii *Web Components* oraz powiązanym z nim rejestrowi niestandardowych elementów możliwe jest zarejestrowanie nowej definicji znacznika tzw. *custom element*, a następnie użyć niestandardowego elementu w prosty sposób w pliku HTML. Po rejestracji element taki używa się tak jak każdy inny standardowy element HTML. W chwili pisania pracy wszystkie popularne przeglądarki wspierają to rozwiązanie [46].

Element niestandardowy rozszerza element HTML (klasę *HTMLElement*) lub inną klasę, która po niej dziedziczy, na przykład *HTMLParagraphElement*. Klasę taką następnie rejestrujemy w rejestrze niestandardowych elementów wraz z nazwą znacznika. Od tego momentu znacznik ten będzie elementem HTML zrozumiałym dla przeglądarki internetowej. Wspomniany rejestr (interfejs *CustomElementRegistry*) dostępny jest jako właściwość okna przeglądarki – *window.customElements* [47]. Schemat rejestracji pokazano na listingu 4.3. Na każdym zarejestrowanym w ten sposób obiekcie, przeglądarka będzie we właściwym momencie próbować wykonywać tzw. metody cyklu życia:

- *connectedCallback* – wywoływana, kiedy element jest dołączony do drzewa DOM;
- *disconectedCallback* – wywoływana, kiedy element jest odłączony od drzewa DOM;
- *adopted* – wywoływana, kiedy element został przeniesiony do innego dokumentu;
- *attributeChangedCallback* – wywoływana, kiedy jeden z atrybutów HTML ulega zmianie [47].

Listing 4.3. Przykładowy schemat kodu *Web Componentu* oraz jego rejestracja.
Źródło: opracowanie własne za [47]

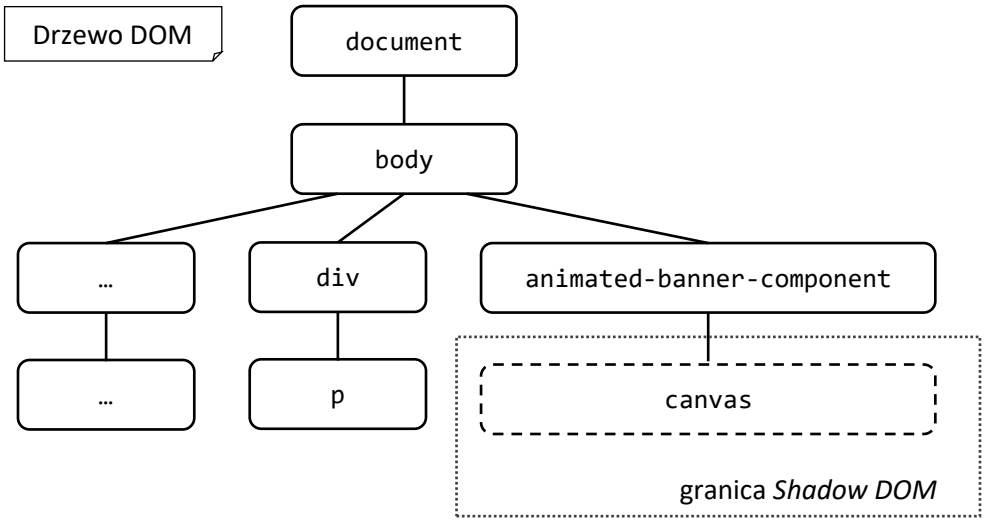
```
export class AnimatedBannerComponent extends HTMLElement {

  constructor() {
    super();
  }

  connectedCallback() {…}
  disconnectedCallback() {…}
  adoptedCallback() {…}
  attributeChangedCallback(name, oldValue, newValue) {…}
}

customElements.define("animated-banner-component", AnimatedBannerComponent);
```

Komponent *AnimatedBanner* rejestrowany jest również przy wykorzystaniu tego mechanizmu. Metoda *define* wywoływana jest w skrypcie rejestrującym komponent, wytworzonym przez narzędzia *frameworka Stencil.js.* Wykorzystane są również metody cyklu życia *Web Componentów,* jednak i one opakowane są przez narzędzia tego *frameworka.* Dla programisty udostępnione są metody cyklu życia opisane w podrozdziale 4.8, opisującym dokładnie *Stencjil.js*.

4.5 Shadow DOM

Dla każdego elementu HTML odczytanego z parsowanego pliku HTML tworzony jest odpowiedni obiekt. Obiekty te dodawane są do drzewiastej struktury przechowywanej w pamięci, odpowiadającej strukturze pliku HTML, czyli drzewa DOM [36].

Web Componenty jako wydzielone fragmenty strony internetowej, przeznaczone do ponownego wykorzystania, posiadają własny szablon HTML zawierający kolejne zagnieżdżone elementy HTML, własne style oraz kod realizujący funkcjonalności tego komponentu. Programiści zajmujący się tworzeniem komponentów nie zawsze muszą wiedzieć na jakiej stronie zostaną one w przyszłości użyte, a więc w jakim otoczeniu będą działać. I odwrotnie, programiści tworzący strony internetowe nie zawsze muszą zagłębiać się we wszystkie szczegóły działania komponentu. Aby nie dopuścić do zagrożeń jakim może być przenikanie się działania kodu lub stylowania, obiekty komponentów są enkapsulowane, czyli umieszczane w wydzielonym drzewie elementów tzw. *Shadow DOM*. [48].



Rysunek 4.2. Przykład drzewa DOM zawierającego poddrzewo *Shadow DOM*.
Źródło: opracowanie własne za [48]

Na rysunku 4.2 zobrazowano schemat przykładowego drzewa DOM, w którym umieszczono komponent *AnimatedBanner*. Jak pokazuje rysunek, sam niestandardowy element HTML, zarejestrowany pod nazwą znacznika „*animated-banner-component*” jest dostępny z poziomu zwykłego drzewa DOM. Jest to tzw. „*shadow-host*” [48]. Enkapsulowane w *Shadow DOM* są dopiero elementy HTML zawarte wewnątrz komponentu, w jego szablonie, tutaj jest to element *canvas*. Końcowy efekt kodu HTML - strony sparsowanej przez przeglądarkę zaprezentowano na listingu 4.4.

Listing 4.4. Komponent *AnimatedBanner* w kodzie HTML strony sparsowanej przez przeglądarkę.
Źródło: opracowanie własne

```
<animated-banner-component width='500' height='250' background='black'>
  #shadow-root
    <canvas width='500' height='250' background='black'></canvas>
</animated-banner-component>
```

4.6 Canvas API

Canvas API to interfejs programowania (zestaw metod) służący do rysowania grafik na elemencie HTML *canvas* przy pomocy kodu napisanego w języku *JavaScript*. Metody te zapewniają możliwość umieszczania na elemencie *canvas* dowolnych kształtów wektorowych, tekstu i obrazków, w tym plików rastrowych wczytanych z zewnętrznego źródła, oraz czyszczenie elementu [28].

Koncepcja rozwiązania przedstawionego w niniejszej pracy wykorzystuje właśnie tę technologię. Komponent *AnimatedBanner* zawiera w sobie element HTML *canvas* i przekazuje dedykowanym wtyczkom animacji referencję do tego elementu. Wtyczki rysują (jednocześnie lub nie) swoje części składowe animacji na elemencie. Używają do tego między innymi metod:

- *fillText* – umieszcza dowolny tekst we wskazanym miejscu elementu *canvas* [30];
- *drawImage* – umieszcza grafikę rastrową we wskazanym miejscu elementu *canvas* [31].

Po narysowaniu każdej klatki, element *canvas* jest czyszczony za pomocą metody *clearRect* i przygotowywany w ten sposób do rysowania kolejnej klatki. Przykładowe wykorzystanie metod rysowania na elemencie *canvas* i czyszczenia go przedstawiono na listingu 4.5.

Listing 4.5. Przykładowe wykorzystanie metod *Canvas API*.
Źródło: opracowanie własne

```
export class AnimatedBannerComponent {
  …
  this.canvas
    .getContext('2d')
    .clearRect(0, 0, this.canvas.clientWidth, this.canvas.clientHeight)
  …
}

export class SlideTextAnimation {
  …
  this.canvas
    .getContext('2d')
    .drawImage(this.image, this.imageX, this.imageY)
  …
}

export class SlideImageAnimation {
  …
  this.canvas
    .getContext('2d')
    .fillText(this.text, this.textX, this.textY)
  …
}
```

Metody interfejsu *Canvas API* skoncentrowane są na rysowaniu grafiki 2D. Istnieje również interfejs *WebGL API*, który skupia się na rysowaniu grafiki z wykorzystaniem akceleracji sprzętowej 2D oraz 3D [28]. Ponieważ rysowanie odbywa się również na elemencie *canvas* nie ma przeciwskazań, aby stworzyć dedykowaną wtyczkę animacji, wykorzystującą interfejs *WebGL*.

4.7 Request Animation Frame

Metoda *requestAnimationFrame* to metoda obiektu *window* dostępna we wszystkich współczesnych przeglądarkach. Wysyła prośbę wywołania dowolnej innej funkcji, przekazanej w argumencie, w czasie następnego odświeżenia ekranu [49]. Metoda ta jest często wykorzystywana do rysowania animacji w sposób optymalny, bo zsynchronizowany z odświeżaniem ekranu [50]. Kolejną właściwością tej metody jest oszczędność zużycia energii, ponieważ w większości przeglądarek, jej wywołania są wstrzymywane, gdy działają one w tle [49].

Należy pamiętać, że w wyniku działania metody, przekazana funkcja zostanie wykonana tylko raz, przy okazji najbliższego odświeżenia ekranu. Można to rozumieć jako narysowanie jednej klatki animacji. Aby wykonać dłuższą animację, należy zapętlić wykonywanie tej metody, a następnie przerwać pętle w odpowiednim momencie. Ten mechanizm został wykorzystany przy tworzeniu prototypu, będącego częścią niniejszej pracy i zobrazowany na listingu 4.6.

Listing 4.6. Kod odpowiedzialny za rysowanie animacji w pętli.
Źródło: opracowanie własne

```
private runAnimation(): void {
  this.updateObjectPosition()
  this.drawObjectOnCanvas()
  …

  if(this.endReached) {
   window.cancelAnimationFrame(requestRef)
  } else {
    reqestRef = window.requestAnimationFrame(this.runAnimation.bind(this))
  }
}

private drawObjectOnCanvas(): void {
  this.canvasContext.fillStyle = this.color
  this.canvasContext.fillText(this.text, this.textX, this.textY)
}
```

Ważnym elementem mechanizmu jest ostatnie polecenie wywoływane w metodzie *runAnimation*, czyli *window.requestAnimationFrame(this.runAnimation.bind(this))*. Konstrukcja ta oznacza żądanie ponownego wywołania metody *runAnimation* przy następnym odświeżeniu ekranu. W ten sposób powstaje zoptymalizowana pętla rysowania klatek animacji wykonywana z prędkością raz na każde odświeżenie ekranu.

Metoda *requestAnimationFrame* zwraca referencję do żądania. Na przedstawionym na listingu 4.6. przykładzie, referencja zostaje przypisana do zmiennej *requestRef*. Za jej pomocą można anulować żądanie, które nie zostało jeszcze wykonane [51]. W tym celu wywoływana jest metoda *window.cancelAnimationFrame(requestRef)*.

W pętli tego typu częstotliwość rysowania klatek animacji jest zgodna z częstotliwością odświeżania ekranu. W większości przypadków odświeżanie ekranu wynosi 60Hz, czyli klatki rysowane są z prędkością 60 razy na sekundę. Ponieważ istnieją ekrany, które oferują inną częstotliwość odświeżania, pętla na takich ekranach będzie wywoływała się również z inną częstotliwością. Należy to uwzględnić przy projektowaniu tego typu pętli animacji i uzależnić postęp animacji od upływającego czasu, zwracanego przykładowo przez metodę *performance.now* [49].

4.8 Stencil.js

Stencil.js to *framework* który powstał w celu przyspieszenia procesu tworzenia *Web Componentów*, a następnie ułatwienia użycia ich na dowolnych stronach internetowych. Wykorzystuje technologie *Web Components API* i *Shadow DOM*, które są standardowo udostępniane przez wszystkie współczesne przeglądarki, jednak usprawnia ich użycie. Dostarcza on także rozwiązania ułatwiające integrację powstających za jego pomocą komponentów z aplikacjami internetowymi stworzonymi za pomocą frameworków takich jak *Angular, React, Vue* czy *Ember* [52].

Stencil.js wspiera pisanie kodu w języku *TypeScript*. Kod jest następnie kompilowany do języka *JavaScript*. Wbudowane narzędzia pozwalają na tworzenie zminimalizowanych paczek zawierających kompletną definicję komponentu, jego logikę oraz szablon HTML wraz z niezbędnymi stylami. Kod źródłowy podlega optymalizacji obejmującej eliminację nieużywanego kodu (ang. *tree-shaking*) oraz minimalizację kodu. Dodatkowo generowane są skrypty pozwalające zarejestrować znaczniki *Web Componentu* jako element niestandardowy (tzw. *custom element*). Przez przeglądarkę internetową uruchamiana jest właśnie ta zminimalizowana i skompilowana do języka *JavaScript* postać komponentu, a dokładnie jeden ze skryptów rejestrujących.

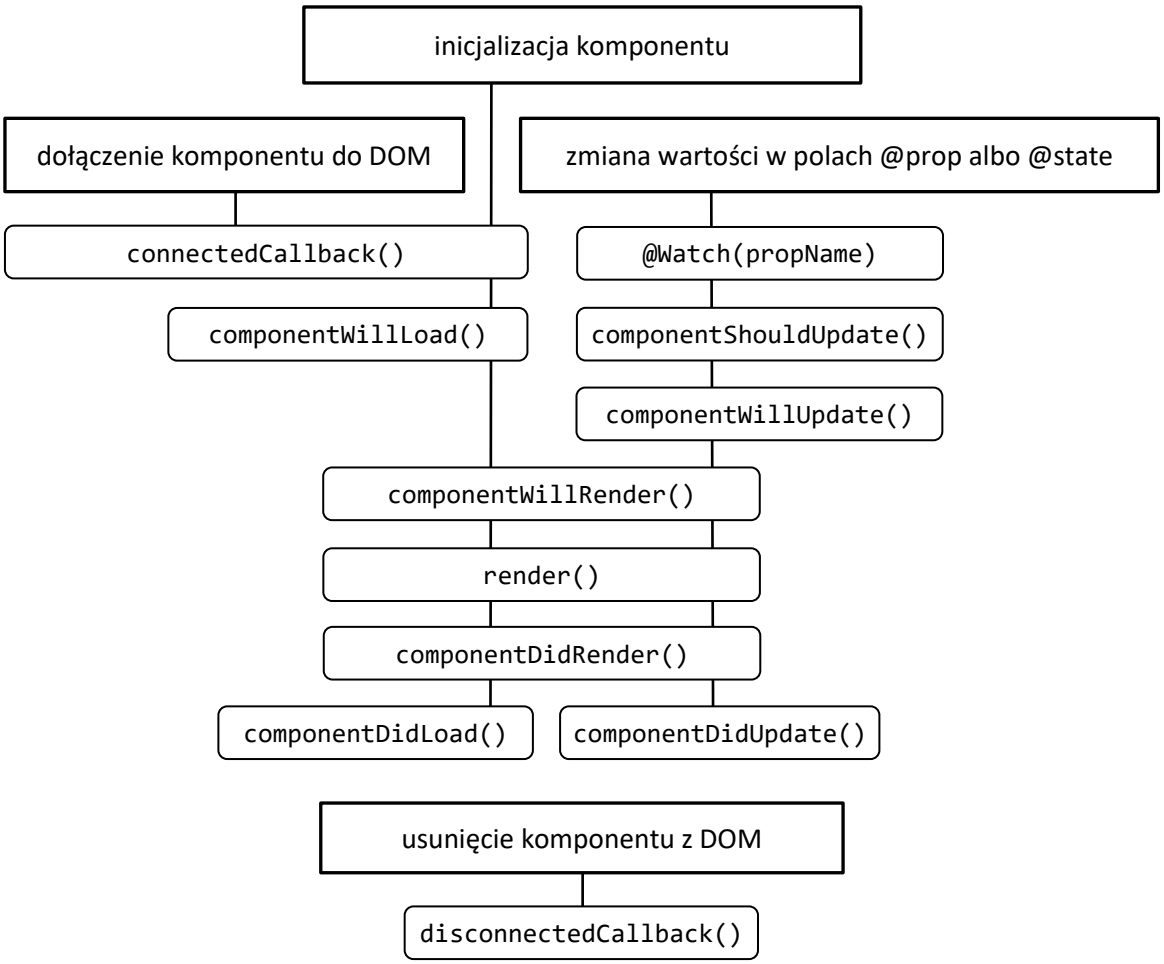
Stencil.js jest technologią darmową udostępnioną przez twórców, „*Ionic core team”,* w ramach licencji MIT [53] i został użyty do stworzenia części prototypu, jakim jest komponent *AnimatedBanner*. Wykorzystano szereg oferowanych przez niego mechanizmów takich jak:

- przekazywanie atrybutów HTML elementu będącego *shadow-hostem* do wnętrza komponentu;
- emisja zdarzeń (ang. *event*) wykrytych wewnątrz drzewa *Shadow DOM* komponentu na zewnątrz;
- nasłuchiwanie wewnątrz komponentu na zdarzenia z zewnątrz;
- wywoływanie metod cyklu życia.

Sposób ich wykorzystania został opisany szerzej w podrozdziale 5.2 skupiającym się na szczegółach implementacji komponentu *AnimatedBanner*.

Metody związane z cyklem życia komponentu wywoływane są automatycznie w określonych momentach trwania programu (ang. *runtime*). Wystarczy, że metody o właściwych nazwach są zaimplementowane w klasie komponentu. Część z nich wywoływana jest w następstwie cyklu życia elementu niestandardowego (ang. *custom element*) jakim jest każdy *Web Component*. Metody te opisano w podrozdziale 4.4. Dla przykładu, metoda *connectedCallback* zaimplementowana w klasie komponentu wywoływana jest bezpośrednio w wyniku wywołania metody *connectedCallback* na elemencie niestandardowym. Analogiczna sytuacja ma miejsce w przypadku metody *disconnectedCallback.* Pozostałe są specyficzne dla *frameworka Stencil.js* [54]. Kompletny cykl życia komponentów tworzonych za pomocą *Stencil.js* zaprezentowano na rysunku 4.3. Na diagramie zawarte są cztery prostokąty, symbolizują one wydarzenia, które inicjują wywoływanie szeregu metod cyklu życia:

- inicjalizacja komponentu (ang. *component initialized*) – prowadzi do wywołania metod *connectedCallback, componentWillLoad, componentWillRender, render, componentDidRender, componentDidLoad*;
- dołączenie komponentu do drzewa DOM (ang. *component reattached)* – wywołuje metodę *connectedCallback*;
- zmiana wartości w polach oznaczonych dekoratorami *@Prop* albo *@State* – prowadzi do wywołania metod *@Watch, componentShouldUpdate, componentWillUpdate, componentWillRender, render, componendDidRender, componentDidUpdate;*
- usunięcie komponentu z drzewa DOM (ang. *component removed)* – wywołuje metodę *disconnectedCallback* [54].



Rysunek 4.3. Metody cyklu życia komponentów wytworzonych przy użyciu frameworka Stencil.js. Źródło: Opracowanie własne za [54]

4.9 Node.js i NMP

Jest to darmowe, międzyplatformowe środowisko uruchomieniowe (ang. *runtime environment*) dla aplikacji napisanych w języku *JavaScript*. Oznacza to, że przy jego pomocy można uruchamiać aplikacje napisane w tym języku, poza środowiskiem przeglądarki. Zostało utworzone z myślą o tworzeniu serwerów WWW napisanych w języku *JavaScript* [55], jednak rozpropagowane zostało również jako środowisko uruchomieniowe menadżerów pakietów typu NPM oraz jako narzędzie wspierające tworzenie aplikacji internetowych. Wszystkie współczesne *frameworki*, w tym także użyte w niniejszej pracy *Angular* i *Stencil.js*, wymagają instalacji środowiska *Node.js*.

Node Package Manager (w skrócie NPM) to menadżer pakietów dla języka *JavaScript* działający w środowisku uruchomieniowym *Node.js* [56]. Zarządzanie pakietami odbywa się z poziomu konsoli, chociaż jest też dobrze zintegrowany ze środowiskiem developerskim *IntelliJ*. Przy wytwarzaniu prototypu będącego efektem niniejszej pracy, NPM jest wykorzystywany na trzy sposoby:

- do instalacji pakietów niezbędnych do uruchomienia *frameworków Angular* i *Stencil.js*;
- do uruchomienia edytora animacji oraz komponentu *AnimatedBanner* wytworzonych przy użyciu *frameworków Angular* i *Stencil.js*;
- do publikacji skompilowanego komponentu *AnimatedBanner* jako pakietu NPM gotowego do użycia w innych aplikacjach, m.in. edytorze animacji.

4.10 Angular

Angular jest to, jak określają go twórcy, platforma programistyczna, służąca do budowy skalowalnych aplikacji internetowych. Rozwiązanie obejmuje:

- *framework* oparty na komponentach i modułach, nadający się do budowy zarówno małej aplikacji na potrzeby prywatne jak i rozwiązań korporacyjnych;
- ogromną bazę bezpłatnych i otwartoźródłowych bibliotek i wtyczek;
- zestaw narzędzi wspierających pracę, testy i migrację do kolejnych wersji;
- rozbudowaną bazę wiedzy obejmującą dokumentację oraz materiały szkoleniowe w postaci kompletnych kursów i krótszych *tutoriali* [57];

Angular jest jednym z trzech najpopularniejszych *frameworków*. Jest darmowy i udostępniany przez firmę przez *Google LLC.* Jest to następca przestarzałego *frameworka AngularJS,* jednak jest to zupełnie nowe i niezależne rozwiązanie. Został wykorzystany do stworzenia części prototypu jakim jest edytor animacji.

Podobnie jak *Stencil.js,* także i *Angular* wymaga instalacji środowiska *Node.js* oraz menadżera pakietów NPM. Komponenty, z których złożona jest widoczna część aplikacji, budowane są w sposób hierarchiczny (komponenty bardziej szczegółowe znajdują się wewnątrz komponentów bardziej ogólnych) i składają się zwykle z trzech elementów: szablonu HTML, pliku ze stylami, oraz pliku zawierającego logikę działania napisanej w języku *TypeScript.* Poza komponentami, wykorzystuje się także koncepcję serwisów, czyli elementów które zawierają jedynie logikę biznesową lub zarządzają danymi. Elementy takie mogą być używane i współdzielone przez wiele komponentów.

4.11 Angular Material

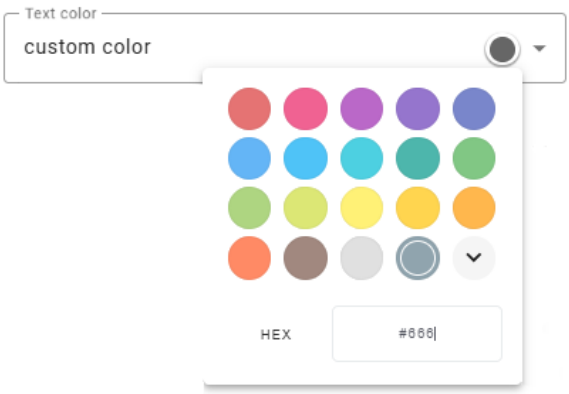
Angular Material to biblioteka komponentów dedykowanych do aplikacji tworzonych za pomocą *frameworka Angular.* Oferuje ona komplet elementów niezbędnych do wytworzenia graficznego interfejsu użytkownika aplikacji internetowej, w tym wiele typów kontrolek formularzy, takich jak pola wyboru, listy, ale również menu, okna dialogowe i całe układy ekranu (ang. *layouty*). Jest darmowa i udostępniana przez *Google LLC* w ramach licencji MIT [58], a przy tym niezwykle popularna oraz niezawodna [59].

Biblioteka ta została wykorzystana przy tworzeniu części prototypu jakim jest edytor animacji. Użyto między innymi komponenty takie jak:

- *Input* – pole tekstowe i numeryczne umożliwiające edycję parametrów animacji;
- *Select* – kontrolka umożliwiająca wybór opcji parametrów animacji z rozwijanej listy opcji;
- *Slide Toggle* – kontrolka umożliwiająca przełączanie parametru animacji w zakresie tak/nie;
- *Dialog* – komponent okna dialogowego z instrukcjami i kodem gotowym do użycia;
- *Tabs* – zakładki znajdujące się w oknie dialogowym;
- *Snackbar* – komponent wyświetlający użytkownikowi krótkie notyfikacje.

4.12 Bilbioteka ngx-colors

Biblioteka *ngx-colors* to komponent umożliwiający użytkownikowi wybór dowolnego koloru z palety kolorów i jest gotowy do ponownego użycia w aplikacjach internetowych tworzonych przy użyciu *frameworka Angular*. Narzędzie zachowuje styl kontrolek z biblioteki *AngularMaterial,* co zobrazowano na rysunku 4.4.



Rysunek 4.4. Komponent *ngx-colors* umożliwiający użytkownikowi wybór koloru, użyty wraz z kontrolką biblioteki *Angular Material* w edytorze animacji. Źródło: opracowanie własne

Biblioteka udostępniona jest w ramach licencji MIT [60]. Oznacza to, że można jej używać bez ograniczeń, a nawet dowolnie ją modyfikować, pod warunkiem zachowania informacji o jej autorze [61]. Autorem jest Agustin Torres publikujący kod jako Krone Corylus. Do wytworzenia edytora animacji, stanowiącego część niniejszej pracy, wykorzystano bibliotekę *ngx-colors* w wersji 3.5.2. Biblioteka jest instalowana przy pomocy menadżera pakietów NPM.

4.13 System kontroli wersji Git

System kontroli wersji (ang. *Version Control System*) to narzędzie służące do śledzenia zmian w plikach zawierających kod. Dzięki niemu można tworzyć wiele wersji kodu, równocześnie nad nimi pracować, a następnie łączyć je w kontrolowany sposób. Umożliwia także śledzenie historii zmian całości lub wybranych fragmentów plików. Pozwala to w prosty, ale i dokładny sposób, sprawdzić kto i w jakim czasie dokonał zmian oraz przywrócić dowolną wersję z przeszłości [62].

Przy wykonywaniu prototypu, stanowiącego element niniejszej pracy, wykorzystano najbardziej popularny systemu kontroli wersji - *Git*. System ten jest udostępniony w ramach licencji otwartoźródłowej (ang. *open source*) GPLv2, co oznacza, że jest darmowy w użyciu [63]. Cechuje się dużą szybkością, niezawodnością oraz dobrą integracją z narzędziami *IntelliJ*.

4.14 IntelliJ Webstorm

Kod źródłowy wszystkich elementów prototypu, stanowiącego element niniejszej pracy, napisany został przy użyciu *IntelliJ Webstorm*. Jest to jedno z najbardziej popularnych [64] na rynku zintegrowanych środowisk developerskich (ang. *Integrated Development Environment*), czyli narzędzi służących do:

- tworzenia kodu źródłowego;
- kompilowania, budowania i uruchamiania aplikacji;
- analizy błędów;
- refaktoryzacji kodu (zmianie jego struktury bez zmiany funkcjonalności) [65].

IntelliJ Webstorm jest to rozwiązanie komercyjne rozwijane przez firmę *JetBrains*. Istnieje jednak darmowa wersja społecznościowa *Community* oraz również bezpłatna wersja dedykowana dla środowisk akademickich [66].

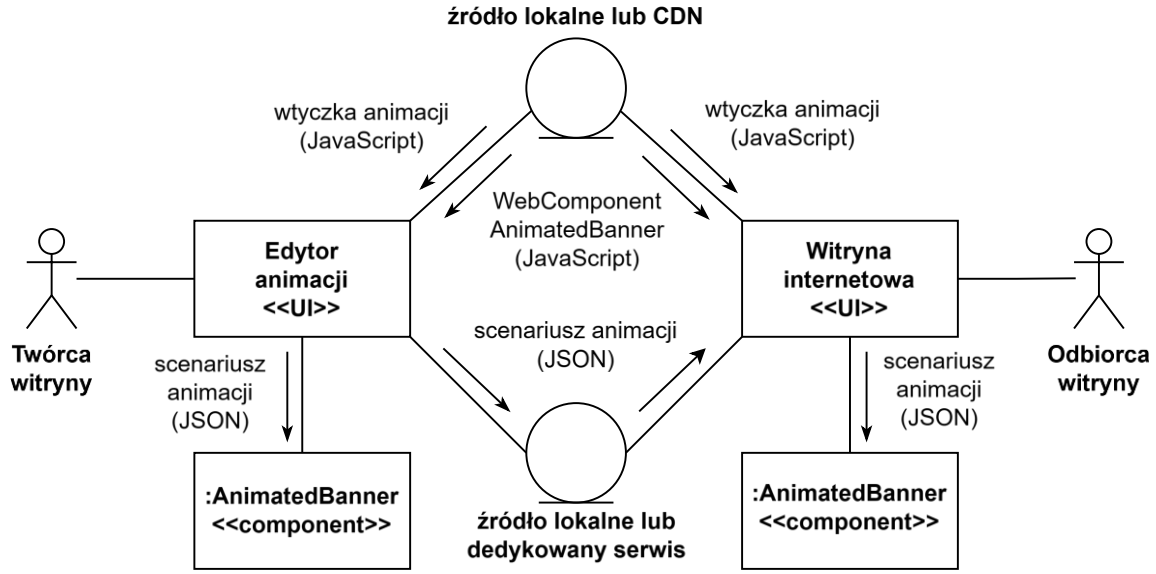
5. Opis implementacji prototypu

Rozdział ten zawiera opis rozwiązań implementacyjnych zastosowanych przy realizacji pracy.

5.1 Architektura rozwiązania

Rozwiązanie składa się z następujących elementów:

- *Web Componentu*, zwanego dalej **komponentem *AnimatedBanner*** umieszczanego w prosty sposób na dowolnej stronie internetowej. Generuje on animacje przy wykorzystaniu dedykowanych wtyczek animacji oraz technologii *Canvas API*. Animacje rysowane są na elemencie HTML *canvas* według przekazanego „scenariusza animacji".
- **dedykowanych wtyczek animacji** napisanych w języku *TypeScript* lub *JavaScript*, wykorzystywanych przez komponent *AnimatedBanner* do generowania animacji. System wtyczek zapewnia generyczność rozwiązania i możliwość rozszerzenia go o nowe efekty;
- **edytora animacji**, który służy do tworzenia i podglądu animacji o zadanych parametrach w czasie rzeczywistym. Animacje generowane są przez zawarty w edytorze komponent *AnimatedBanner*. Edytor wyświetla instrukcje implementacji w formie gotowej do skopiowania oraz umożliwia eksport „scenariusza animacji" w postaci kodu gotowego do użycia.



Rysunek 5.1. Schemat komunikacji pomiędzy elementami systemu.
Źródło: opracowanie własne

Definicja

Scenariuszem animacji nazywamy dane, utrwalone przy pomocy zapisu zgodnego ze standardem JSON, przekazywane do komponentu *AnimatedBanner* jako atrybut *animations.* Dane te służą komponentowi do wywoływania pożądanych animacji we właściwej kolejności. Scenariusz jest to lista obiektów, z których każdy zawiera przynajmniej klucz *animationName.* Klucz ten służy do wywołania animacji zarejestrowanej pod tą nazwą. Sposób rejestracji animacji opisano w pkt. 5.3.8. Następnie komponent przekazuje do odpowiedniej animacji właściwe parametry, które przypisane są do pozostałych kluczy danego obiektu. □

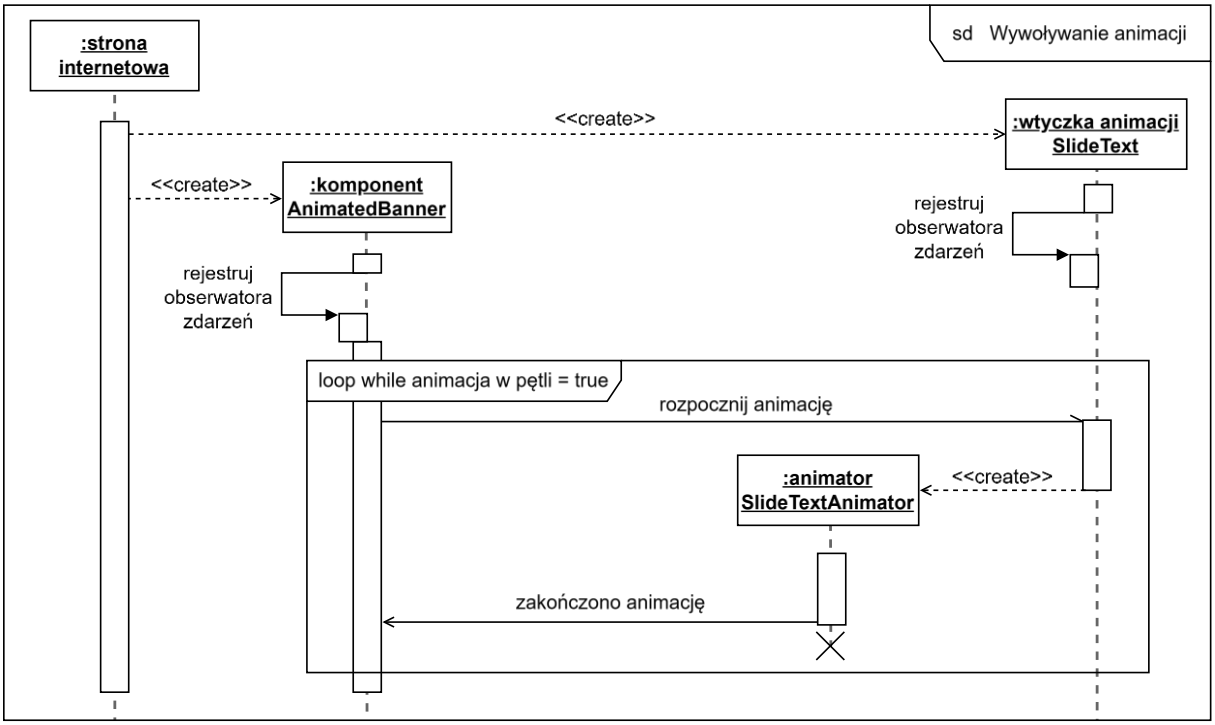
Komponent *AnimatedBanner* zarządza składowymi animacjami, uruchamia je zgodnie z otrzymanym scenariuszem animacji. Scenariusz animacji może przewidywać jedną lub wiele takich animacji. Mogą one być odgrywane jednocześnie, jedna po drugiej lub „nachodzić na siebie". Przed uruchomieniem poszczególnej składowej animacji musi zostać zarejestrowana odpowiednia dedykowana wtyczka animacji.

Definicja

Składową animacji nazywamy jeden z elementów całego scenariusza animacji, np. poszczególny animowany tekst lub poszczególny animowany obraz lub jakiekolwiek inną część scenariusza animacji obsługiwaną niezależnie przez wtyczkę animacji i utworzonym przez nią obiektem zwanym animatorem. □

Animatorem nazywamy obiekt utworzony przez wtyczkę animacji na żądanie komponentu *AnimatedBanner*. Obiekt ten otrzymuje w konstruktorze UID animacji, referencję do elementu HTML *canvas* oraz zestaw parametrów animacji. Jest on odpowiedzialny za odegranie składowej animacji, a dokładnie za rysowanie jej poszczególnych klatek na elemencie *canvas* należącym do komponentu *AnimatedBanner*. Po wykonaniu swojego zadania obiekt przestaje być użyteczny i zostanie usunięty z pamięci przez mechanizm *garbage collector.* □

Na rysunku 5.2 pokazano schemat wywoływania animacji przez komponent *AnimatedBanner*. Komunikacja odbywa się za pomocą zdarzeń użytkownika (ang. *custom events*). Zgodnie ze scenariuszem animacji, na żądanie komponentu powoływane są obiekty animatorów odpowiedniego typu. Komponent przekazuje im niezbędne parametry oraz referencje do elementu HTML *canvas.* Następnie nasłuchuje na zdarzenie zakończenia animacji. Kiedy otrzyma informację od wszystkich animatorów, że zainicjowane przez niego animacje zakończyły się, komponent uruchamia cały scenariusz animacji od początku.



Rysunek 5.2. Schemat wywoływania animacji przez komponent *AnimatedBanner*.
Źródło: opracowanie własne

5.2 Komponent AnimatedBanner

Podrozdział ten zawiera opis szczegółów implementacji komponentu *AnimatedBanner,* stanowiącego jeden z trzech elementów prototypu.

5.2.1 Sposób działania

Komponent *AnimatedBanner,* umieszczany jest, w kodzie HTML dowolnej strony internetowej jako *Web Component.* Strona internetowa przekazuje do komponentu zestaw atrybutów HTML. Szczegółowy opis parametrów podano w pkt. 5.2.3. Wraz z nimi przekazywany jest atrybut *animations,* który zawiera scenariusz animacji. Komponent *AnimatedBanner* przekazuje do wywoływanych przez siebie animacji referencję do elementu *canvas,* który stanowi cześć komponentu. Poszczególne animacje rysują na tym elemencie „swoją część zadania" np. tekst lub obraz we właściwym miejscu i skali. Ponieważ może istnieć wiele animacji składowych odgrywanych jednocześnie, na elemencie *canvas* może znaleźć się wiele tekstów czy obrazów. Komponent *AnimatedBanner* zajmuje się czyszczeniem tego elementu. W celu optymalizacji działania zarówno poszczególni animatorzy rysujący klatki animacji jak i komponent, który czyści element, korzystają z wbudowanej metody *window.requestAnimationFrame* (opisanej w podrozdziale 4.7).

5.2.2 Struktura klasy *AnimatedBannerComponent*

Komponent *AnimatedBanner* jest napisany w języku *TypeScript*. Użyty jest *framework Stencil.js* wspierający tworzenie *Web Componentów* oraz menadżer pakietów *Node Package Manager*. Wykorzystanie tych narzędzi pozwala znacznie uprościć proces wytwórczy oraz zapewnić poprawność i kompletność rozwiązania. Wbudowane w *Stencil.js* narzędzia pozwalają na budowanie zminimalizowanych paczek zawierających kompletną definicję komponentu, jego logikę oraz szablon HTML wraz z niezbędnymi stylami. Dodatkowo generowane są skrypty pozwalające zarejestrować znaczniki *Web Componentu* jako element niestandardowy (tzw. *custom element*). Przez przeglądarkę internetową uruchamiana jest właśnie ta zminimalizowana i skompilowana do języka *JavaScript* postać komponentu, a dokładnie skrypt rejestrujący. Sposób umieszczania komponentu na stronie internetowej i jego rejestrację opisano w pkt. 5.2.8.

Komponent *AnimatedBanner* składa się z pojedynczej klasy *AnimatedBannerComponent,* która jest zgodna z metodologią tworzenia *Web Componentów* zaproponowaną przez twórców *frameworka Stencil.js* [67].

Listing 5.1. Dekorator klasy *AnimatedBannerComponent*.
Źródło: opracowanie własne za [67]

```
@Component({
  tag: 'animated-banner-component',
  styleUrl: 'animated-banner-component.css',
  shadow: true,
})
export class AnimatedBannerComponent { … }
```

Jak pokazano na listingu nr 5.1, definicja klasy jest wzbogacona o dekorator *@Component,* za pomocą którego przekazywane są opcje zawierające m.in. znacznik, pod jakim komponent będzie zarejestrowany (w tym przypadku „*animated-banner-component*"), ścieżkę do pliku zawierającego style oraz flagę „*shadow*", zaznaczającą, że komponent będzie enkapsulowany – jego kod HTML będzie wydzielony w oddzielnym drzewie elementów tzw. *shadow DOM* [67].

Klasa *AnimatedBannerComponent* zawiera pola i metody, oba typy składowych klasy występują zarówno jako publiczne jak i prywatne.

Listing 5.2. Pola i metody klasy *AnimatedBannerComponent*.
Źródło: opracowanie własne

```
export class AnimatedBannerComponent {
  @Element() public el: HTMLElement
  @Prop() public background: string = 'transparent'
  @Prop() public height: number = 200
  @Prop() public width: number = 400
  @Prop() public matchParent: boolean = false
  @Prop() public animations: object[] = []
  @Event() public animationClicked: EventEmitter<object>
  @Listen('resize', {target: 'window'}) public handleResize(): void
  private canvas: HTMLCanvasElement
  private runningAnimations: any = []

  public render(): string
  public componentDidLoad(): void

  private setupCanvas(): void
  private setCanvasBackground(): void
  private setCanvasSize (): void
  private setCanvasOnclick(): void
  private startAllAnimations(): void
  private startAnimation(animation: Animation): void
  private startContinuousCleaningCanvasProcess(): void
  private setupAnimationFinishedListener(): void
  private stopLoopedAnimations(): void

  private get uid(): string
  private get aspectRatio(): number
  private get animationClickedEventData(): object
}
```

Jak pokazano na listingu nr 5.2, wewnątrz klasy komponentu zdefiniowano:

- pole *el* oznaczone dekoratorem *@Element.* Do tego pola *framework Stencil.js* przypisuje referencję do elementu HTML jakim jest cały *web component*. Dzięki temu, do elementu można odwołać się w kodzie w prosty sposób – poprzez *this.el* [68];
- pola oznaczone dekoratorem *@Prop*, które są wykorzystywane do przechowywania danych przekazanych do komponentu jako atrybuty HTML [69]. Mechanizm przekazywania wartości atrybutów i sposób ich wykorzystania w komponencie omówiono w pkt. 5.2.3;
- pole oznaczone dekoratorem *@Event.* Dekorator ten pozwala zdefiniować zdarzenie emitowane z *Web Componentu* na zewnątrz [70]. Sposób emitowania zdarzeń omówiono w pkt. 5.2.4;
- funkcję *handleResize* opatrzoną dekoratorem *@Listen.* Dekorator ten służy do przypisywania funkcji wykonywanych w przypadku wykrycia określonych zdarzeń (ang. *event*) pochodzących spoza komponentu [70]. Sposób obsługi zdarzenia typu *„resize”* przez komponent omówiono w pkt. 5.3.5;
- dwa pola prywatne: pole o nazwie *canvas*, wykorzystane do przechowywania referencji do elementu HTML typu *canvas*, będącego elementem komponentu oraz pole *runningAnimations*, przechowujące listę uruchomionych przez komponent animatorów;

- dwie metody publiczne wywoływane w określonych punktach cyklu życia komponentu. Omówione zostały szczegółowo w pkt. 5.2.6. Cykl życia definiowany przez *framework Stencil.js* omówiono w podrozdziale 4.8;
- metody prywatne służące do wykonywania zadań związanych z funkcjonalnością klasy, które omówiono szczegółowo w pkt 5.2.7;
- trzy metody prywatne typu *getter*, pełniące charakter pomocniczy, nie są znaczące dla projektowanego rozwiązania i pominięto ich opis.

5.2.3 Parametry komponentu i sposób ich przekazania

Framework Stencil.js ułatwia dostęp z poziomu kodu klasy napisanej w języku *TypeScript* do atrybutów HTML przypisanych do elementu HTML jakim jest *Web Component*. Jak pokazano na listingu 5.3, w klasie *AnimatedBannerComponent* zdefiniowane są pola publiczne opatrzone dekoratorem *@Props.* W trakcie trwania programu (ang. *runtime)*, do odpowiednich pól przypisywane są wartości odczytane z elementu HTML, a następnie przekształcone na właściwą postać [69]. W przypadku komponentu *AnimatedBanner* zdefiniowane jest pięć pól, do każdego z nich przypisany jest typ oraz wartość domyślna.

Listing 5.3. Pola klasy *AnimatedBannerComponent* opatrzone dekoratorem *@Prop*.
Źródło: opracowanie własne za [69]

```
export class AnimatedBannerComponent {
  @Prop() public background: string = 'transparent'
  @Prop() public height: number = 200
  @Prop() public width: number = 400
  @Prop() public matchParent: boolean = false
  @Prop() public animations: Animation[] = []
  …
}
```

Atrybuty HTML ustawiające tło (*background*), szerokość (*width*) i wysokość (*height*) ustawione na komponencie *AnimatedBanner,* są odczytane i następnie przekazane dalej do elementu *canvas.* Dzięki temu, możliwe jest ustawienie tła oraz szerokości i wysokości elementu *canvas*, niedostępnego z poziomu zwykłego drzewa DOM. Przekazanie wartości do elementu *canvas* odbywa się w metodzie *setupCanvas,* omówionej w pkt. 5.2.7. Końcowy efekt widoczny w strukturze strony internetowej zobrazowano na listingu 5.4.

Listing 5.4. Komponent *AnimatedBanner* w kodzie źródłowym strony HTML po kompilacji.
Źródło: opracowanie własne

```
<animated-banner-component width='500' height='250' background='black'>
  #shadow-root
    <canvas width='500' height='250' background='black'></canvas>
</animated-banner-component>
```

Atrybut *match-parent*, a następnie wartość pola *matchParent*, wykorzystuje się do ustalenia czy komponent ma wypełniać całą szerokość udostępnioną przez element będący jego rodzicem. W przypadku przekazania wartości *true* szerokość (atrybut w*idth*) i wysokość (atrybut h*eight*) traktowane są jedynie jako wartości służące do wyliczenia oczekiwanej proporcji. Proces ten opisano w pkt. 5.2.7 przy okazji opisu metody *setupCanvas.*

Najbardziej istotnym i jednocześnie najciekawszym z pól jest to o nazwie *animations*, przechowujące scenariusz animacji. Jak wyjaśniono w pkt. 5.2.1, scenariusz animacji to dane dotyczące składowych animacji, które mają zostać odtworzone przez komponent *AnimatedBanner*. Przechowywane są w postaci listy obiektów. W przypadku, kiedy do wytworzenia strony internetowej nie jest wykorzystywany żaden z nowoczesnych *frameworków* typu *Angular, Vue, React* lub *Ember*, plik HTML trafia do przeglądarki w nieprzetworzonej postaci. W takiej sytuacji nie ma możliwości zapisu wartości typu lista w kodzie HTML jak to ma miejsce w przypadku użycia jednego z wyżej wymienionych *frameworków*. Wartość typu lista należy ustawić za pomocą kodu *JavaScript*. Przykład ustawienia takiej wartości pokazuje listing 5.5. Instrukcja dostępna jest również w edytorze animacji.

Listing 5.5. Ustawienie wartości pola animations w tradycyjnym pliku HTML.
Źródło: opracowanie własne

```
<animated-banner-component
  id='banner1'
  width='500'
  height='250'
  match-parent='false'
  background='black'
></animated-banner-component>
<script>
  document.querySelector('#banner1').animations = animations;
</script>
```

W przypadku, kiedy do wytworzenia strony internetowej wykorzystywany jest nowoczesny *framework*, taki jak *Angular, Vue, React* lub *Ember*, zawartość stron internetowych definiowana jest w szablonach przypominających składnię HTML. Jednak przed trafieniem do przeglądarki internetowej są one kompilowane [71]. W takich przypadkach istnieje możliwość korzystania z tzw. *data bindingu* tj. mechanizmu przypisywania do atrybutów HTML zmiennych zdefiniowanych w kodzie zapisanym w języku *JavaScript*. Różnicę widać porównując listing 5.5 z listingiem 5.6, prezentującym sposób ustawienia takiej wartości w przypadku użycia *frameworka Angular*.

Listing 5.6. Ustawienie wartości pola animations w przypadku użycia *frameworka Angular*.
Źródło: opracowanie własne

```
<animated-banner-component
  width='500'
  height='250'
  match-parent='false'
  background='black'
  [animations]='animations'
></animated-banner-component>
```

5.2.4 Emitowanie zdarzeń z komponentu

Framework Stencil.js ułatwia emitowanie zdarzeń (ang. *event)* wykrytych wewnątrz *Web Componentu*, na zewnątrz. Komponent *AnimatedBanner* obsługuje zdarzenie kliknięcia na element HTML typu *canvas,* znajdujący się wewnątrz komponentu. Kod niezbędny do implementacji emisji zdarzenia pokazano na listingu 5.7. W obrębie klasy komponentu zdefiniowane jest pole opatrzone dekoratorem *@Event*. Do tego pola tego typu *framework* przypisuje obiekt typu *EventEmitter*. Do elementu HTML typu *canvas*, przypisano funkcję anonimową wykonywaną w przypadku wykrycia

kliknięcia. Funkcja ta wywołuje z kolej metodę *emit* na obiekcie *EventEmitter*, przypisanym do pola *animationClicked* [70].

Listing 5.7. Emitowanie zdarzeń z komponentu – kod przed kompilacją.
Źródło: opracowanie własne

```
export class AnimatedBannerComponent {
  …
  @Event() animationClicked: EventEmitter<object>
  …
  private setCanvasOnclick(): void {
    this.canvas.onclick = (): void => {
      this.animationClicked.emit(this.animationClickedEventData)
    }
  }
  …
}
```

Na listingu 5.8 pokazano fragmenty kodu w postaci skompilowanej przez *framework Stenci.js*. Kluczowe polecenie to *elm.dispatchEvent(ev)*, gdzie *elm* to element HTML jakim jest tworzony *Web Component*, a *ev* to *Custom Event* o nazwie „*animationClicked*". Dzięki temu, dowolna strona internetowa zawierająca komponent *AnimatedBanner* może nasłuchiwać na zdarzenie „*animationClicked*" bezpośrednio na tym komponencie.

Listing 5.8. Emitowanie zdarzeń z komponentu – kod po komplikacji.
Źródło: opracowanie własne przetworzone przez narzędzia *Stencil.js*

```
this.animationClicked = createEvent(this, 'animationClicked', 7);
…
const createEvent = (ref, name, flags) => {
const elm = getElement(ref);
return {
emit: (detail) => {
            …
return emitEvent(elm, name, getOpt(flags)
            …

const emitEvent = (elm, name, opts) => {
const ev = new CustomEvent(name, opts)
elm.dispatchEvent(ev);
};
```

5.2.5 Nasłuchiwanie przez komponent na zdarzenia z zewnątrz

Framework Stencil.js ułatwia nasłuchiwanie na zdarzenia (ang. *event)* pochodzące spoza komponentu. Wewnątrz klasy *Web Componentu* umieszcza się funkcję, która ma być wykonana w przypadku wystąpienia zdarzenia i oznacza się ja dekoratorem *@Listen*, a jako opcje przekazuje się nazwę zdarzenia i element HTML, na którym to zdarzenie ma zaistnieć. Skrypt rejestrujący definicję *Web Componentu*, wygenerowany przez *framework Stencil.js*, rejestruje także funkcję nasłuchującą na wskazane zdarzenie za pomocą standardowej metody elementu HTML *addEventListener*, która jest wykonywana na wskazanym elemencie HTML, najczęściej będzie to element *window*. Zarejestrowaną

w ten sposób funkcją wykonywaną w przypadku wystąpienia zdarzenia jest oczywiście ta funkcja oznaczona dekoratorem *@Listen* [70].

Listing 5.9. Pole klasy *AnimatedBannerComponent* opatrzone dekoratorem *@Listen*. Źródło: opracowanie własne

```
export class AnimatedBannerComponent {
  …
  @Listen('resize', {target: 'window'})
  public handleResize(): void {
    if (this.matchParent) {
    this.canvas.style.width = `${this.el.clientWidth}px`
    this.canvas.style.height = `${(this.el.clientWidth) / this.aspectRatio}px`
    }
  }
  …
  private get aspectRatio(): number {
    return this.width / this.height
  }
}
```

Jak pokazano na listingu 5.9, w opcjach przekazano nazwę zdarzenia, tutaj *„resize”* oraz element, na którym zaistniało zdarzenie, tutaj *„window”*. Do pola została przypisana funkcja *handleResize*. W efekcie tego, komponent *AnimatedBanner* obsługuje skalowanie okna przeglądarki, tj. zdarzenie *„resize”*, za pomocą wskazanej funkcji. W przypadku, gdy przekazany atrybut *match-parent* przyjmuje wartość logiczną *true*, funkcja ustawia szerokość elementu *canvas* równej szerokości *el*, czyli całego *Web Componentu* oraz wysokość wyliczoną ze współczynnika proporcji. Oznacza to, że przy skalowaniu okienka przeglądarki przez użytkownika, element *canvas* zajmuje możliwie duży obszar, ograniczony przez element będący rodzicem komponentu *AnimatedBanner*, zachowując przy tym ustanowione początkowo proporcje.

5.2.6 Metody publiczne – obsługujące cykl życia

Jak opisano w podrozdziale 4.8, *Stencil.js* wspiera koncepcję cyklu życia obiektu. Metody związane z cyklem życia komponentu wywoływane są w określonych momentach. Na listingu 5.10 uwidoczniono dwie metody publiczne, czyli metody cyklu życia.

Metoda *render* jest metodą obowiązkową dla każdego komponentu napisanego we *frameworku Stencil.js* i ma za zadanie zwrócić skompilowany szablon przypominający składnię HTML. Zapis szablonu HTML w formie funkcji napisanej w języku *TypeScript* lub *JavaScript* umożliwia wykorzystanie dyrektyw -specjalnych funkcji i zmiennych. Innymi słowy, jest to wzór, przepis na elementy HTML, które po wykonaniu funkcji i podmianie zmiennych na ich aktualne wartości będą stanowić docelową zawartość elementu *Web Component* i dopiero w takiej postaci zostaną dołączone do *Shadow DOM*. Możliwy efekt końcowy zaprezentowano wcześniej na listingu 5.4. Metoda jest wykonywana pierwszy raz po tym jak komponent jest dołączony do drzewa DOM, a następnie za każdym razem, gdy zmianie ulegną dane wykorzystywane w komponencie tj. pola oznaczone dekoratorem *@Prop* albo *@State* [54].

Z punktu widzenia *frameworka Stenci.js,* metoda *componentDidLoad* jest metodą opcjonalną i wywoływana jest jednokrotnie po tym jak komponent został załadowany i nastąpiło już pierwsze wykonanie metody *render* [54]. W przypadku komponentu *AnimatedBanner* metoda ta jest kluczowa. Jest wykorzystana do inicjalizacji i rozpoczęcia animacji poprzez wywołanie, kolejnych czterech metod prywatnych obsługujących główną funkcjonalność komponentu. Metody te opisano w pkt. 5.2.7.

Listing 5.10. Metody klasy *AnimatedBannerComponent.*
Źródło: opracowanie własne

```
export class AnimatedBannerComponent {
  …
  public render(): string {
    return <canvas id='canvas' ref={(el) => this.canvas = el}></canvas>
  }

  public componentDidLoad(): void {
    this.setupCanvas()
    this.startAllAnimations()
    this.setupAnimationFinishedListener()
    this.startContinuousCleaningCanvasProcess()
  }

  private setupCanvas(): void {
    this.setCanvasBackground()
    this.setCanvasSize()
    this.setCanvasOnclick()
  }

  private startAllAnimations(): void {
    this.animations.forEach(animation: Animation => {
      …
      this.startAnimation(animation)
      …
    })
  }

  private setupAnimationFinishedListener(): void {
    window.addEventListener('animation-finished', (event: CustomEvent) => {
      …
      this.startAllAnimations()
      …
    })
  }

  private startContinuousCleaningCanvasProcess(): void {
    …
    this.canvas.getContext('2d').clearRect(0, 0, canvasWidth, canvasHeight)
    requestAnimationFrame(this.startContinuousCleaningCanvasProcess.bind(this))
    …
  }

  private setCanvasBackground() { … }

  private setCanvasSize() { … }

  private setCanvasOnclick() { … }

  private startAnimation() { … }
  …
}
```

5.2.7 Metody prywatne – obsługujące funkcjonalność

Jak pokazano na listingu 5.10, metoda cyklu życia *componentDidLoad* wywołuje cztery metody prywatne – są to metody obsługujące funkcjonalność komponentu.

Metoda *setupCanvas*, jak sama nazwa wskazuje przygotowuje element *canvas*. Wywołuje trzy kolejne metody prywatne:

- *setCanvasBackground*, która przypisuje wartość przechowywaną w polu *@Prop background* do elementu HTML *canvas*, jako atrybut HTML określający tło tego elementu;
- *setCanvasSize*, która przypisuje wartości przechowywane w polach *@Prop width* i *@Prop height* do elementu HTML *canvas*, jako atrybuty HTML określające odpowiednio szerokość i wysokość tego elementu;
- *setCanvasOnclick*, której zadaniem jest umożliwienie emitowania zdarzeń typu kliknięcie. Metodę obrazuje listing 5.7, a sposób działania opisano w pkt. 5.2.4.

Z punktu widzenia implementacji funkcjonalności najbardziej istotne metody to metoda *startAllAnimations* oraz wywoływana przez nią metoda *startAnimation*. Obie metody pokazano na listingu 5.11. Sposób ich działania jest prosty. Wykonywana jest iteracja po liście animacji, które mają być odegrane według otrzymanego scenariusza animacji. Dla każdej z nich emitowane jest zdarzenie użytkownika (ang. *CustomEvent)*, które jest sygnałem do rozpoczęcia odpowiedniej animacji. Emisja zdarzenia jest opóźniona o czas podany w scenariuszu animacji – czyli parametr „*startAt”* podany dla każdej z animacji. Wraz ze zdarzeniem emitowane są dane zawierające unikalny ID animacji, referencję do elementu HTML *canvas* i zestaw parametrów dla danej animacji, pochodzący ze scenariusza animacji.

Listing 5.11. Metody *startAllAnimations* i *startAnimation*. Źródło: opracowanie własne

```
private startAllAnimations(): void {
  this.animations.forEach(animation: Animation => {
    const animationDelay: number = (animation.startAt || 0) * 1000
    setTimeout(() => this.startAnimation(animation), animationDelay)
  })
}

private startAnimation(animation: Animation): void {
  window.dispatchEvent(new CustomEvent(animation.animationName, {
    detail: { uid: this.uid, canvas: this.canvas, animationParams: animation }
  }))
}
```

Metoda prywatna *setupAnimationFinishedListener*, ma za zadanie ustanowić obserwatora zdarzeń (ang. *event listener*), który uruchomi cykl animacji od początku, kiedy już zakończy się poprzedni cykl i wszyscy animatorzy zgłoszą zakończenie działania. Sposób działania metody pokazano na listingu 5.12. Metoda ta wykonywana jest jednokrotnie po załadowaniu komponentu. Ustanowiony zostaje nasłuch na zdarzenia o nazwie „*animation-finished”*, które to jest emitowane przez każdego z animatorów po zakończeniu jego działania. Każde odebrane zdarzenie tego typu powinno zawierać unikalny ID składowej animacji, jeżeli takie ID istnieje na liście trwających w tej chwili składowych animacji przechowywanych przez komponent *AnimatedBanner* w polu *runningAnimations* – zostaje z niej usunięte. Gdy usunięte zostaną z niej wszystkie składowe animacje i lista jest pusta, wywołana zostaje metoda *startAllAnimations*. Lista trwających w tej chwili składowych animacji zostaje ponownie wypełniona i cykl animacji zaczyna się od początku.

Listing 5.12. Metoda *setupAnimationFinishedListener*. Źródło: opracowanie własne

```
private setupAnimationFinishedListener(): void {
  window.addEventListener('animation-finished', (event: CustomEvent): void => {
    let i = this.runningAnimations.findIndex(a => a.uid === event.detail.uid)
    if (i === -1) return
    else this.runningAnimations.splice(i, 1)
    if (this.runningAnimations.length === 0) {
      this.startAllAnimations()
    }
  })
}
```

Metoda *startContinuousCleaningCanvasProcess,* rozpoczyna pętlę procesu czyszczenia elementu HTML *canvas.* Element ten musi być wyczyszczony, po tym jak wszystkie animacje narysują już swoją klatkę animacji i przygotowany w ten sposób do rysowania kolejnej klatki.

5.2.8 Umieszczenie komponentu na stronie internetowej

Komponent *AnimatedBanner*, jak każdy *Web Component,* można w prosty sposób umieścić na dowolnej stronie internetowej zgodnej ze standardem HTML5 oraz pod warunkiem, że klient korzysta z przeglądarki internetowej, która wspiera technologię *Web Componentów*. W momencie pisania pracy wszystkie popularne przeglądarki wspierają to rozwiązanie (patrz rozdział 4.4) [46].

Użycie *Web Componentu* na stronie internetowej, musi zostać poprzedzone zarejestrowaniem jego definicji. Rejestracja definicji jest niezbędna, aby przeglądarka mogła właściwie zinterpretować napotkany fragment kodu HTML zawierający niestandardowe, dotychczas nieznane, znaczniki HTML. Do rejestracji służy metoda *define*, udostępniona przez interfejs *CustomElementRegistry* [47]. Technologię opisano w podrozdziale 4.4. W przypadku wykorzystania *frameworka Stencil.js* do tworzenia i generowania paczki dystrybucyjnej *Web Componentu*, obok skompilowanej wersji samego komponentu otrzymywany jest również skrypt, który automatycznie zarejestruje komponent [72]. Dzięki temu, aby użyć komponent *AnimatedBanner* na dowolnej stronie internetowej, wystarczy uruchomić skrypt rejestrujący ten *Web Component* przygotowany przez *framework Stencil.js*. Można to wykonać na jeden z poniższych sposobów:

- wczytanie pliku z kodem *JavaScript* z zewnętrznego źródła (tzw. CDN).

  W nagłówku (ang. *head*) strony pliku strony internetowej należy dodać fragment pokazany na listingu 5.13;

Listing 5.13 Wczytanie pliku rejestrującego komponent *AnimatedBanner* z zewnętrznego źródła CDN Źródło: opracowanie własne za [72]

```
<script type='module'>
  import { defineCustomElements }
    from 'https://unpkg.com/animated-banner/loader/index.es2017.js';
  defineCustomElements();
</script>
```

- uprzednie pobranie paczki z kodem JavaScript i wczytanie skryptu ze źródła lokalnego.

  Należy pobrać paczkę dystrybucyjną ze skompilowanym kodem komponentu *AnimatedBanner* ze źródła (https://unpkg.com/browse/animated-banner/), a następnie wczytać plik *index.es2017.js* z katalogu *loader.* Ewentualnie inny, odpowiedni do sytuacji plik;

- zainstalowanie pakietu NPM *AnimatedBanner*.

  O ile dostępny jest NPM (*Node Package Manager*) jest to polecany sposób na instalację paczki ze skompilowanym kodem komponentu *AnimatedBanner*. Wystarczy dodać pakiet do tworzonej strony internetowej, poprzez wykonanie polecenia „*npm install animated-banner --save”*, a następnie wczytać skrypt poprzez umieszczenie w nagłówku (ang. *head*) strony pliku strony internetowej fragmentu pokazanego na listingu 5.14.

Listing 5.14 Wczytanie pliku rejestrującego komponent *AnimatedBanner* z katalogu *node_modules*.
Źródło: opracowanie własne

```
<script type='module'
  src='node_modules/animated-banner/dist/animated-banner/animated-
banner.esm.js'>
</script>
```

W przypadku, gdy do wytworzenia strony internetowej używa się współczesnych *frameworków*, takich jak *Angular*, *React*, *Vue* czy *Ember*, sposób rejestracji komponentu może się różnić. Należy zapoznać się z instrukcją dedykowaną dla wykorzystywanego *frameworka*. Najbardziej aktualne instrukcje znajdują się na stronie oficjalnej internetowej *Stenci.js* [52]. Instrukcje i odpowiednie linki dostępne są również w aplikacji edytor animacji. W przypadku użycia *frameworka Angular* należy zainstalować pakiet NPM ze skompilowanym kodem komponentu *AnimatedBanner,* a następnie zarejestrować komponent w pliku *main.ts*, tak jak to pokazano na listingu 5.15.

Listing 5.15. Wczytanie pliku rejestrującego komponent *AnimatedBanner* z katalogu *node_modules*
Źródło: opracowanie własne za [73]

```
import {defineCustomElements} from 'animated-banner/loader'
…
void defineCustomElements(window)
```

Jeżeli *Web Component* jest już zarejestrowany, można używać go w kodzie HTML strony internetowej, tak jak każdy inny element HTML. Przykład użycia pokazano na listingu 5.16.

Listing 5.16. Użycie komponentu *AnimatedBanner* z kodzie HTML
Źródło: opracowanie własne

```
<animated-banner-component
  id='animated-banner-1'
  width='500'
  height='250'
  match-parent='false'
  background='black'
></animated-banner-component>
```

Na koniec należy jeszcze pamiętać o zarejestrowaniu definicji wykorzystywanych animacji, które zostało opisane w pkt. 5.3.8. Wszystkie te kroki opisane są także w instrukcjach dostępnych z poziomu narzędzia edytora animacji.

5.3 Wtyczka animacji (opis na podstawie przykładowej wtyczki)

Podrozdział ten zawiera opis szczegółów implementacji jednego z trzech elementów prototypu jakim jest dedykowana wtyczka animacji. Struktura opisana została na przykładzie wtyczki *SlideText*, jednak sposób działania mechanizmu i wykorzystania wtyczki jest jednakowy dla każdej z nich.

5.3.1 Sposób działania

Jak opisano w podrozdziale 5.1, komponent *AnimatedBanner* uruchamia poszczególne składowe animacje, zgodnie ze scenariuszem animacji. Scenariusz animacji może przewidywać jedną lub wiele takich animacji. Mogą one być odgrywane jednocześnie, jedna po drugiej lub „nachodzić na siebie". Przed uruchomieniem składowej animacji musi zostać zarejestrowana odpowiednia wtyczka animacji – sposób rejestracji wtyczki opisano w pkt. 5.3.8.

Rejestracja wtyczki animacji oznacza utworzenie obserwatora zdarzeń (ang. *event listener*), który w przypadku wykrycia znanego mu zdarzenia uruchomi odpowiedni obiekt animacji, zwany dalej animatorem. Animator jest odpowiedzialny za generowanie składowej animacji tj. rysowanie klatek animacji na elemencie HTML *canvas*. Mechanizm omówiono dokładniej w pkt. 5.3.4. Opcjonalnie tworzony jest też drugi obserwator zdarzeń, który służy do komunikacji z edytorem animacji i zwraca na żądanie listę parametrów używanych przez animatora danego typu. Mechanizm ten omówiono szerzej w pkt. 5.3.5.

Jak pokazano na listingu 5.11 wraz ze zdarzeniem rozpoczynającym składową animację, emitowane są dane zawierające unikalny ID składowej animacji, referencję do elementu HTML *canvas* i zestaw parametrów dla danej składowej animacji. Od momentu uruchomienia, animator generuje treści na elemencie HTML *canvas*, który jest częścią składową komponentu *AnimatedBanner*. Element ten może być dzielony pomiędzy wielu animatorów i każdy z nich rysuje „swoją część" klatki animacji. W odpowiednim momencie komponent *AnimatedBanner* czyści element *canvas* i przygotowuje go do rysowania kolejnej klatki animacji. W celu optymalizacji mechanizmu, generowanie klatek zgrane jest z częstotliwością odświeżania ekranu. Wykorzystano do tego funkcjonalność *requestAnimationFrame*, omówioną w podrozdziale 4.7. Mechanizm rysowania klatek animacji omówiono w pkt. 5.3.6. Po zakończeniu swojej pracy animator wysyła zdarzenie powiadamiające o tym fakcie komponent.

5.3.2 Struktura skryptu wtyczki animacji na przykładzie wtyczki *SlideText*

Wtyczkę napisano w języku *TypeScript* i skompilowano ją do języka *JavaScript*, aby była zrozumiała przez przeglądarkę internetową. Nic nie stoi na przeszkodzie, aby wtyczki tworzyć bezpośrednio przy użyciu języka *JavaScript* lub innego języka, który można skompilować do języka *JavaScript.* Aby wtyczka była kompatybilna z komponentem *AnimatedBanner* musi przynajmniej:

- posiadać pole statyczne *animationName* typu *string*;
- implementować interfejs *AnimatorInterface* tj. metody publiczne *animate*, która rozpoczyna animację oraz *stop*, która wymusza zakończenie animacji;
- zarejestrować rozpoczynającego składową animację obserwatora zdarzeń, który będzie zgodny ze schematem opisanym w pkt. 5.3.4.

Ponadto, aby wtyczka animacji była kompatybilna z edytorem animacji, musi rejestrować obserwatora zdarzeń zwracającego na żądanie listę parametrów używanych przez animatora do generowania składowej animacji. Mechanizm ten omówiono szerzej w pkt. 5.3.5. Przykładową strukturę skryptu wtyczki animacji pokazano na listingu 5.17. Plik składa się z definicji klasy oraz wywołań dwóch funkcji statycznych rejestrujących wspomnianych obserwatorów zdarzeń. Dla zachowania czytelności pominięto prywatne pola i metody pomocnicze służące głównie do przechowywania stanu składowej animacji i wyliczania jej postępu.

Na listingu 5.17 pokazano wykorzystanie interfejsu *AnimatedInterface* oraz dwóch szczególnych typów obiektów: *AmationOptions*, oraz *AnimationParams.* Opisano je w pkt. 5.3.3.

Listing 5.17. Przykładowa struktura skryptu wtyczki animacji na przykładzie wtyczki *SlideText*. Źródło: opracowanie własne

```
class SlideText implements AnimatorInterface {
  public static animationName = 'slide-text'
  public static animationVersion = '0.3.3'
  public static resourceUrl = 'https://unpkg.com/animated-banner@0.3.3/…'
  public static animationOptions: AnimationOptions
  …
  constructor(private uid: string,
              private canvas: HTMLCanvasElement,
              private params: AnimationParams)

  public static registerAnimator(): void
  public static registerParamsProvider(): void
  public animate(): void
  public stop(): void
  private runAnimation(): void
  private updateObjectPosition(): void
  private drawObjectOnCanvas(): void
  private resetObjectPosition(): void
  private sendAnimationFinishedEvent(): void
  …
}

ABSlideText.registerAnimator()
ABSlideText.registerParamsProvider()
```

5.3.3 Interfejsy oraz typy obiektów zdefiniowane na potrzeby wtyczek animacji

Klasa wtyczki animacji musi implementować interfejs *AnimatorInterface* (zaprezentowany na listingu 5.18), czyli posiadać metody publiczne *animate*, która rozpoczyna animację oraz *stop*, która wymusza zakończenie animacji. Interfejsy nie są obsługiwane przez język *JavaScript*, jednak przy tworzeniu wtyczki w tym języku także należy pamiętać o implementacji tych metod.

Listing 5.18. Interfejs *AnimatorInterface*. Źródło: opracowanie własne

```
export interface AnimatorInterface {
  animate: Function
  stop: Function
}
```

Na potrzeby usystematyzowania opcji animacji i ich obsługi przez edytor animacji zdefiniowano typ obiektu *AnimationOptions*. Definicję typu oraz przykładowy obiekt tego typu pokazano na listingu 5.19. Zapis ten jest charakterystyczny dla języka *TypeScript* i dopuszcza, aby obiekt posiadał dowolną liczbę kluczy typu *string*, do każdego z nich muszą być jednak obowiązkowo przypisane obiekty posiadające trzy pola obowiązkowe:

- *label* – typu *string*;
- *type* – literał równy jeden z wymienionych wartości (np. „*string*”);
- *defualt* – typu *string* lub *numer* lub *boolean;*

i trzy pola opcjonalnie:

- *min* – typu *number*;
- *max* – typu *number*;
- *options* – typu tablica napisów.

Listing 5.19. Definicja typu *AnimationOptions* i przykładowy obiekt tego typu. Źródło: opracowanie własne

```
type AnimationOptions = {
  [key: string]: {
    label: string,
    type: 'string' | 'number' | 'select' | 'color' | 'boolean' | 'image',
    default: string | number | boolean,
    min?: number,
    max?: number,
    options?: string[],
  }
}
public static animationOptions: AnimationOptions = {
  startAt: {
    label: 'Start at',
    type: 'number',
    default: 0,
    min: 1,
    max: 100
  },
  loop: {
    label: 'Loop animation',
    type: 'boolean',
    default: false
  },
  targetUrl: {
    label: 'Target URL',
    type: 'string',
    default: 'http:/something.com',
  }
  text: {
    label: 'Text',
    type: 'string',
    default: 'Hello Banner!',
  },
  direction: {
    label: 'Direction',
    type: 'select',
    default: 'left to right',
    options: ['left to right', 'right to left']
  },
  color: {
    label: 'Text color',
    type: 'color',
    default: '#666'
  },
}
```

Obiekt typu *AnimationParams* nawiązuje do obiektu *AnimationOptions,* porównując listingi 5.19 i 5.20, można zauważyć, że przechowuje on wybrane już opcje animacji. Definicja dopuszcza obiekt o kluczach typu *string* i wartościach typu *string* lub *number* lub *boolean,* przy czym wyróżnia trzy „specjalne" klucze: *startAt, loop* i *targetUrl* i wymaga od nich konkretnych typów, odpowiednio: *number, boolean* i *string.*

Listing 5.20. Definicja typu *AnimationParams* i przykładowy obiekt tego typu.
Źródło: opracowanie własne

```
type AnimationParams = {
  startAt?: number
  loop?: boolean
  targetUrl?: string
  [key: string]: string | number | boolean
}

let animationParams = {
  startAt: 1,
  loop: false,
  targetUrl: 'http:/something.com',
  text: 'Hello Banner!',
  direction: 'left to right',
  color: '#666'
}
```

5.3.4 Rozpoczynanie składowej animacji w wyniku zdarzenia

Metoda *registerAnimator* jest metodą wywoływaną przy wczytaniu pliku wtyczki animacji. Jak pokazano na listingu 5.21, wywołanie tej metody powoduje rejestrację obserwatora zdarzeń (ang. *event listener)* oczekującego na zdarzenie o nazwie równej nazwie animacji zapisanej w polu *animationName.* Za pomocą zdarzeń tego typu, komponent *AnimatedBanner* będzie uruchamiał kolejne składowe animacje, zgodnie ze scenariuszem animacji. Wraz ze zdarzeniem przekazane są dane zawierające unikalny numer animacji UID, referencje do elementu HTML *canvas* i parametry animacji. Po wykryciu zdarzenia, metoda tworzy obiekt animatora, przekazując mu dane odebrane ze szczegółów wydarzenia – polecenie *new SlideText(uid, canvas, params).* Następnie dodaje referencję obiektu animatora do listy trwających animacji (*runningAnimations.push(animator)*) i wywołuje metodę *animate,* rozpoczynając w ten sposób rysowanie składowej animacji.

Listing 5.21. Metoda rejestrująca obserwatora zdarzeń rozpoczynającego składową animację.
Źródło: opracowanie własne

```
public static registerAnimator(): void {
  window.addEventListener(this.animationName, (event: CustomEvent): void => {
    const uid: string = event.detail.uid
    const canvas: HTMLCanvasElement = event.detail.canvas
    const params: AnimationParams = event.detail.animationParams
    const animator = new SlideText(uid, canvas, params)
    canvas.runningAnimations.push(animator)
    animator.animate()
  })
}
```

5.3.5 Zwracanie listy obsługiwanych parametrów animacji

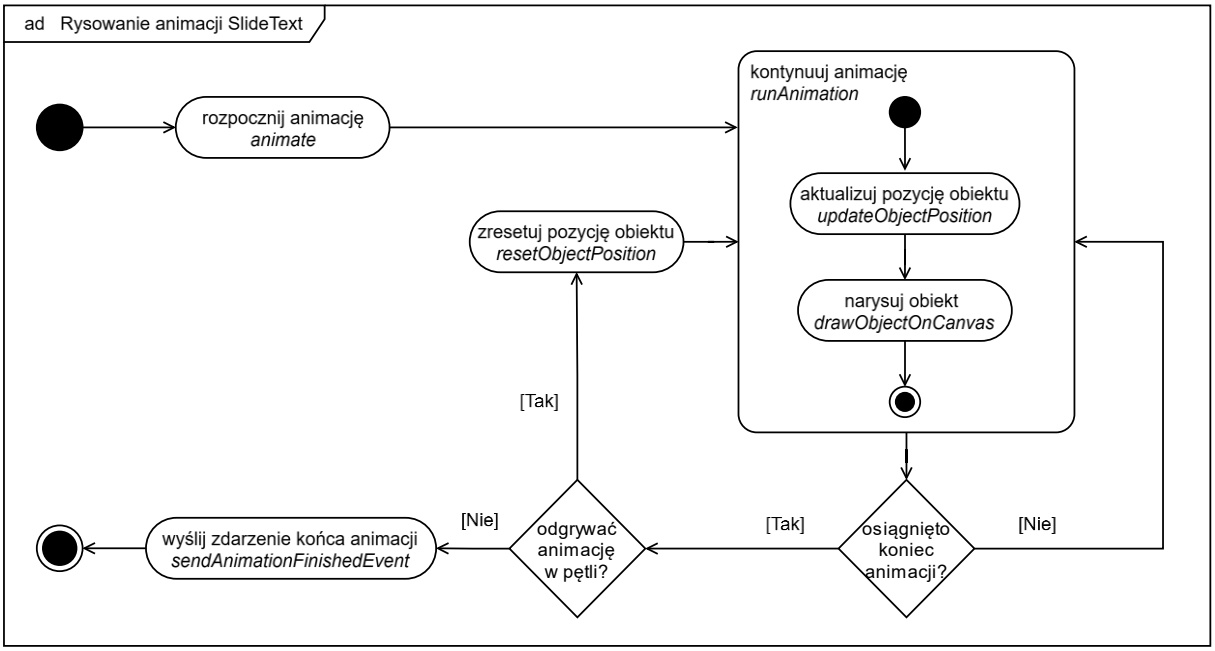
Metoda *registerParamsProvider*, pokazana na listingu 5.22, jest metodą wywoływaną przy wczytaniu pliku wtyczki animacji. Rejestruje ona obserwator zdarzeń (ang. *event listener)* oczekujący na zdarzenie o nazwie „*get-${this.animationName}-params*", gdzie *${this.animationName}* to nazwa równa nazwie animacji zapisanej w polu *animationName.* Zdarzenia tego typu wywoływane są przez edytor animacji. W odpowiedzi wysyłane jest zdarzenie zwrotne zawierające obiekt z nazwą animacji, adresem URL zasobu animacji oraz obiektem typu *AnimationOptions,* opisanym w pkt. 5.3.3 i zawierającym możliwe do wyświetlenia przez edytor animacji opcje dotyczące składowej animacji.

Listing 5.22. Metoda rejestrująca obserwatora zdarzeń zwracającego listę obsługiwanych parametrów. Źródło: opracowanie własne

```
public static registerParamsProvider(): void {
  window.addEventListener(`get-${this.animationName}-params`, () => {
    window.dispatchEvent(new CustomEvent('selected-animation-options', {
      detail: {
        animationName: SlideText.animationName,
        resourceUrl: SlideText.resourceUrl,
        animationOptions: SlideText.animationOptions
      }
```

5.3.6 Mechanizm rysowania klatek animacji na przykładzie wtyczki *SlideText*

Mechanizm pętli rysowania klatek animacji wygląda podobnie dla wszystkich stworzonych w ramach niniejszej pracy animacji i przedstawiony jest na rysunku 5.3.



Rysunek 5.3. Diagram aktywności rysowania animacji SlideText. Źródło: opracowanie własne

Proces rozpoczynany jest w metodzie publicznej *animate*, która wywołuje pierwszą iterację metody prywatnej *runAnimation.* W ramach rysowania klatki animacji, wywoływana jest metoda *updateObjectPosition,* służąca do aktualizacji stanu określającego pozycję animowanego tekstu oraz metoda *drawObjectOnCanvas,* która rysuje tekst na elemencie HTML *canvas.*

Następnie sprawdzany jest warunek osiągnięcia końca animacji. Jeżeli warunek jest spełniony, a składowa animacja nie ma być zapętlona, przerywana jest pętla rysowania klatek animacji i wywoływana jest metoda *sendAnimationFinishedEvent.* Mechanizm powiadamiania o zakończeniu animacji opisany jest w pkt. 5.3.7. Jeżeli warunek osiągniecia animacji jest spełniony, ale wybrana została opcja zapętlenia animacji, resetowany jest stan określający pozycję animowanego tekstu do wartości domyślnych i pętla rysowania klatek animacji trwa dalej.

Ważnym elementem mechanizmu jest ostatni wiersz metody *runAnimation* widoczny na listingu 5.23. Zawiera on polecenie *window.requestAnimationFrame(this.runAnimation.bind(this)).* Konstrukcja ta oznacza żądanie ponownego wywołania metody *runAnimation* przy następnym odświeżeniu ekranu. W ten sposób powstaje zoptymalizowana pętla rysowania klatek animacji z prędkością raz na każde odświeżenie ekranu. W większości przypadków odświeżanie wynosi 60Hz, czyli klatki rysowane są z prędkością 60 razy na sekundę [49]. Opis działania metody *window.requestAnimationFrame* znajduje się w podrozdziale 4.7.

Listing 5.23. Kod odpowiedzialny za rysowanie animacji w pętli.
Źródło: opracowanie własne

```
public animate(): void {
  this.runAnimation()
}

private runAnimation(): void {
  this.updateObjectPosition()
  this.drawObjectOnCanvas()

  if (this.endReached && !this.loop) {
    this.sendAnimationFinishedEvent()
    return
  }
  if (this.endReached && this.loop) {
    this.resetObjectPosition()
  }
  window.requestAnimationFrame(this.runAnimation.bind(this))
}

private drawObjectOnCanvas(): void {
  this.canvasContext.fillStyle = this.color
  this.canvasContext.fillText(this.text, this.textX, this.textY)
}
```

5.3.7 Powiadamianie o zakończeniu animacji

Po osiągnięciu warunku końca animacji (patrz rysunek 5.3) animator może zawiadomić komponent *AnimatedBanner* o zakończeniu swojego działania. Wykonuje to za pomocą zdarzenia o nazwie „*animation-finished*”. Komponent *AnimatedBanner* trzyma listę wszystkich odgrywanych składowych animacji, jeżeli wszystkie zakończą swoje zadania, może rozpocząć odgrywać cały scenariusz animacji od początku.

Listing 5.24. Metoda powiadamiająca o zakończeniu działania składowej animacji.
Źródło: opracowanie własne

```
private sendAnimationFinishedEvent(): void {
  window.dispatchEvent(new CustomEvent('animation-finished', {
    detail: {
      animationName: ABSlideText.animationName,
      uid: this.uid
    }
  }))
}
```

5.3.8 Wykorzystanie animacji na stronie internetowej

Aby komponent *AnimatedBanner* mógł wywołać animację danego typu, niezbędna jest uprzednia rejestracja odpowiedniej wtyczki animacji. Jak pokazano na listingu 5.17, wystarczające jest wczytanie skryptu zawierającego wtyczkę animacji. Rejestracja obserwatora zdarzeń, niezbędnego do startowania animacji, wykonywana jest automatycznie.

Wczytać i uruchomić skrypt zawierający wtyczkę animacji można wykonać na jeden z poniższych sposobów:

- wczytanie pliku z kodem *JavaScript* z zewnętrznego źródła (tzw. CDN).

  W nagłówku (ang. *head*) strony pliku strony internetowej należy dodać fragment analogiczny do tego pokazanego na listingu 5.25, gdzie źródło *src* to ścieżka do zasobu animacji;

Listing 5.25. Wczytanie pliku wtyczki animacji z zewnętrznego źródła CDN.
Źródło: opracowanie własne

```
<script
  type='module'
  src='https://unpkg.com/animated-banner/animations/slide-text.js'>
</script>
```

- uprzednie pobranie pliku z kodem JavaScript i wczytanie skryptu ze źródła lokalnego;
- zainstalowanie pakietu NPM *AnimatedBanner*

  O ile dostępny jest NPM (*Node Package Manager*), a wykorzystywana animacja należy do paczki podstawowej komponentu *AnimatedBanner*, wystarczy dodać pakiet do tworzonej strony internetowej, poprzez wykonanie polecenia „*npm install animated-banner –save”*, a następnie wzytać skrypt poprzez umieszczenie w nagłówku (ang. *head*) strony pliku strony internetowej fragmentu pokazanego na listingu 5.26;

Listing 5.26 Wczytanie pliku rejestrującego komponent *AnimatedBanner* z katalogu *node_modules.*
Źródło: opracowanie własne

```
<script
  type='module' src='node_modules/animated-banner/animations/slide-text.js'>
</script>
```

5.4 Edytor animacji

Podrozdział ten zawiera opis szczegółów implementacji edytora animacji, stanowiącego jeden z trzech elementów prototypu.

5.4.1 Sposób działania

Edytor animacji służy do tworzenia scenariusza animacji o zadanych parametrach i podglądu w czasie rzeczywistym animacji generowanej przez komponent *AnimatedBanner*. Edytor wyświetla także instrukcje implementacji w formie gotowej do skopiowania oraz umożliwia eksport scenariusza w postaci kodu gotowego do użycia.

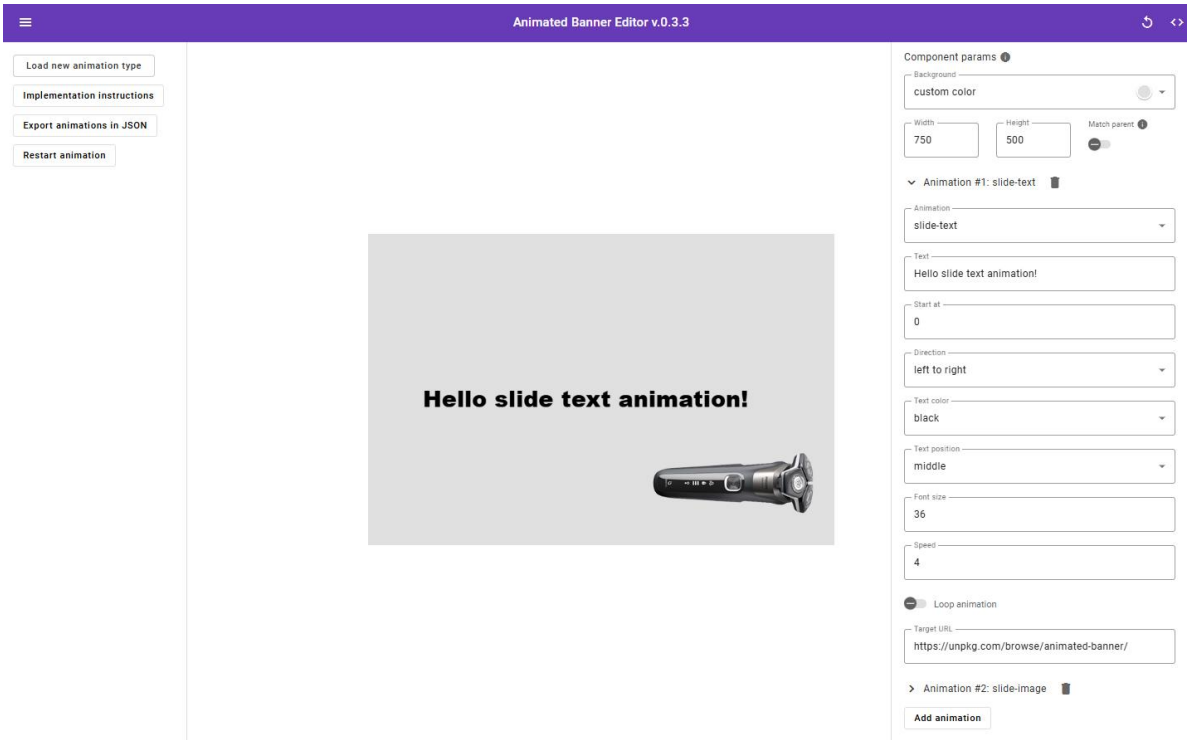
Jak pokazano na rysunku 5.4, ekran edytora animacji składa się z trzech paneli. Panel znajdujący się po lewej stronie ekranu to menu zawierające cztery pozycje:

- *Load new animation type*, służy do wczytania pliku z dodatkową definicją animacji i użycia jej w scenariuszu animacji;
- *Implementation instructions,* otwiera okno dialogowe zawierające instrukcję implementacji;
- *Export animations in JSON,* pozwala wyeksportować scenariusz animacji w formacie JSON;
- *Restart animation,* wymusza rozpoczęcie animacji od początku.

Panel środkowy zawiera komponent *AnimatedBanner,* który na bieżąco wyświetla przekazywany do niego scenariusz animacji.

Panel znajdujący się po prawej stronie to panel służący do edycji parametrów scenariusza animacji, który przekazywany jest do komponentu. Został opisany szerzej w pkt. 5.4.2.

W rozdziale 6 przedstawiono kompletny scenariusz użycia całego rozwiązania, w tym także sposób korzystania z edytora animacji i ustawiania parametrów składowej animacji.



Rysunek 5.4. Ekran edytora animacji.
Źródło: opracowanie własne

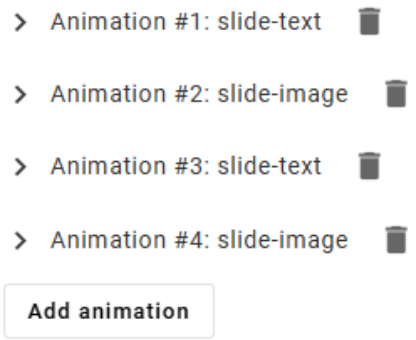
5.4.2 Panel służący do edycji parametrów animacji

Panel służący do edycji parametrów znajduje się po prawej stronie ekranu edytora. Realizuje on główną funkcjonalność aplikacji tj. pozwala na edycję parametrów scenariusza animacji. Każda zmiana parametrów jest widoczna na bieżąco w środkowym panelu aplikacji, gdzie komponent *AnimatedBanner* odgrywa przekazywany do niego scenariusz.

W górnej części panelu znajduje się sekcja parametrów dotyczących komponentu (ang. *Component params).* Sekcję tę ilustruje rysunek 5.5. Użytkownik ma do dyspozycji cztery kontrolki pozwalające na edycję parametrów komponentu tj. tła (ang. *background),* jego szerokości (ang. *width*), wysokości (ang. *height*) oraz przełącznika „dopasuj do rodzica" (ang. *match-parent)*, służącego do ustawiania czy komponent będzie wypełniał całą przestrzeń udostępnioną mu przez element nadrzędny. Ustawione tutaj wartości są przekazywane do komponentu jako atrybuty HTML, a następnie ustawiane na elemencie HTML *canvas.* Proces ten został omówiony w pkt. 5.2.3. Warto porównać przestawiający panel rysunek 5.5, z listingiem 5.30, odpowiedzialnym za jego wygenerowanie.

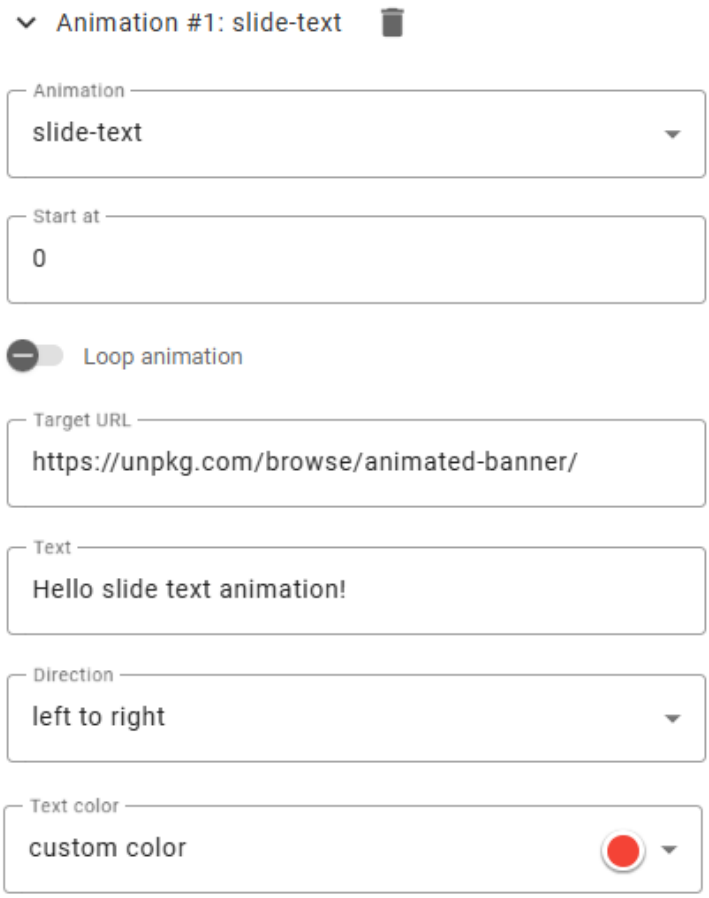
Rysunek 5.5. Sekcja parametrów dotyczących komponentu.
Źródło: opracowanie własne

Poniżej sekcji parametrów komponentu znajduje się sekcja zawierająca listę składowych animacji. Użytkownik może dodać kolejną składową za pomocą przycisku *Add animation* i usunąć dowolną z nich za pomocą przycisku zawierającego miniaturę śmietnika. Każda wyświetlona na liście składowa animacja może być widoczna w formie zminimalizowanej lub rozwiniętej. Na rysunku 5.6 zaprezentowano przykładową listę czterech składowych animacji w formie zminimalizowanej. Widoczne na liście składowe animacje są elementami całego scenariusza animacji.



Rysunek 5.6. Sekcja z listą składowych animacji w formie zminimalizowanej.
Źródło: opracowanie własne

Po rozwinięciu dowolnej z animacji, użytkownik ma dostęp do kontrolek służących do edycji parametrów danej składowej animacji. Przykładowy zestaw dostępny dla animacji typu *slide-text* pokazano na rysunku 5.7. Należy pamiętać, że zestawy kontrolek mogą być różne dla różnych typów składowych animacji.



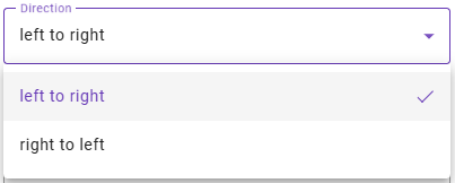
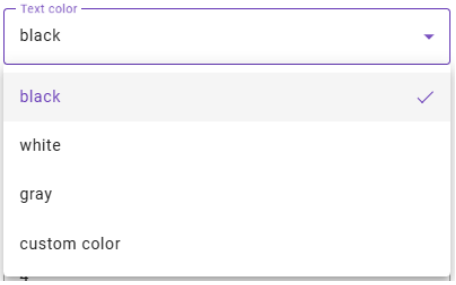
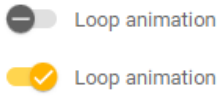
Rysunek 5.7. Zestaw kontrolek służący do edycji parametrów składowej animacji.
Źródło: opracowanie własne

5.4.3 Rodzaje kontrolek dostępnych do edycji parametrów

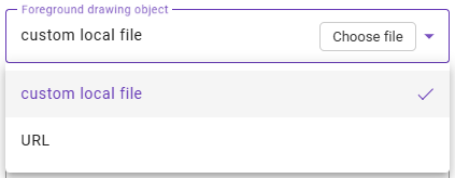
Poszczególne typy animacji udostępniają różne zestawy kontrolek. Kontrolki te dodawane są przez aplikację w sposób generyczny, na podstawie informacji opisujących jakie opcje są przez daną wtyczkę obsługiwane i zwracane przez wtyczkę na żądanie edytora animacji. Sposób odpytywania o dostępne opcje i odpowiadania na zapytanie opisany został w pkt. 5.3.5. Obiektem stanowiącym podstawę do konfiguracji kontrolek jest obiekt typu *AnimationOptions* pokazany na listingu 5.19. Warto go porównać z zestawem kontrolek wygenerowanym na jego podstawie i pokazanym na rysunku 5.7.

System został zaprojektowany i wykonany w sposób możliwie elastyczny i otwarty na rozbudowę. W chwili pisania pracy system oferuje kontrolki sześciu rodzajów, zgodnie z dopuszczalnymi wartościami klucza *type* przedstawionymi na listingu 5.19, są to: *string*, *number*, *select*, *color*, *boolean*, *image*. Oczywiście kontrolek danego typu może być w jednym zestawie wiele. Rodzaje obiektów opcji i odpowiadające im kontrolki przedstawione zostały w tabeli 5.1.

Tabela 5.1. Obsługiwane typów parametrów i odpowiadające im kontrolki.
Źródło: opracowanie własne

Obiekt opcji i przykładowe wartości	**Odpowiadająca mu kontrolka**
`{ type: 'string', label: 'Text', default: 'Slide text animation!' }`	
`{ type: 'number', label: 'Start at', default: 0, min: 1, max: 100, }`	 Kontrolka typu *number* umożliwia wybranie numeru. Zakres może zostać ograniczony z dołu i/lub z góry.
`{ type: 'select', label: 'Direction', default: 'left to right', options: ['left to right', 'left to right'] }`	
`{ type: 'color', label: 'Text color', default: 'black' }`	 Kontrolka typu *color* umożliwia wybranie z listy jednego z podstawowych kolorów lub specjalnej opcji „*custom color”*. Po wybraniu tej ostatniej użytkownikowi wyświetla się okno wyboru kolorów. Wykorzystano gotowe narzędzie *ngx-colors*, opisane w podrozdziale 4.12.
`{ type: 'boolean', label: 'Loop', default: false }`	 Kontrolka typu *boolean* przyjmuje formę przełącznika (ang. *switch)* który zwraca wartość pozytywna lub negatywną.

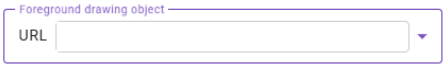
```
{
 type: 'image',
 label: 'Foreground drawing object',
 default: ''
}
```



Kontrolka typu *image* umożliwia wybranie jednej z dwóch opcji specjalnych:

- *Custom local file* – pozwala na wybranie pliku z lokalnego zasobu;

- *URL* – pozwala na wpisanie adresu URL obrazka będącego zasobem zdalnym.



5.4.4 Okno dialogowe zawierające instrukcję implementacji

Po wybraniu opcji *Implementation instructions* z menu znajdującego się po lewej stronie ekranu otwarte zostaje okno dialogowe, zaprezentowane na rysunku 5.8, zawierające instrukcję implementacji rozwiązania na dowolnej stronie internetowej. Zawiera ono sześć zakładek, z czego dwie pierwsze z nich to:

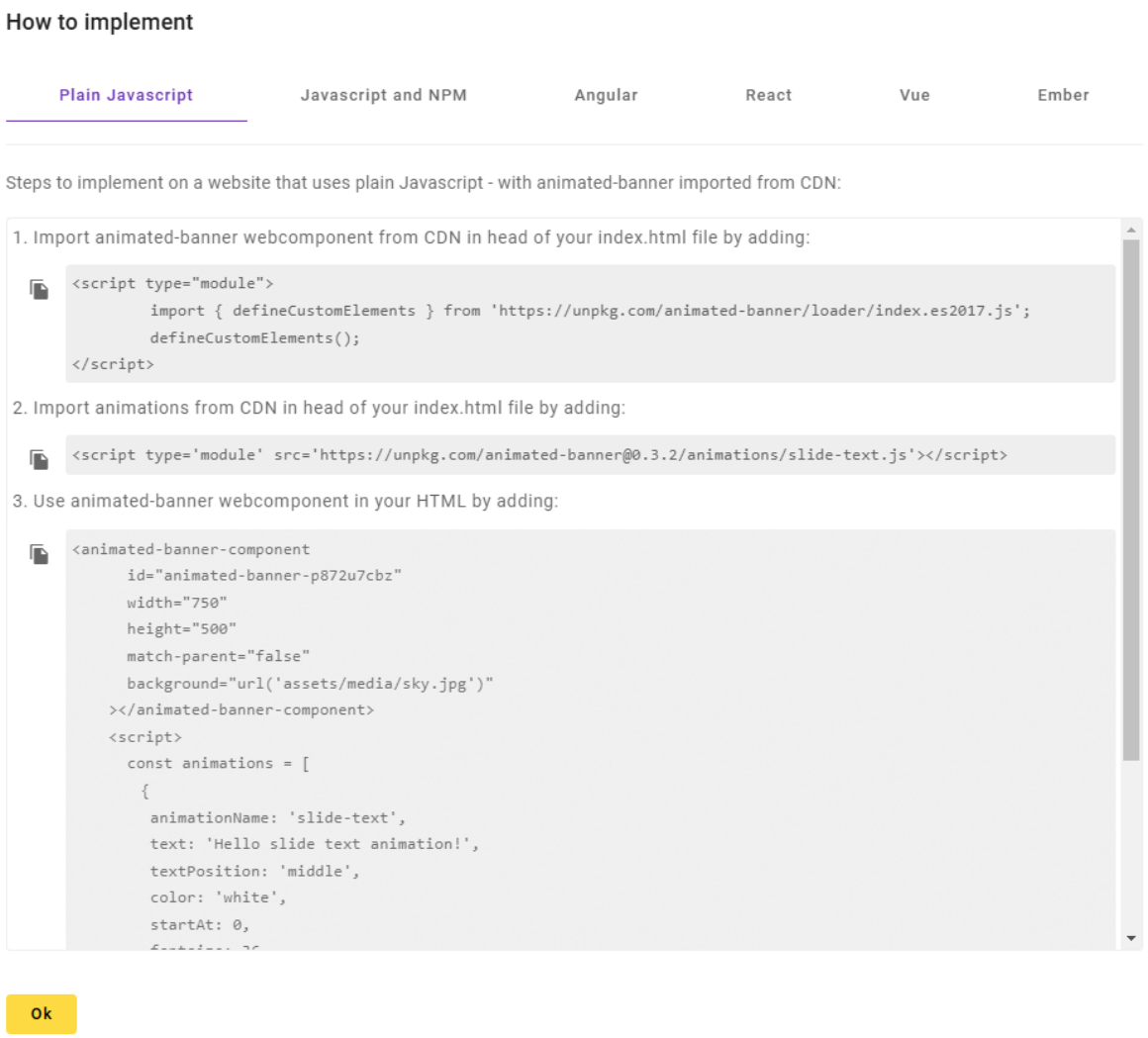
- *Plain JavaScript* – zawiera instrukcję implementacji rozwiązania dla przypadku tworzenia strony internetowej w czystym środowisku *JavaScript* bez użycia żadnego nowoczesnego *frameworka*;
- *JavaScript and NPM* – zawiera instrukcję implementacji rozwiązania dla przypadku tworzenia strony internetowej w środowisku JavaScript, bez użycia *frameworka*, ale z użyciem menadżera pakietów NPM (*Node Package Manager*).

Pozostałe cztery, zawierają instrukcję implementacji rozwiązania dla przypadku tworzenia strony internetowej przy użyciu jednego z *frameworków*, i są to odpowiednio zakładki:

- *Angular;*
- *React;*
- *Vue;*
- *Ember*.

Po wybraniu odpowiedniej zakładki, wyświetlana jest szczegółowa instrukcja użycia komponentu dla wybranego przypadku. Instrukcje podzielone są na poszczególne kroki, jakie należy wykonać, aby użyć komponentu *AnimatedBanner* na stronie internetowej. Opisany jest także sposób na przekazanie scenariusza animacji utworzonego za pomocą edytora animacji. Użytkownikowi umożliwione zostało kopiowanie do schowka systemowego niezbędnego kodu. Kopiowanie odbywa się przy pomocy przycisków oznaczonych ikoną dokumentu. Przyciski znajdują się obok każdego z fragmentów kodu.

Przykład użycia okna dialogowego zawierającego instrukcję implementacji zaprezentowano w rozdziale 6, mechanizm umieszczania komponentu na stronie internetowej opisano też w pkt. 5.2.8.

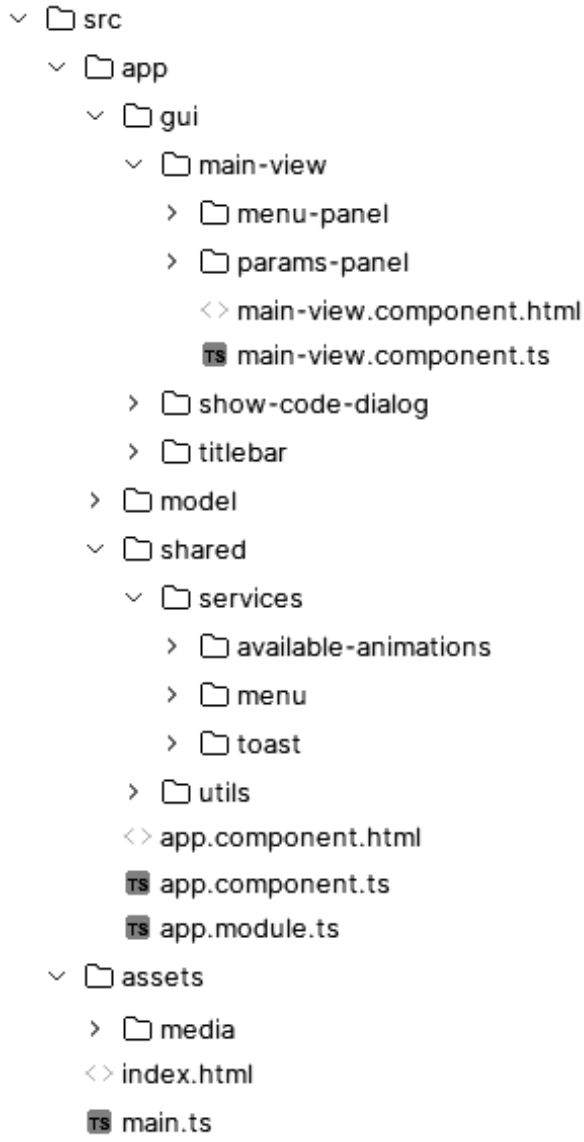


Rysunek 5.8. Okno dialogowe zawierające instrukcje implementacji.
Źródło: opracowanie własne

5.4.5 Struktura aplikacji

Edytor Animacji jest aplikacją działającą w oknie przeglądarki internetowej. Została napisana w języku *TypeScript*, przy użyciu *frameworka Angular* oraz menadżera pakietów *Node Package Manager*. Do budowy graficznego interfejsu użytkownika wykorzystano, dedykowaną dla użytego *frameworka,* bibliotekę komponentów *Angular Material*. Do umożliwienia intuicyjnego wyboru kolorów wykorzystano gotowe narzędzie *ngx-colors*, opisane w podrozdziale 4.12.

Zgodnie z koncepcją *frameworka Angular*, kod podzielony jest na komponenty i serwisy odpowiedzialne za poszczególne funkcjonalności [74]. Ponieważ aplikacja nie jest rozbudowana, wszystkie jej komponenty działają w obrębie jednego modułu. Jednak dla czytelności struktury aplikacji, pliki rozmieszczono w różnych folderach, według obszarów ich odpowiedzialności.



Rysunek 5.9. Struktura folderów i plików aplikacji edytor animacji.
Źródło: opracowanie własne

Jak można zauważyć na rysunku 5.9, główny katalog plików źródłowych *src* zawiera katalog *assets* (zasoby typu obrazki), oraz katalog *app* (kod aplikacji). Zawiera on także pliki *index.html* oraz *main.ts* – są to pierwsze pliki wczytywane przez każdą aplikację utworzoną we *frameworku Angular* [75]. Zgodnie z instrukcją użycia komponentu *AnimatedBanner* w aplikacjach wytworzonych przy użyciu tego *frameworka*, do pliku *main.ts* dodano niezbędny fragment kodu, wczytujący definicję komponentu. Kod pokazany został na listingu 5.15 i opisany w pkt 5.2.8

5.4.6 Istotne komponenty aplikacji edytor animacji

Jak pokazano na listingu 5.27, ciało (ang. *body*) strony internetowej *índex.html* zawiera wyłącznie jeden element HTML, posiadający niestandardowe znaczniki *app-root*. W to miejsce *framework* renderuje główny komponent aplikacji tj. *app.component*.

Listing 5.27. Zawartość pliku *index.html* aplikacji edytor animacji
Źródło: opracowanie własne

```
<!doctype html>
<html>
  <head>…</head>
  <body>
    <app-root></app-root>
  </body>
</html>
```

Szablon HTML głównego komponentu również jest bardzo prosty. Jak pokazano na listingu 5.28 zawiera wyłącznie dwa elementy, również niestandardowe. Odpowiadają one komponentom paska tytułu (*titlebar)* oraz głównego widoku (*main-view*), stworzonym na potrzeby aplikacji. Pliki tych komponentów umieszczone zostały w folderze *gui*. Jak widać na rysunku 5.9, trzecim elementem znajdującym się bezpośrednio w tym folderze jest komponent okna dialogowego z instrukcjami implementacji (*show-code-dialog*).

Listing 5.28. Zawartość szablonu HTML komponentu *app-root*
Źródło: opracowanie własne

```
<app-titlebar></app-titlebar>
<app-main-view></app-main-view>
```

Z punktu widzenia działania aplikacji, komponent *titlebar* można uznać za mało istotny, a jego opis pominąć. Istotnym elementem jest natomiast komponent *main-view*. Budowa jego szablonu została pokazana na listingu 5.29. Widać, że podobnie jak ekran aplikacji pokazany na rysunku 5.4 (opisany w pkt. 5.4.1), tak szablon komponentu składa się z trzech głównych elementów:

Listing 5.29. Uproszczony kod szablonu HTML komponentu *main-view.*
Źródło: opracowanie własne

```
<app-menu-panel></app-menu-panel>

<div id="animated-banner-container">
    …
    <animated-banner-component
      [background]="params.background"
      [matchParent]="params.matchParent"
      [height]="params.height"
      [width]="params.width"
      [animations]="params.animations"
      (animationClicked)="onAnimatedBannerClick($event)"
    ></animated-banner-component>
    …
</div>

<app-params-panel
  (paramEmitter)="params=$event"
></app-params-panel>
```

- komponentu panelu menu (*menu-panel*), który odpowiada za wyświetlanie panelu menu po lewej stronie ekranu, z resztą aplikacji komunikuje się poprzez serwis;
- sekcji zawierającej komponent *AnimatedBanner* (*div id=animated-banner-container*), który odpowiada za wyświetlenie środkowego panelu. Do zawartego w nim komponentu przekazywane są atrybuty HTML których wartości pochodzą z obiektu *params*;
- komponentu panelu do edycji parametrów scenariusza animacji (*params-panel*) – odpowiada za wyświetlanie panelu z zestawem kontrolek po prawej stronie ekranu. Za jego pomocą, jak opisano w pkt. 5.4.2, użytkownik ustawia parametry scenariusza animacji.

Jak pokazano na listingu 5.29, obiekt z danymi zawierającymi wybrane opcje emitowany jest przez komponent *params-panel* za pomocą wydarzenia *paramEmitter,* a jego wartość przypisywana jest do obiektu *params,* będącego polem komponentu *main-view*. Dane z tego obiektu przekazywane są następnie jako atrybuty HTML do komponentu *AnimatedBanner*. Dzięki temu mechanizmowi opcje wybrane przez użytkownika za pomocą kontrolek znajdujących się w panelu po prawej stronie ekranu są na bieżąco przekazywane do komponentu i użytkownik ma stały podgląd na efekt swoich działań.

Komponent *params-panel* i zagnieżdżony w nim komponent *animation-params* są dosyć rozbudowane, dlatego aby wyjaśnić sposób ich działania na listingach 5.30 i 5.31 pokazano jedynie uproszczone schematy szablonów tych komponentów. Łatwo można wyodrębnić kod odpowiedzialny za opisaną w pkt. 5.4.3 sekcję czterech parametrów dotyczących całego komponentu (ang. *Component params*), sekcję zawierająca listę składowych animacji (lista komponentów *animation-params*) oraz przycisk *Add animation,* służący do dodawania kolejnej składowej animacji.

Listing 5.30. Uproszczony schemat szablonu HTML komponentu *params-panel.*
Źródło: opracowanie własne

```
<h4>Component params</h4>

<app-background-control
  [(ngModel)]="params.background">Background
</app-background-control>

<app-number-control
  [(ngModel)]="params.width">Width
</app-number-control>

<app-number-control
  [(ngModel)]="params.height">Height
</app-number-control>

<app-boolean-control
  [(ngModel)]="params.matchParent">Match parent
</app-boolean-control>

<ng-container *ngFor="let animationParams of params.animations; let i=index">
  <h4>Animation #{{i+1}}</h4>
  <button (click)="removeAnimation(i)">Remove</button>
  <app-animation-params
    [animationParams]="animationParams"
    (onParamsChange)="onAnimationParamsChange($event)">
  </app-animation-params>
</ng-container>

<button (click)="addNextAnimation()">Add animation</button>
```

Listing 5.31. Uproszczony schemat szablonu HTML komponentu *animation-params.*
Źródło: opracowanie własne

```
<app-select-control
  [(ngModel)]="animationParams.animationName"
  (ngModelChange)="getAnimationOptions()">Animation
</app-select-control>

<ng-container *ngFor="let key of animationOptions">
  <app-text-control
    *ngIf=" animationOptions[key].type==='string'"
    [(ngModel)]="animationParams[key]">animationOptions[key].label
  </app-text-control>

  <app-number-control
    *ngIf="animationOptions[key].type==='number'"
    [(ngModel)]="animationParams[key]">animationOptions[key].label
  </app-number-control>

  <app-select-control
    *ngIf="animationOptions[key].type==='select'"
    [(ngModel)]="animationParams[key]">animationOptions[key].label
  </app-select-control>

  <app-color-control
    *ngIf="animationOptions[key].type==='color'"
    [(ngModel)]="animationParams[key]">animationOptions[key].label
  </app-color-control>

  <app-boolean-control
    *ngIf="animationOptions[key].type==='boolean'"
    [(ngModel)]="animationParams[key]">animationOptions[key].label
  </app-boolean-control>

  <app-image-control
    *ngIf="animationOptions[key].type==='image'"
    [(ngModel)]="animationParams[key]">animationOptions[key].label
  </app-image-control>
</ng-container>
```

Na listingu 5.31 pokazano mechanizm generycznego dodawania kontrolek na podstawie obiektu typu *AnimationOptions* (patrz listing 5.19 opisany w pkt. 5.3.3). Można zauważyć, że występuje tu iteracja po kluczach tego obiektu, czyli również obiektach zawierających przynajmniej trzy pola: *type*, *label*, *default*. Jak opisano w pkt. 5.4.3 pole *type* przyjmuje jedną z sześciu dopuszczalnych wartości i w zależności od tej wartości dodawany jest właściwy typ kontrolki. Wartość każdej kontrolki powiązana jest za pomocą mechanizmu *two-way-binding* z wartością zapisaną w obiekcie typu *AnimationParams* pod właściwym, odpowiadającym opcji kluczem.

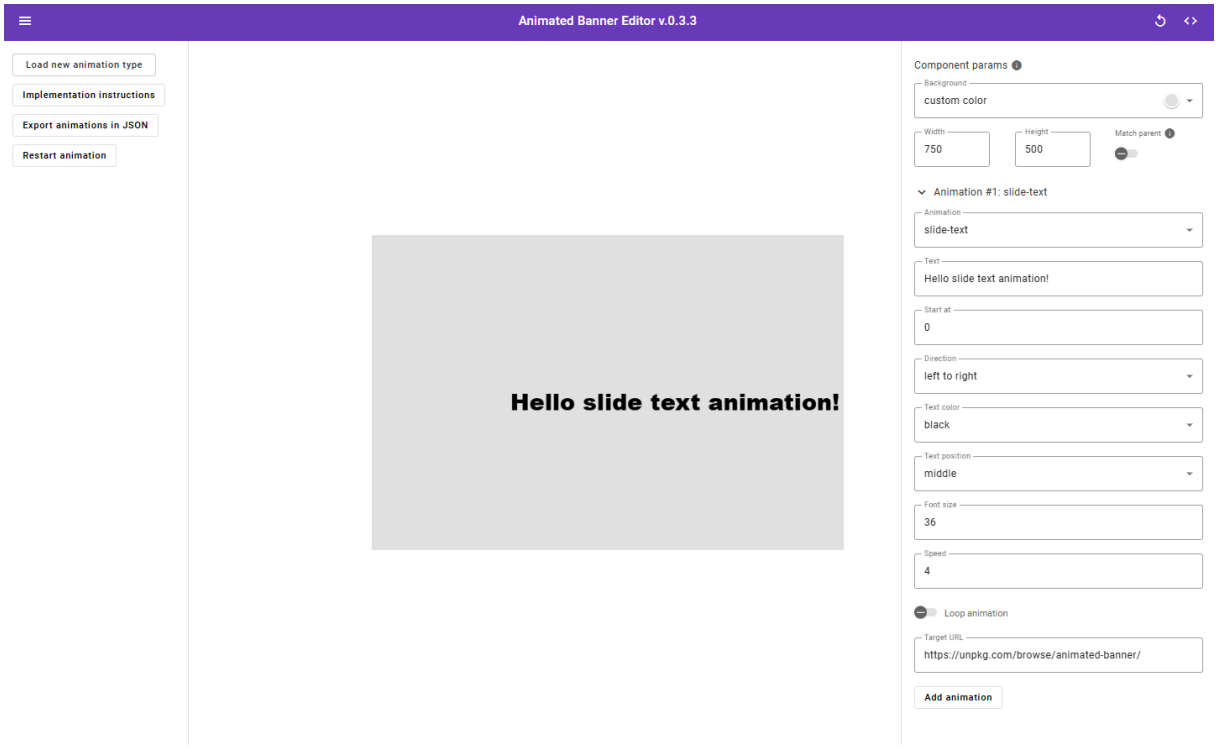
6. Przykładowe wykorzystanie prototypu

Rozdział ten przedstawia przykładowy, kompletny scenariusz użycia „Elastycznego systemu tworzenia animacji dla portali internetowych". Prototyp powstał w ramach niniejszej pracy magisterskiej w celu ułatwienia procesu umieszczania i aktualizacji treści animacji na stronach internetowych. Rozwiązanie skierowane jest dla osób nieposiadających wiedzy i umiejętności programistycznych.

6.1 Wytworzenie animacji za pomocą edytora

Użytkownik uruchamia aplikację edytor animacji w celu opracowania grafiki, która następnie zostanie umieszczona na stronie internetowej. Jest to aplikacja internetowa typu SPA działająca w oknie przeglądarki internetowej. Jak widać na rysunku 6.1, interfejs użytkownika składa się z trzech części:

- panelu znajdującego się po lewej stronie, który zawiera cztery przyciski akcji;
- panelu środkowego zawierającego komponent *AnimatedBanner*, który na bieżąco odgrywa przekazywany do niego scenariusz animacji;
- panelu znajdującego się po prawej stronie, służącego do edycji parametrów scenariusza animacji.



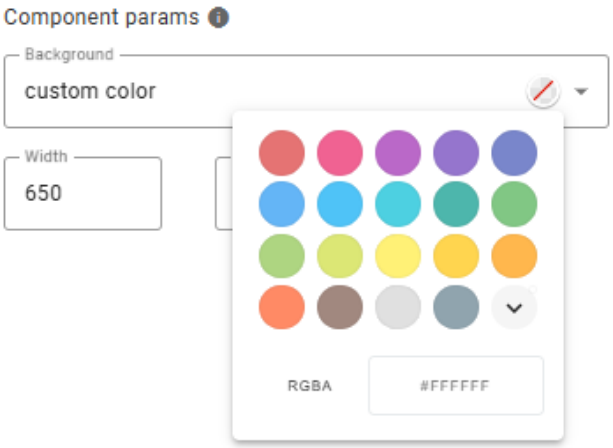
Rysunek 6.1. Widok interfejsu użytkownika edytora animacji.
Źródło: opracowanie własne

Za pomocą kontrolek w sekcji dotyczącej parametrów komponentu (ang. *Component params*), znajdującej się w prawym górnym rogu ekranu, użytkownik ustawia wartości atrybutów elementu HTML *canvas*. Animacja generowana jest w ramach tego elementu. Jego szerokość (ang. *width*) oraz wysokość (ang. *height*) definiuje wymiary prostokątnego obszaru, na którym będzie wyświetlana animacja. Użytkownik może zaznaczyć opcję „dopasuj do rodzica" (ang. *match-parent*). Wybranie tej

opcji sprawi, że komponent będzie wypełniał całą przestrzeń udostępnioną mu przez element nadrzędny przy zachowaniu proporcji wskazanych za pomocą atrybutów „szerokość” i „wysokość”.

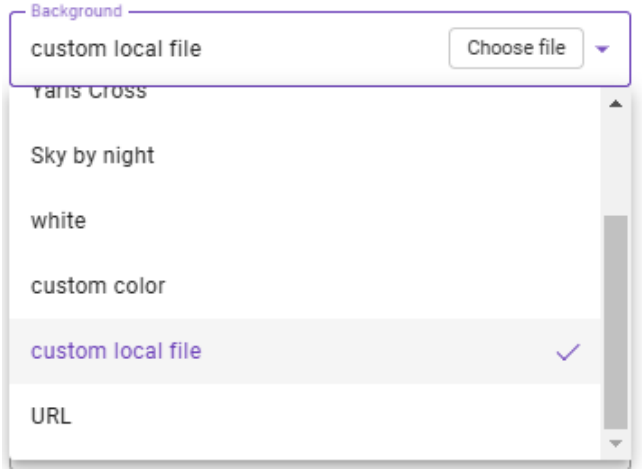
Lista opcji tła animacji (ang. *background*), udostępnia wybór kilku predefiniowanych kolorów i przykładowych obrazów, a także trzy opcje specjalne:

- kolor niestandardowy (ang. *custom color*) – wyświetla dodatkowy komponent, przedstawiony na rysunku 6.2, umożliwiający użytkownikowi wybór dowolnego koloru tła z palety kolorów;
- plik lokalny (ang. *custom local file*) – umożliwia wybór tła z pliku zapisanego lokalnie na komputerze użytkownika;
- URL – umożliwia podanie adresu URL wskazującego na obrazek znajdujący się w Internecie.



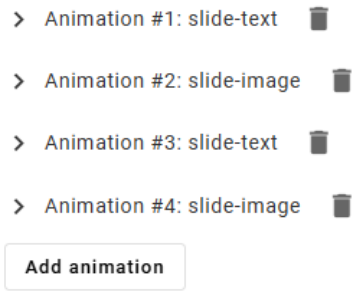
Rysunek 6.2. Widok komponentu umożliwiającego użytkownikowi wybór dowolnego koloru.
Źródło: opracowanie własne

Opisywany scenariusz zakłada, że użytkownik ma przygotowane tło animacji na swoim komputerze i wybiera opcję „plik lokalny” (ang. *custom local file*). W takim przypadku pojawia się przycisk służący do wyboru pliku (ang. *choose file*). Sytuację tę przedstawiono na rysunku 6.3. Po jego naciśnięciu użytkownik wybiera plik ze źródła lokalnego za pomocą systemowego okna wyboru pliku.



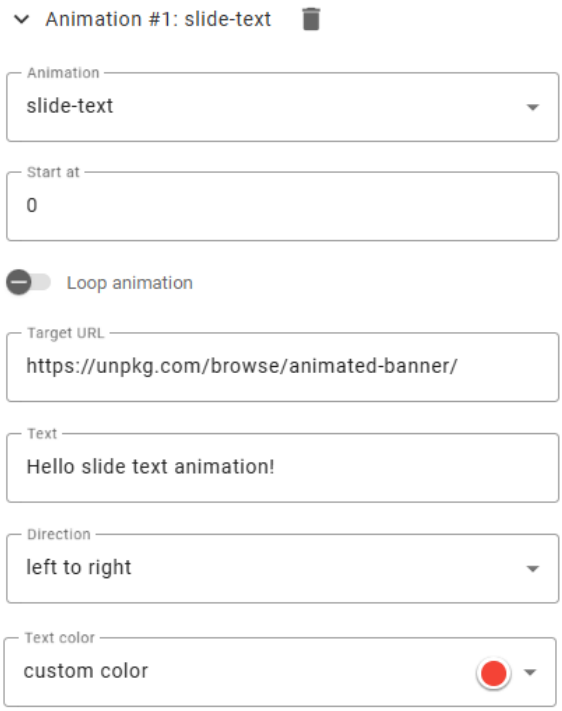
Rysunek 6.3. Proces wyboru tła z pliku lokalnego.
Źródło: opracowanie własne

Po ustawieniu parametrów komponentu, użytkownik przechodzi do kolejnej sekcji dotyczącej animacji. Istnieje możliwość dodania wielu składowych animacji, które będą współtworzyć efekt końcowy. Domyślnym ustawieniem widocznym po uruchomieniu edytora jest jedna składowa animacja o angielskiej nazwie *„slide-text”*, której zadaniem jest przesuwanie zadanego tekstu. Poniżej znajduje się przycisk „Dodaj animację” (ang. *Add animation*). Użytkownik ma możliwość dodania wielu składowych animacji, które będą numerowane kolejno, tak jak pokazano to na rysunku 6.4. Składowe animacje można usuwać za pomocą przycisku zawierającego miniaturę śmietnika.



Rysunek 6.4. Lista animacji składowych w postaci minimalizowanej.
Źródło: opracowanie własne

Po rozwinięciu wybranego elementu listy użytkownik może dostosować efekt do swoich potrzeb. Zestaw wyświetlanych kontrolek uzależniony jest od tego jakie ustawienia oferowane są przez daną animację, a konkretnie od listy parametrów zgłaszanych przez skrypt dedykowanej wtyczki. Mechanizm generowania zawartości tego panelu został opisany w podrozdziale 5.4.



Rysunek 6.5. Przykładowy zestaw kontrolek służących do edycji parametrów animacji.
Źródło: opracowanie własne

Typ animacji ustawiany jest za pomocą kontrolki „Animacja” (ang. *animation*), widocznej na rysunku 6.5 jako pierwsza od góry. Lista obejmuje animacje udostępniane przez wczytane wtyczki. W opisywanym przypadku użytkownik chce wgrać dodatkową wtyczkę oferującą nowy typ animacji. Do tego celu służy przycisk *„Load new animation type”* znajdujący się w lewym panelu edytora. Po wybraniu pliku z dedykowaną wtyczką i poprawnym jej wczytaniu, użytkownikowi zostanie wyświetlony komunikat potwierdzający, że operacja zakończyła się sukcesem, a nowy typ animacji pojawi się na liście.

Następnym krokiem jest dostosowanie parametrów składowych animacji do oczekiwanych efektów. Zakładany scenariusz przewiduje, że użytkownik potrzebuje animowanego banera informującego o aktualnych wyprzedażach, niech będą to na przykład dwie golarki męskie. Dla każdego z produktu dodawane są: przesuwający się tekst oraz obrazek. Na liście animacji składowych dodaje się zcztery animacje, dwie o nazwie *„slide-text”* i dwie o nazwie *„slide-image”*. Przykładowe ustawienia parametrów animacji dla omawianego scenariusza pokazano w tabeli 6.1.

Tabela 6.1. Przykładowe ustawienia parametrów czterech animacji dla omawianego scenariusza. Źródło: opracowanie własne

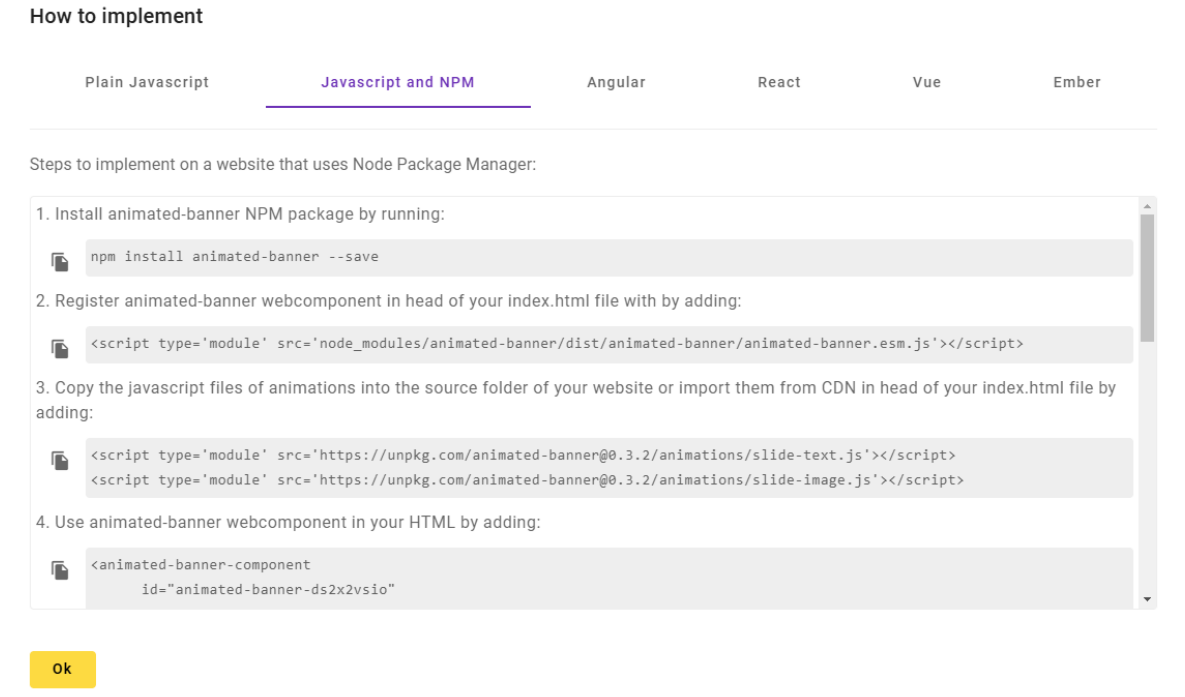
Parametr	**Animacja #1**	**Animacja #2**	**Animacja #3**	**Animacja #4**
animationName	slide-text	slide-image	slide-text	slide-image
text	Phillips OneBlade 164zł!	n/d	Phillips Shaver s5886 549zł!	n/d
foregroundSrc	n/d		n/d	
position	top	middle	top	middle
startAt	1	1	8	8
speed	4	3	4	2
targetUrl	http://www… … philips-oneblade-pro-qp6530-15.bhtml	http://www… … philips-oneblade-pro-qp6530-15.bhtml	http://www… … philips-golarka-philips-s5886-38.bhtml	http://www… … philips-golarka-philips-s5886-38.bhtml
color	yellow	n/d	blue	n/d
fontsize	36	n/d	36	n/d
direction	right to left	left to right	right to left	left to right

Efektem tych ustawień jest animacja przesuwających się na zmianę tekstów i obrazków. Kilka wybranych klatek animacji pokazano na rysunku 6.6. W przypadku uzyskania zamierzonego efektu można przystępować do implementacji animacji na dowolnej stronie internetowej.

Rysunek 6.6. Wybrane klatki z przygotowanej animacji.
Źródło: opracowanie własne

6.2 Osadzenie animacji na dowolnej stronie internetowej

Instrukcja osadzenia animacji znajduje się w edytorze animacji. Po wciśnięciu guzika *„Impelemtation instructions”*, znajdującego się w panelu po lewej stronie ekranu aplikacji, użytkownikowi wyświetlone zostanie okno dialogowe zawierające szczegółowe instrukcje implementacji systemu, pokazane na rysunku 6.7.



Rysunek 6.7. Okno dialogowe zawierające instrukcje implementacji systemu.
Źródło: opracowanie własne

Zgodnie z opisem zamieszczonym w podrozdziale 5.4, edytor wyświetla instrukcje dedykowane dla kilku najpopularniejszych sposobów budowania strony internetowej. W opisywanym scenariuszu strona internetowa nie korzysta z *frameworków* takich jak np. *Angular*, *React* czy *Vue*. Użyto natomiast menadżera pakietów NPM. Użytkownik wybiera więc drugą zakładkę – *„JavaScript and NPM”* i wykonuje wskazane w instrukcji czynności:

- instaluje pakiet niniejszego systemu za pomocą menadżera pakietów NPM;
- importuje definicję komponentu *AnimatedBanner* w nagłówku pliku *index.html*;
- dodaje linki prowadzące do plików wtyczek umieszczonych w sieci dostarczania zawartości (ang. *content delivery network*, CDN) do nagłówka pliku *index.html*, ewentualnie kopiuje pliki i importuje je jako źródła lokalne;
- kopiuje do schowka systemowego część kodu HTML zawierającą kod komponentu *AnimatedBanner* wraz z parametrami animacji, używając do tego przycisku „kopiuj” i wkleja je do pliku HTML na swojej stronie. Uproszczony kod pokazano na listingu 6.1. Warto zwrócić uwagę, że użyte w animacji obrazy przekazane są w formie zakodowanej *base64.*

Listing 6.1. Uproszczony kod strony internetowej użytkownika odgrywającej wytworzoną animacje. Źródło: opracowanie własne

```
<animated-banner-component
  id="animated-banner-ds2x2vsio"
  width="500"
  height="250"
  background="url('…') ">
</animated-banner-component>
<script>
  const animations = [
    {
       animationName: 'slide-text',
       text: "Phillips OneBlade 164zł!",
       …
    },
    {
       animationName: 'slide-image',
      foregroundSrc: "…",
       …
    },
    {
       animationName: 'slide-text',
       "text": "Phillips Shaver s5886 549zł!",
       …
    },
    {
       animationName: 'slide-image',
      foregroundSrc: "…",
       …
    }
  ];
  document.querySelector('#animated-banner-ds2x2vsio').animations = animations;
</script>
```

Po wykonaniu wymienionych czynności, animacja widoczna na stronie internetowej użytkownika wygląda identycznie jak ta, którą stworzył w edytorze animacji.

6.3 Implementacja mechanizmu łatwej zmiany treści

W poprzednim podrozdziale animowane treści widoczne na portalu internetowym użytkownika zostały skopiowane z edytora animacji i wpisane na stałe w kod strony. Scenariusz zakłada, że użytkownik oczekuje, aby teksty i obrazy można było podmieniać w sposób łatwy i bez ingerencji w kod jego aplikacji internetowej. Problem ten można rozwiązać na wiele sposobów. Jednym z nich jest wczytanie przez stronę internetową „scenariusza animacji" z oddzielnego pliku zapisanego w formacie JSON i przekazanie go do komponentu w postaci atrybutu HTML. Opracowany system wspiera takie właśnie rozwiązanie.

Użytkownik może wyeksportować do pliku w formacie JSON cały „scenariusz animacji", zawierający wszystkie ustawione przez niego parametry, za pomocą guzika *„Export animations in JSON"*, znajdującego się w panelu po lewej stronie ekranu edytora.

Kod umieszczony uprzednio na stronie internetowej użytkownika należy zmodyfikować, tak aby parametry animacji nie były zapisane w nim na stałe, ale pobierane z zewnętrznego pliku. Listing 6.2 pokazuje kod strony internetowej użytkownika po przeprowadzonych zmianach. Dane animacji wyeksportowane z edytora animacji do pliku o nazwie *„animated-banner-ds2x2vsio-params.json"* są pobierane przy użyciu metody *fetch* i przypisane do komponentu jako wartość atrybutu *animations*. Przy okazji podobny zabieg zastosowany został również dla atrybutu określającego tło animacji (ang. *background*). Od teraz, wraz ze zmianami treści animacji, użytkownik może także zmienić jej tło.

Listing 6.2. Kod strony internetowej użytkownika po modyfikacjach.
Źródło: opracowanie własne

```
<animated-banner-component
  id="animated-banner-ds2x2vsio"
  width="500"
  height="250"
></animated-banner-component>

<script>
  const component = document.querySelector('#animated-banner-ds2x2vsio');
  let params = {};
  async function getData() {
    const resp = await fetch("./assets/animated-banner-ds2x2vsio-params.json");
    params = await resp.json();
  }

  getData()
    .then(() => {
      component.background = params.background;
      component.animations = params.animations;
  })
</script>
```

Po dokonaniu zmian w kodzie strony, użytkownik może aktualizować animacje na swoim portalu internetowym poprzez zmianę parametrów w edytorze animacji, wyeksportowanie „scenariusza animacji" do pliku w formacie JSON i podmianie danych na serwerze np. za pomocą systemu CMS.

Przy następnym otwarciu strony internetowej użytkownika, znajdujący się tam komponent *AnimatedBanner* wczyta nową wersję pliku z danymi i wyświetli zaktualizowaną animacje.

7. Podsumowanie

Rozdział ten zawiera krótkie omówienie wyników pracy. Opisuje jej osiągnięcia i niedostatki, możliwe do rozwiązania w przyszłości. Rozdział kończy się wnioskami.

7.1 Trudności napotkane podczas realizacji pracy

Podczas tworzenia pracy magisterskiej napotkano na wyzwania, najciekawsze z nich to:

- konieczność komunikacji pomiędzy komponentem *AnimatedBanner,* udostępniającym element HTML *canvas*, a wtyczkami animacji, które na nim rysują. Dobrym rozwiązaniem okazały się zdarzenia użytkownika (ang. *custom events*). Podobne rozwiązanie zastosowano również do komunikacji edytora animacji z wtyczkami. Mechanizmy te opisano w rozdziale piątym.
- brak możliwości wykrywania zdarzeń DOM (ang. *event*) np. kliknięcia w konkretny tekst narysowany już na elemencie *canvas*. Obiekty takie jak kształty czy teksty umieszczone na *canvas* podlegają rasteryzacji i nie stanowią oddzielnych obiektów modelu DOM. W związku z tym nie obsługują zdarzeń. Funkcjonalność jest przydatna w sytuacji, kiedy widocznych jest kilka obiektów, których kliknięcia powinny powodować różne następstwa. Jednym z możliwych sposobów obejścia problemu jest sprawdzanie czy miejsce kliknięcia w element *canvas* pokrywa się z narysowanym już na nim obiektem. Wymaga to jednak skomplikowanego mechanizmu wyliczania i zapamiętywania lokalizacji każdego narysowanego obiektu po stronie wtyczki animacji. Na potrzeby realizowanej pracy problem rozwiązano w ten sposób, że komponent wykrywa kliknięcia myszy i emituje listę odgrywanych w tym momencie składowych animacji wraz z ich parametrami. Jedną z przekazywanych informacji jest, przechowujące hiperłącze, pole „*targetURL”*. Jego wartość ustawiana jest przez użytkownika w edytorze animacji. Dzięki temu mechanizmowi można otworzyć hiperłącze powiązane z daną animacją składową w następstwie kliknięcia na komponent;
- wycieki pamięci podczas działania edytora animacji, powodujące mnożenie się obiektów animatora w pamięci. Aby uzyskać efekt podglądu animacji w czasie rzeczywistym, przy każdej zmianie parametrów scenariusza animacji tworzone są nowe obiekty animatorów. Dotychczasowe, już niepotrzebne obiekty, nie są od razu usuwane przez mechanizm *garbage collector*. Dzieje się tak dlatego, że przechowują one referencje do istniejącego elementu HTML *canvas* i mechanizm prawdopodobnie ocenia, że obiekty mogą być jeszcze w użyciu. Zastosowanym obejściem problemu jest wprowadzenie do edytora animacji mechanizmu usuwania komponentu każdorazowo po zmianach parametrów i następnie generowania go na nowo, wraz z nowym elementem *canvas*. Problem nie występuje w przypadku użycia komponentu odgrywającego animacje na docelowej stronie internetowej.

7.2 Mocne i słabe strony zaproponowanego rozwiązania

Opracowany w ramach pracy magisterskiej „Elastyczny system tworzenia animacji dla portali internetowych” ma liczne zalety, najważniejsze z nich to:

- proste osadzenie komponentu odgrywającego animacje na dowolnej stronie internetowej, dzięki udostępnieniu go w postaci *Web Componentu* opublikowanego w ogólnodostępnym repozytorium pakietów *Node* (ang. *Node Package Manager Registry*) oraz sieci dostarczania zawartości (ang. *content delivery network*, CDN);
- możliwość zaprojektowania animacji za pomocą intuicyjnego w obsłudze edytora animacji i zapisania go w postaci „scenariusza animacji’ gotowego do użycia na stronie internetowej zawierającej komponent odgrywający animacje. Umożliwia to umieszczenie animacji na stronach

internetowych osobom nie posiadającym wiedzy z zakresu programowania oraz późniejsze zmiany animacji bez ingerencji w kod portalu internetowego;

- system dedykowanych wtyczek animacji zapewnia elastyczność rozwiązania i umożliwia rozszerzenie go o dodatkowe efekty;
- przekazywanie parametrów animacji do komponentu w postaci atrybutów HTML umożliwia szybką podmianę treści animacji z poziomu kodu portalu internetowego na którym osadzony jest komponent. Przy zastosowaniu dodatkowych mechanizmów możliwe jest zróżnicowane treści dla poszczególnych odbiorców lub w zależności od różnych sytuacji. Na przykład możliwe jest wyświetlenie różnych treści tekstu w zależności od wybranej wersji językowej strony, bez zmiany pozostałych elementów takich jak obrazy lub zaplanowany przebieg animacji;
- animacje generowane przy użyciu *Canvas API* „ważą" dużo mniej niż pliki animacji w postaci bitmapowej, a wprowadzanie drobnych zmian jak na przykład zmiany treści tekstu wymaga mniej pracy. Oferują większy wachlarz możliwości graficznych niż animowanie elementów DOM, w teorii są też bardziej wydajne.

Podczas realizacji prototypu dostrzeżono też niedostatki rozwiązania:

- implementacja animacji przy użyciu *Canvas API* jest trudniejsza w porównaniu do innych technologii, ponadto w przedstawionym rozwiązaniu dla uzyskania nowych efektów istnieje potrzeba tworzenia dedykowanych wtyczek zgodnych z system co jest czasochłonne. Implementacja niektórych funkcjonalności wymaga pracy zarówno po stronie komponentu jak i po stronie wtyczek animujących;
- wczytywanie zdalnych skryptów *JavaScript*, takich jak dedykowane wtyczki animacji, pobranych z nieznanego źródła może stanowić zagrożenie i często jest sprzeczne z polityką bezpieczeństwa.

7.3 Możliwe kierunki rozwoju prototypu

W wyniku analizy napotkanych trudności oraz dostrzeżonych niedostatków wytworzonego prototypu formułuje się następujące możliwe kierunki jego rozwoju:

- rozwój funkcjonalności systemu poprzez dodanie większej ilości gotowych wtyczek animacji;
- zbadanie obszaru animacji wykorzystujących interfejs *WebGL* i kontekst 3D;
- rozwój funkcjonalności edytora animacji o panel wyświetlający oś czasu scenariusza animacji (ang. timeline), ułatwiający użytkownikowi orientację co do kolejności i czasu odgrywania składowych animacji;
- dalsze uproszczenia obsługi systemu w zakresie ułatwień zmiany animowanych treści na stronie internetowej poprzez możliwość przesłania przygotowanego „scenariusza animacji" za pomocą żądania typu POST bezpośrednio z edytora animacji pod wskazany URL (np. na przygotowany już serwer) lub bezpośrednie połączenie komponentu odgrywającego animację do wskazanego źródła URL, skąd komponent mógłby pobrać „scenariusz animacji";
- poszerzenie funkcjonalności istniejących wtyczek o mechanizm wyświetlający klatki przy użyciu metody *requestAnimationFrame* ale obliczający czas trwania animacji w oparciu o stały punkt na osi czasu [49], zwracanego przykładowo przez metodę *performance.now* [15];
- rozwiązanie problemu umieszczania na elemencie HTML *canvas* animowanego pliku rastrowego w formacie GIF, APNG lub WebP w celu osiągnięcia efektu „animacji w animacji" np. wjazdu samochodu z poruszającymi się kołami lub wejścia postaci. Na elemencie *canvas* rysowana jest jedynie pierwsza klatka takiego animowanego pliku. Obejściem problemu może być rysowanie po kolei poszczególnych klatek w kolejnych iteracjach pętli animacji, wymaga to jednak dodatkowych nakładów pracy, w tym ekstrakcji poszczególnych klatek z pliku.

7.4 Zakończenie

W ramach pracy wykonano analizę istniejących na rynku technik umieszczania animacji na portalach internetowych. W oparciu o wyniki analizy, przedstawiono koncepcję „Elastycznego systemu tworzenia animacji dla portali internetowych" wprowadzającą usprawnienia w stosunku do istniejących rozwiązań. Poprzez implementację prototypu rozwiązania podjęto próbę ułatwienia procesu umieszczania i aktualizacji animacji na stronach internetowych dla osób, które nie posiadają wystarczających kompetencji technicznych z obszaru programowania.

Opisano dostrzeżone wady i zalety rozwiązania oraz trudności napotkane podczas tworzenia prototypu. Należy pamiętać, że prototyp, który powstał w ramach pracy nie jest jeszcze system w pełni gotowym do komercyjnego użycia, stanowi bazę do dalszego rozwoju tej technologii. Podział systemu na trzy elementy spełnił swoje założenia. Edytor animacji, umożliwia wytworzenie i modyfikację animacji w sposób łatwy i efektywny. Zapisywane za jego pomocą „scenariusze animacji" pozwalają na odtwarzanie zaprojektowanych efektów bez potrzeby ingerencji w kod strony internetowej. Koncepcja dedykowanych wtyczek animacji zapewnia generyczność systemu i umożliwia rozszerzenie go o kolejne efekty w przyszłości. Udostępnienie komponentu odgrywającego animacje jako *Web Componentu* skutecznie uprościło sposób osadzenia go na dowolnej stronie internetowej.

Dzięki takiemu rozwiązaniu, nawet osoby bez specjalistycznej wiedzy mogą efektywnie zarządzać animowanymi treściami w swoich aplikacjach internetowych.

Bibliografia

[1]. J. Hawley, „Website Animation: All There is to know in 2024”, https://mightyfinedesign.co/website-animation-all-there-is-to-know/#types-of-website-animation, dostęp: 21 maja 2024 r.

[2]. I. Paul, „So long, Flash! YouTube now defaults to HTML5 on the web”, https://www.pcworld.com/article/431524/so-long-flash-youtube-now-defaults-to-html5-on-the-web.html, dostęp: 21 maja 2024 r.

[3]. Oficjalna strona internetowa *Adobe* – producenta rozwiązania *FlashPlayer*, https://www.adobe.com/products/flashplayer/enterprise-end-of-life.html, dostęp: 21 maja 2024 r.

[4]. „Ifarme vs. Javascipt Tags”, https://www.knowonlineadvertising.com/facts-about-online-advertising/sharing-knowledge/popular-third-party-tags/iframe-vs-jscript-tags/#google_vignette, dostęp: 21 maja 2024 r.

[5]. „Iframe”, https://support.google.com/adsense/answer/32769?hl=en, dostęp: 21 maja 2024 r.

[6]. Mozilla Developer Network Web Docs, „<img>: The Image Embed element”, https://developer.mozilla.org/en-US/docs/Web/HTML/Element/img, dostęp: 22 maja 2024 r.

[7]. Mozilla Developer Network Web Docs, „<video>: The Video Embed element”, https://developer.mozilla.org/en-US/docs/Web/HTML/Element/video, dostęp: 22 maja 2024 r.

[8]. Mozilla Developer Network Web Docs, „svg”, https://developer.mozilla.org/en-US/docs/Web/SVG/Element/svg, dostęp: 22 maja 2024 r.

[9]. Mozilla Developer Network Web Docs, „<animate>”, https://developer.mozilla.org/en-US/docs/Web/SVG/Element/animate, dostęp: 22 maja 2024 r.

[10]. Mozilla Developer Network Web Docs, „<animateMotion>”, https://developer.mozilla.org/en-US/docs/Web/SVG/Element/animateMotion, dostęp: 22 maja 2024 r.

[11]. Mozilla Developer Network Web Docs, „<animateTransform>”, https://developer.mozilla.org/en-US/docs/Web/SVG/Element/animateTransform, dostęp: 22 maja 2024 r.

[12]. W3Schools, „CSS Animations”, https://www.w3schools.com/css/css3_animations.asp, dostęp: 23 maja 2024 r.

[13]. Mozilla Developer Network Web Docs, „Using CSS animations”, https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_animations/Using_CSS_animations, dostęp: 23 maja 2024 r.

[14]. Oficjalna strona internetowa biblioteki *Animista*, https://animista.net/, dostęp: 23 maja 2024 r.

[15]. Mozilla Developer Network Web Docs, „Performance: now() method,” https://developer.mozilla.org/en-US/docs/Web/API/Performance/now, dostęp: 24 maja 2024 r.

[16]. W3Schools, „How TO – JavaScript HTML Animations”, https://www.w3schools.com/howto/howto_js_animate.asp, dostęp: 24 maja 2024 r.

[17]. „jQuery Usage Statistics”, https://trends.builtwith.com/javascript/jQuery, dostęp: 25 maja 2024 r.

[18]. Dokumentacja biblioteki *jQuery*, https://jquery.com/license/, dostęp: 25 maja 2024 r.

[19]. Dokumentacja biblioteki *jQuery*, https://api.jquery.com/animate/, dostęp: 25 maja 2024 r.

[20]. Kod źródłowy biblioteki *jQuery*, https://github.com/jquery/jquery/blob/main/src/effects.js, dostęp: 25 maja 2024 r.

[21]. „GSAP”, https://webtechsurvey.com/technology/gsap, dostęp: 26 maja 2024 r.

[22]. Coding Crafts, „10 Best Website Animation Libraries for Stunning Designs”, https://www.linkedin.com/pulse/10-best-website-animation-libraries-stunning-designs-qia6f/, dostęp: 26 maja 2024 r.

[23]. Oficjalna strona internetowa biblioteki GSAP, https://gsap.com/, dostęp: 26 maja 2024 r.

[24]. Oficjalna strona internetowa biblioteki GSAP, https://gsap.com/resources/get-started#so-what-properties-can-i-animate, dostęp: 26 maja 2024 r.

[25]. Dokumentacja biblioteki GSAP, https://gsap.com/docs/v3/GSAP/CorePlugins/CSS/#non-animatable-properties, dostęp: 26 maja 2024 r.

[26]. Dokumentacja biblioteki GSAP, https://gsap.com/docs/v3/GSAP/Tween, dostęp: 26 maja 2024 r.

[27]. Oficjalna strona internetowa biblioteki GSAP, https://gsap.com/resources/get-started#any-numeric-value-color-or-complex-string-containing-numbers, dostęp: 26 maja 2024 r.

[28]. Mozilla Developer Network Web Docs, „Canvas API", https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API, dostęp: 27 maja 2024 r.

[29]. Mozilla Developer Network Web Docs, „WebGL: 2D and 3D graphics for the web", https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API, dostęp: 27 maja 2024 r.

[30]. Mozilla Developer Network Web Docs, „CanvasRenderingContext2D", https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D, dostęp: 28 maja 2024 r.

[31]. Mozilla Developer Network Web Docs, „CanvasRenderingContext2D:drawImage() method", https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/drawImage, dostęp: 28 maja 2024 r.

[32]. Mozilla Developer Network Web Docs, „CanvasRenderingContext2D: rotate() method", https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/rotate, dostęp: 28 maja 2024 r.

[33]. Kod źródłowy biblioteki EaselJS, https://github.com/CreateJS/EaselJS/blob/master/LICENSE.txt, dostęp: 29 maja 2024 r.

[34]. Dokumentacja biblioteki EaselJS, https://createjs.com/docs/easeljs/modules/EaselJS.html, dostęp: 29 maja 2024 r.

[35]. Dokumentacja biblioteki EaselJS, https://createjs.com/docs/tweenjs/classes/RotationPlugin.html, dostęp: 29 maja 2024 r.

[36]. Oficjalna specyfikacja HTML, https://html.spec.whatwg.org/#a-quick-introduction-to-html, dostęp: 8 maja 2024 r.

[37]. W3Schools, „HTML Introduction", https://www.w3schools.com/html/html_intro.asp, dostęp: 8 maja 2024 r.

[38]. Oficjalna specyfikacja HTML, https://html.spec.whatwg.org/#the-html-element, dostęp: 8 maja 2024 r.

[39]. Oficjalna specyfikacja HTML, https://html.spec.whatwg.org/#the-head-element, dostęp: 8 maja 2024 r.

[40]. Oficjalna specyfikacja HTML, https://html.spec.whatwg.org/#the-script-element, dostęp: 8 maja 2024 r.

[41]. Oficjalna specyfikacja HTML, https://html.spec.whatwg.org/#element-definitions, dostęp: 8 maja 2024 r.

[42]. Mozilla Developer Network Web Docs, „JavaScript", https://developer.mozilla.org/en-US/docs/Web/JavaScript, dostęp: 8 maja 2024 r.

[43]. Oficjalna strona internetowa języka TypeScript, https://www.typescriptlang.org/why-create-typescript/, dostęp: 8 maja 2024 r.

[44]. Oficjalna dokumentacja języka TypeScript, https://www.typescriptlang.org/docs/handbook/decorators.html, dostęp: 8 maja 2024 r.

[45]. Mozilla Developer Network Web Docs, „Web Components", https://developer.mozilla.org/en-US/docs/Web/API/Web_components, dostęp: 9 maja 2024 r.

[46]. „Introduction", https://www.webcomponents.org/introduction, dostęp: 9 maja 2024 r.

[47]. Mozilla Developer Network Web Docs, „Using custom elements", https://developer.mozilla.org/en-US/docs/Web/API/Web_components/Using_custom_elements, dostęp: 9 maja 2024 r.

[48]. Mozilla Developer Network Web Docs, „Using shadow DOM", https://developer.mozilla.org/en-US/docs/Web/API/Web_components/Using_shadow_DOM, dostęp: 9 maja 2024 r.

[49]. Mozilla Developer Network Web Docs, „Window: requestAnimationFrame() method", https://developer.mozilla.org/en-US/docs/Web/API/window/requestAnimationFrame, dostęp: 9 maja 2024 r.

[50]. L. Remi, „Animating with javascript: from setInterval to requestAnimationFrame", https://hacks.mozilla.org/2011/08/animating-with-javascript-from-setinterval-to-requestanimationframe/, dostęp: 9 maja 2024 r.

[51]. Mozilla Developer Network Web Docs, „Window: cancelAnimationFrame() method”, https://developer.mozilla.org/en-US/docs/Web/API/Window/cancelAnimationFrame, dostęp: 9 maja 2024 r.

[52]. Dokumentacja *frameworka Stencil.js*, https://stenciljs.com/docs/overview, dostęp: 9 maja 2024 r.

[53]. Dokumentacja *frameworka Stencil.js*, https://stenciljs.com/docs/faq, dostęp: 9 maja 2024 r.

[54]. Dokumentacja *frameworka Stencil.js*, https://stenciljs.com/docs/component-lifecycle, dostęp: 9 maja 2024 r.

[55]. Oficjalna strona internetowa rozwiązania *Node.JS*, https://nodejs.org/en, dostęp: 10 maja 2024 r.

[56]. Dokumentacja narzędzia NPM, https://docs.npmjs.com/about-npm, dostęp: 10 maja 2024 r.

[57]. Oficjalna strona internetowa *frameworka Angular*, https://angular.io/guide/what-is-angular, dostęp: 10 maja 2024 r.

[58]. Repozytorium kodu źródłowego *frameworka Angular*, https://github.com/angular/components?tab=readme-ov-file#readme, dostęp: 10 maja 2024 r.

[59]. N. Arunodi, „Top 10 Angular Component Libraries”, https://www.syncfusion.com/blogs/post/top-angular-component-libraries, dostęp: 10 maja 2024 r.

[60]. Informacja na temat narzędzia *ngx-colors*, https://www.npmjs.com/package/ngx-colors, dostęp: 10 maja 2024 r.

[61]. Repozytorium kodu źródłowego narzędzia *ngx-colors*, https://github.com/KroneCorylus/ngx-colors?tab=MIT-1-ov-file#readme, dostęp: 10 maja 2024 r.

[62]. Dokumentacja techniczna rozwiązania *Github*, https://docs.github.com/en/get-started/using-git/about-git, dostęp: 11 maja 2024 r.

[63]. Oficjalna strona internetowa technologii *Git*, https://git-scm.com/about/free-and-open-source, dostęp: 11 maja 2024 r.

[64]. „Top IDE index”, https://pypl.github.io/IDE.html, dostęp: 11 maja 2024 r.

[65]. P. Krugiej, „IntelliJ IDEA w akcji – Do czego służy IntelliJ IDEA”
https://kursjava.com/intellij-idea-w-akcji/do-czego-sluzy-intellij-idea/,
dostęp: 11 maja 2024 r.

[66]. Oficjalna strona internetowa firmy *JetBrains* – wydawcy IDE *Webstorm*,
https://www.jetbrains.com/community/education/#students,
dostęp: 11 maja 2024 r.

[67]. Dokumentacja *frameworka Stencil.js*,
https://stenciljs.com/docs/component,
dostęp: 1 maja 2024 r.

[68]. Dokumentacja *frameworka Stencil.js*,
https://stenciljs.com/docs/host-element,
dostęp: 1 maja 2024 r.

[69]. Dokumentacja *frameworka Stencil.js*,
https://stenciljs.com/docs/properties,
dostęp: 1 maja 2024 r.

[70]. Dokumentacja *frameworka Stencil.js*,
https://stenciljs.com/docs/events,
dostęp: 1 maja 2024 r.

[71]. R. Carniato, „A Look at Compilation in JavaScript Frameworks”,
https://dev.to/this-is-learning/a-look-at-compilation-in-javascript-frameworks-3caj,
dostęp: 1 maja 2024 r.

[72]. Dokumentacja *frameworka Stencil.js*,
https://stenciljs.com/docs/custom-elements,
dostęp: 1 maja 2024 r.

[73]. Dokumentacja *frameworka Stencil.js*,
https://stenciljs.com/docs/angular,
dostęp: 1 maja 2024 r.

[74]. Dokumentacja *frameworka Angular*,
https://angular.io/guide/architecture,
dostęp: 4 maja 2024 r.

[75]. Dokumentacja *frameworka Angular*,
https://angular.io/guide/file-structure,
dostęp: 4 maja 2024 r.

Dodatki

Dodatek A: Spis listingów

Dodatek B: Spis rysunków

Dodatek C: Spis tabel