Možnosti využití animovaných SVG ve webovém rozhraní

Robin Dostál

Bakalářská práce
2024

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Robin Dostál**
Osobní číslo:	**A21323**
Studijní program:	**B0613A140020 Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Možnosti využití animovaných SVG ve webovém rozhraní**
Téma práce anglicky:	**Possibilities of Using Animated SVGs in the Web Interface**

Zásady pro vypracování

1. Představte grafický formát SVG v kontextu použití ve webových dokumentech, základní principy, možnosti syntaxe, jeho vlastnosti a výhody.
2. Stručně představte jazyk JavaScript a základní možnosti interakce s SVG elementy.
3. Vyhledejte a představte javaskriptové knihovny pro pokročilejší práci s SVG grafikou na webu.
4. Pomocí vybraných knihoven vytvořte ukázkové příklady jejich využití a uveďte porovnání jejich možností a způsobu použití.
5. Vyberte a otestujte programy pro tvorbu SVG animací, vypracujte ukázkové příklady a uveďte porovnání programů.

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. GLITSCHKA, Von. *Vektory: základní výcvik*. Brno: Computer Press, 2013. ISBN 9788025141298.
2. PEHLIVANIAN, Ara a NGUYEN, Don. *JavaScript okamžitě*. Brno: Computer Press, 2014. ISBN 9788025141632.
3. ODELL, Den. *JavaScript: průvodce programováním ajaxových aplikací*. Brno: Computer Press, 2010. ISBN 9788025127339.
4. LARSEN, Rob. *Mastering SVG*. Velká Británie: Packt Publishing, 2018. ISBN 9781788621984.
5. *Expressive Animator* [online]. [cit. 2023-11-10]. Dostupné z: https://expressive.app/expressive-animator/
6. *SVGator* [online]. [cit. 2023-11-10]. Dostupné z: https://www.svgator.com/

Vedoucí bakalářské práce: **Ing. Radek Vala, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce: **5. listopadu 2023**
Termín odevzdání bakalářské práce: **13. května 2024**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN
L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 5. ledna 2024

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že při tvorbě této bakalářské práce jsem použil nástroj generativního modelu AI ChatGPT verze 3.5 dostupný na adrese https://chat.openai.com, za účelem úpravy a kontroly textu. Model jsem také použil pro optimalizaci kódu v praktické části. Po použití tohoto nástroje jsem provedl kontrolu obsahu a přebírám za něj plnou zodpovědnost.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne

Robin Dostál, v. r.
podpis studenta

ABSTRAKT

Bakalářská práce se zaměřuje na možnosti využití animovaných SVG ve webovém rozhraní. Cílem práce bylo blíže seznámit čtenáře s možnostmi implementace animovaných SVG do webových stránek a aplikací.

Teoretická část zahrnuje popis formátu SVG, jeho vlastnosti a výhody oproti jiným formátům. Stručně je také představen jazyk JavaScript, který v kombinaci s SVG tvoří primární obsah praktické části. Na závěr jsou představeny JavaScriptové knihovny a programy určené pro animování a manipulaci s SVG.

Praktická část pak obsahuje implementaci zmíněných knihoven z teoretické části. Autor práce zde popisuje postupy, celkově nabyté dojmy a zkušenosti z jednotlivých knihoven a programů.

Klíčová slova: SVG, JavaScript, animace, webový design

ABSTRACT

The thesis focuses on the possibilities of using animated SVGs in web interfaces. The aim of the thesis was to familiarize the reader with the possibilities of implementing animated SVGs in web pages and applications.

The theoretical part includes a description of the SVG format, its features and advantages over other formats. The JavaScript language is also briefly introduced, which in combination with SVG forms the primary content of the practical part. Finally, JavaScript libraries and programs for animating and manipulating SVG are introduced.

The practical part contains the implementation of the libraries mentioned in the theoretical part. The author of the thesis describes here the procedures, overall impressions and experiences gained from each library and program.

Keywords: SVG, JavaScript, animation, web design

Tímto bych chtěl poděkovat vedoucímu bakalářské práce Ing. Radku Valovi, Ph.D., za prvotní myšlenku pro tuto práci, dále za jeho ochotu, cenné rady a konzultace. Také bych chtěl poděkovat mojí rodině, přítelkyni a kamarádům, za podporu, trpělivost a možnost odreagování při tvorbě této bakalářské práce.

OBSAH

ÚVOD

V dnešní době tvoří webové aplikace a internetové stránky nedílnou součást našich každodenních životů. Jejich vizuální prezentace nabývá tak čím dál větší důležitosti. V rámci této prezentace hraje jednu z klíčových rolí animace, díky které jsou stránky nejen vizuálně atraktivní, ale také svým způsobem zlepšuje uživatelskou zkušenost a interaktivitu. Za pomoci animace je tak možné komunikovat s uživatelem, sdělovat mu potřebné informace a při tom všem vytvářet vizuálně poutavé prostředí.

V této bakalářské práci se zaměříme konkrétně na možnost animování souborů SVG (Scalable Vector Graphics) na webových stránkách primárně v kombinaci s jazykem JavaScript. Formát SVG díky svému vektorovému charakteru umožňuje snadné bezeztrátové škálování což je v dnešní době, kdy se používají zobrazovací média s různými velikostmi obrazovek, výhodou. Dále má SVG oproti rastrovým formátům, jakou jsou GIF, APNG nebo třeba WebP, tu výhodu, že jej lze propojit s právě zmíněným JavaScriptem k vytváření interaktivních prvků, které reagují například na akce a události.

Na trhu se mimo jiné také pohybují softwarové programy, určené pro tvorbu vektorových animací, které nabízejí širokou škálu funkcí pro tvorbu složitějších animací s možností využití grafického uživatelského rozhraní.

Cílem této práce je poskytnout přehled o možnostech animování SVG na webových stránkách, porovnat různé přístupy a nástroje a analyzovat jejich praktické využití. To má přispět k lepšímu porozumění a efektivnímu využití animací SVG ve vývoji webových aplikací a stránek.

I. TEORETICKÁ ČÁST

1 FORMÁT SVG

SVG (Scalable Vector Graphics), neboli škálovatelná vektorová grafika, je značkovací jazyk založen na jazyku XML určený k definování obrázků. SVG se běžně využívá u tištěné publikace nebo u technických výkresů. K jeho plnému využití však slouží moderní webové prohlížeče. SVG je velmi flexibilní a existuje mnoho způsobů, jak jej v kombinaci s HTML, CSS a JavaScriptem implementovat do webu. [1]; [2]

```
 1. <?xml version="1.0" encoding="UTF-8"?>
 2. <svg id="SVG" data-name="SVG" xmlns="http://www.w3.org/2000/svg"
        viewBox="0 0 300 300">
 3.   <defs>
 4.     <style>
 5.       .cls-1 {
 6.         fill: #f7b30b;
 7.         stroke-width: 0px;
 8.       }
 9.     </style>
10.   </defs>
11.   <g id="svg-path" data-name="svg-path">
12.     <path class="cls-1" d="M10 80 Q 95 10 180 80">
13.   </g>
14. </svg>
```

Zdrojový kód 1.1 Ukázka kódu SVG vyexportovaného z programu Adobe Illustrator

Jelikož je SVG ve své podstatě kód, nabízí se možnost vytvářet tyto objekty v jakémkoliv textovém editoru. Tento přístup je však vhodný pro základní tvary, jako jsou kruhy, čtverce nebo kombinace různých tahů. Pro tvorbu více komplexnějších obrázků se používají grafické editory, které umožňují práci s vektorovou grafikou. Mezi tyto programy patří například Adobe Illustrator nebo Inkscape. Tyto programy poskytují širokou škálu různých nástrojů a funkcí, pomocí kterých lze vytvářet složitější cesty vektorů, maskovat jednotlivé vrstvy či využívat barevných přechodů, takzvaných gradientů. [3]

1.1 Vlastnosti

SVG objekty jsou tvořeny z vektorů. Díky tomu tyto objekty můžeme zvětšovat a zmenšovat dle libosti bez ztráty kvality. Na rozdíl od rastrové grafiky (PNG, JPEG), která je ideální pro zobrazování fotografií na webových stránkách, je SVG vhodnější spíše pro graficky nenáročné prvky. Většinou se s tím můžeme setkat v podobě log, ale u dnešních moderních webových designů se hojně využívají také SVG ikony, které lze pomocí kaskádových stylů CSS upravovat. Například je možné změnit barvu ikon tak, aby ladily s obsahem stránky.

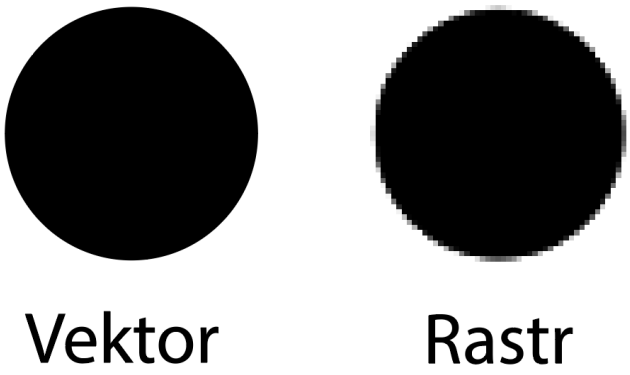
Další, poměrně užitečnou vlastností jsou animace. U samotného SVG objektu jsme tak schopni dynamicky měnit vlastnosti a atributy jako je poloha, barva, velikost, a mnoho dalšího. Tyto animace mohou být specifikovány pomocí CSS nebo JavaScriptu a umožňují interaktivní a pro uživatele atraktivní prezentaci obsahu na stránce. [4]; [5]

1.2 Výhody oproti jiným formátům

Formát SVG oproti jiným formátům disponuje několika výhodami. Následující část popisuje několik těchto výhod.

1.2.1 Vektorová grafika

Jak již samotný název Scalable Vector Graphics vypovídá, tento formát využívá vektorovou grafiku. Oproti klasickým rastrovým formátům, jako jsou například PNG, JPEG nebo GIF, je poněkud odlišná. Například při zmenšování nebo roztahování rastrové grafiky dochází ke zhoršení kvality zobrazení. To se u SVG stát nemůže, jelikož vektorová grafika vychází z matematiky a výsledný objekt se vždy po transformaci přepočítá a zobrazí ve stejné kvalitě.



Obrázek 1.1 Porovnání vektorové a rastrové grafiky

Toho lze v dnešní době využít například u responzivity webových stránek, jelikož se vyžaduje, aby byl obsah, který je na velkých zařízeních, jako jsou monitory, stejně přístupný i na menších mobilních zařízeních a tím pádem se jevil menší nebo větší dle potřeby. Ano, můžeme sice použít rastrový obrázek s větším rozlišením, a tím pádem rozdíly při změně velikosti budou nepatrné, ale tím se dostáváme k další z výhod SVG a tím je jeho velikost. [5]

1.2.2 Velikost

U většiny případů je velikost SVG souborů menší než například u PNG. Mohou ale nastat i situace u komplexnějších obrázků nebo souborů kdy výsledné PNG bude několikanásobně menší než SVG. Zde závisí na volbě uživatele, zda chce dosáhnout co nejmenší velikosti za cenu toho, že další transformování a manipulace objektu povedou, v případě použití rastrového formátu, k jeho deformaci. [3]; [6]

1.2.3 Podpora prohlížečů

SVG však není jediný formát pro ukládání vektorové grafiky. Existují například formáty AI, EPS nebo třeba CDR. Ty to formáty však nejsou webovými prohlížeči běžně podporovány. Jelikož je SVG psáno v kódu XML, dokážou tyto soubory uchovat textové informace jako doslovný text. Tento fakt umožňuje vyhledávačům číst data z SVG pro jejich klíčová slova, což může přispět k lepšímu zařazení webové stránky ve výsledcích vyhledávaní. [7]

1.3 Implementace SVG na webu

Existuje mnoho způsobů, jakými lze SVG implementovat do webu, ať už pomocí HTML nebo CSS. Ne všechny metody jsou však optimální, jelikož SVG obrázky mohou ztratit některé vlastnosti a výhody, které byly popsány v předchozí kapitole. Následující část popisuje několik způsobů, kterými lze SVG implementovat.

1.3.1 Implementace pomocí *<img>*

Jedním ze základních způsobů, jak SVG vložit do webové stránky, je použít stejný způsob, jakým třeba na web vkládáme obrázky typu PNG nebo JPEG, a to pomocí tagu <*img*>. Do atributu *„src“* v tagu <*img*> uvedeme cestu k SVG souboru. Předpokládá se tedy, že se SVG soubor nachází přímo v našem projektu. Pokud není specifikována velikost, jak se má obrázek zobrazit, jeho velikost se odvíjí od toho, jaké jsou uvedeny hodnoty přímo v souboru SVG. Pokud chceme takto vloženému obrázku nastavit velikost, můžeme tak docílit pomocí CSS. [1]

```
1. <img src="logo.svg" alt="logo" height="100“ width="100"/>
```

Zdrojový kód 1.2 Implementace SVG pomocí <*img*>

1.3.2 Implementace pomocí CSS *background-image*

Vložení pomocí *background-image* je podobné jako u předešlého tagu <*img*> v HTML, avšak tento způsob implementace se provádí přímo v CSS. I když má podobná omezení, nabízí o něco více možností úprav díky flexibilitě CSS stylů. [8]

```
1. .logo {
2.     width: 100px;
3.     height: 100px;
4.     background-image: url('logo.svg');
5.     background-size: cover
6.     background-position: center;
7.     background-repeat: no-repeat;
8.  }
```

Zdrojový kód 1.3 Implementace SVG pomocí CSS background-image

1.3.3 Implementace pomocí *inline <svg>*

Jednou z dalších možností, jak vložit SVG přímo do kódu HTML, je pomocí tagu <*svg*>. U této implementace je XML kód z SVG vložen přímo do HTML kódu stránky. Tento přístup zkracuje dobu načítání stránky, protože není potřeba dalších HTTP požadavků. Další výhodou je, že se nám otvírá více možností, jakými lze SVG upravit pomocí CSS nebo JavaScriptu. Nevýhoda však může nastat u komplexních obrázků kdy je XML kód příliš rozsáhlý, tudíž bude zabírat mnoho prostoru v dokumentu. [8]

```
1. <body>
2.  <svg id="SVG" data-name="SVG" viewBox="0 0 100 100">
3.    <g id="logo" data-name="logo">
4.      <path d="M10 80 Q 95 10 180 80">
5.    </g>
6.  </svg>
7. </body>
```

Zdrojový kód 1.4 Implementace SVG inline

1.3.4 Implementace pomocí *<object>*

Implementace pomocí tagu <*object*> je podobná jako u tagu <*img*>. Místo atributu *„src“* je zde přítomen atribut *„data“*, do kterého se podobně jako u již zmíněného <*img*> zadává cesta k SVG souboru. [8]

```
1. <object data="logo.svg" width="100" height="100"></object>
```

Zdrojový kód 1.5 Implementace SVG pomocí <*object*>

1.3.5 Implementace pomocí *<iframe>*

Využívání tohoto způsobu implementace není příliš doporučováno, každopádně jeho vložení je podobné předchozímu tagu <*object*>. Tento přístup není doporučován zejména kvůli problematičtějšímu měnění velikosti u takto vloženého SVG. Pokud chceme, aby se SVG zobrazilo kompletní bez prázdných míst, je potřeba nastavit <*iframe*> přesné rozměry jaké má mít zobrazované SVG. [9]

```
1. <iframe src="logo.svg" width="100px" height="100px"></iframe>
```

Zdrojový kód 1.6 Implementace SVG pomocí <*iframe*>

1.3.6 Implementace pomocí *<embed>*

Opět se jedná o obdobné použití jako u tagu <*object*>. Tato implementace však také není doporučována, jelikož některé internetové prohlížeče nemusí tento způsob implementace podporovat. [3]; [8]

```
1. <embed data="logo.svg" width="100" height="100" />
```

Zdrojový kód 1.7 Implementace SVG pomocí <*embed*>

1.4 Knihovny ikon

Jedním z velmi aktivně využívaných prvků, se kterým se můžeme na webových stránkách setkat, jsou ikony. Ty pro uživatele představují vizuálně výstižný způsob, jak rychle a jednoduše identifikovat specifické funkce, akce a obsah na webových stránkách. Takových funkcí může stránka obsahovat desítky, a z toho důvodu je zapotřebí také desítky ikon. Ty by měly být jednotně stylizovány, aby celkový vzhled webu nabýval celistvosti. K tomuto účelu se využívají takzvané knihovny ikon.

Knihovny většinou obsahují ikony ve formě SVG. Při využívání většího množství ikon na webu máme možnost jednoduše tyto knihovny importovat pomocí odkazů v HTML uvnitř tagu <*head*>, což eliminuje potřebu stahování či instalace. Jako příklad můžeme uvést knihovnu Font Awesome, která nabízí širokou škálu ikon, z nichž část je zcela zdarma a dostupná k volnému použití. [10]

1.4.1 Přidávání ikon do HTML

Například knihovna Font Awesome je navržena pro použití s inline elementy HTML. Lze ji jednoduše do projektu zahrnou pomocí odkazu v hlavičce projektu. Je doporučené dodržovat konzistentnost v rámci webových stránek. Doporučuje se využívat tagu <*i*>, který se běžně využívá pro nastavení stylu písma na kurzívu. V tomto případě slouží jako kontejner pro zobrazení ikony. Pomocí atributu *„class“* je možné určit, jaká ikona se má zobrazit. Do uvozovek stačí uvést prefix *„fa-“* a název ikony, tak jak je specifikován v knihovně. Alternativně lze místo elementu <*i*> použít také element <*span*>. [11]

```
1. <i class="fa-solid fa-user"></i>
```

Zdrojový kód 1.8 Implementace ikony z knihovny Font Awesome

2 JAVASCRIPT

JavaScript je skriptovací jazyk, využívaný pro vytváření dynamického obsahu a interaktivních prvků na webových stránkách. Kromě toho, že přidává možnost interagovat s obsahem webové stránky, se také před načítá se spoustou různých funkcí. JavaScript lze do webu začlenit dvěma způsoby:

- Vnitřní JS - pomocí tagu <*script*> lze JS začlenit přímo v HTML kódu.
- Vnější JS – kód JavaScriptu je uložen v odděleném souboru, který je následně přes odkaz předán do HTML. [12]; [13]

2.1 Vlastnosti JavaScriptu

- JavaScript je vestavěn ve všech dnešních moderních prohlížečích, jako je Google Chrome, Opera, Safari, Mozila Firefox a dalších.
- Jedná se o strukturovaný a objektově orientovaný jazyk.
- Jelikož se jedná o implementovaný jazyk, pracuje v běhovém prostředí, a tudíž není potřeba kód předem kompilovat.
- Pomocí konceptu AJAX (Asynchronous JavaScript and XML) dokáže získávat data ze serverů, které následně může využít k zobrazování prvků na webových stránkách.
- Lze pomocí něj zobrazovat a manipulovat prvky HTML.
- Může spouštět části kódu v závislosti na určité události, jako třeba kliknutí nebo přejetí prvku webové stránky, rolování na stránce, stisknutí klávesy na klávesnici a mnoho dalšího. [12]; [13]

2.2 Stručná historie

V počátečních letech World Wide Webu (WWW) mohli prohlížeče zobrazovat pouze statický obsah. To znamená, že neexistovala žádná možnost dynamického chování, jako klikání nebo přejíždění myši přes prvky a věci tomu podobné. Společnost Netscape, které stála za vytvořením tehdy nejpoužívanějšího vyhledávače Netscape Navigator, chtěla však tato omezení odstranit a v roce 1995 představila skriptovací jazyk, který dokáže fungovat na klientských zařízeních, s názvem JavaScript. [12]

2.3 SVG a JavaScript

SVG jsou reprezentovány pomocí objektového modelu dokumentu (DOM). Díky tomu je jejich manipulace pomocí JavaSriptu poměrně snadná. Tento přístup umožňuje vytvářet interaktivní a dynamické grafické prvky na webových stránkách. [14]

2.3.1 Document Object Model

DOM je klíčovým prvkem umožňujícím interaktivitu webových stránek. Díky němu je možné manipulovat s obsahem, strukturou a styly webových stránek. Propojení mezi DOM a internetovým prohlížečem zajišťuje JavaScript. Téměř veškeré interaktivní akce na webových stránkách, jako je například změna obrázků ve slide show, zobrazování chybových hlášek nebo rozbalování menu, jsou realizovány pomocí JavaScriptu, který přistupuje k DOM a manipuluje s ním. [15]

2.3.2 Manipulace SVG pomocí JavaScriptu

Samotný JavaScript poskytuje široké možnosti interakce, manipulace, a dokonce i tvorby samotných SVG prvků. U SVG vložených přímo v HTML kódu, například formou inline, lze ke konkrétním prvkům v JavaScriptu snadno přistupovat pomocí jejich identifikátoru, kterým je atribut „*id*“.

```
 1. // Vytvoreni kruhu
 2. var kruh = document.createElementNS("http://www.w3.org/2000/svg",
        "circle");
 3. kruh.setAttribute("cx", "100");
 4. kruh.setAttribute("cy", "100");
 5. kruh.setAttribute("r", "50");
 6. kruh.setAttribute("fill", "red");
 7.
 8. // Pridani kruhu do SVG
 9. var container = document.getElementById("mySVG");
10. container.appendChild(kruh);
```

Zdrojový kód 2.1 Vytvoření kruhu pomocí JavaScriptu

Klasický zápis JavaScriptu, jak je demonstrováno v ukázce Zdrojový kód 2.1, je vhodný pro základní manipulaci s SVG prvky. Nicméně, při pokročilejších úlohách může takový přístup vést k neefektivitě a obtížím při správě obsáhlého kódu. V takových případech jsou užitečné právě JavaScriptové knihovny, které umožňují zredukovat množství kódu na několik řádků. Tím se zjednodušuje zápis a zlepšuje se čitelnost kódu, což usnadňuje jeho údržbu a rozšiřování. [14]; [16]

3 VYBRANÉ JAVASCRIPTOVÉ KNIHOVNY

Následující část práce zahrnuje vybrané JavaScriptové knihovny, které byly zvoleny a vyzkoušeny autorem práce.

3.1 SVG.js

Knihovna SVG.js je považována za relativně jednoduchou knihovnu, která je primárně navržena pro manipulaci a animaci SVG prvků. Jako taková nemá žádné závislosti a snaží se pokrýt téměř všechny atributy, které SVG nabízí, a současně má za cíl minimalizovat svou velikost. Kromě toho existuje několik pluginů, které umožňují další přizpůsobení knihovny podle potřeb uživatele. Z uživatelského hlediska disponuje velmi přehlednou syntaxí a jednoduchou čitelností kódu. Do projektu jde knihovnu začlenit hned několika způsoby:

- NPM: npm install @svgdotjs/svg.js
- Yarn: yarn add @svgdotjs/svg.js
- Nalinkováním do hlavičky projektu

Jednou z funkcionalit knihovny SVG.js, která je občas chápána uživateli jako chyba, je generování dvou SVG souborů. Kromě SVG souboru, který je viditelný pro uživatele, SVG.js vytváří také jeden neviditelný <*svg*>. Tento skrytý element slouží k výpočtům dat pro různé prvky a zároveň chrání původní dokumenty vytvořené uživatelem před nechtěnými změnami, které by mohly mít nežádoucí efekty. Dalším důvodem pro vytvoření tohoto skrytého SVG je zlepšení výkonu aplikace.
Knihovna SVG.js je distribuována pod open source licencí MIT. [17]

3.2 Vivus

Vivus je JavaScriptová knihovna, která se specializuje na animaci vykreslování tahů v SVG souborech. Tímto způsobem vytváří dojem, že se SVG obrázky kreslí přímo za chodu, což do webu přináší dynamiku a interaktivitu. Knihovna poskytuje několik typů animací a umožňuje nastavit časování přehrávání jednotlivých elementů, díky čemuž lze vytvářet různé varianty animací. Tato funkcionalita je demonstrována i na oficiálních stránkách knihovny. Animace vykresluje prvky v pořadí, v jakém jsou definovány v SVG značce.

Knihovnu Vivus lze, podobně jako u SVG.js, do projektu začlenit podobnými způsoby:

- NPM: npm install vivus
- Bower: bower install vivus
- Nalinkováním do hlavičky projektu

Jak bylo zmíněno, knihovna Vivus je primárně určena pro animování tahů, a tedy objekty s barevnou výplní nemohou být animovány. S pomocí CSS lze dočasně skrýt vyplnění objektů, což umožňuje částečné obejití uvedeného omezení. Nicméně, tento přístup má omezené možnosti manipulace s animovanými objekty.
Totéž platí pro textové elementy. Autor v jedné diskusi uvádí, že možnost převedení textu na tahy byla možná, avšak toto rozšíření by se značně odrazilo na velikosti knihovny. Pokud je tedy nutné text animovat, je nezbytné převést textové prvky na tahy pomocí grafických programů. Knihovna je distribuována pod open source licencí MIT. [18]; [19]

3.3 Snap.svg

Snap.svg je JavaScriptová knihovna pro práci s SVG. Uživatelům poskytuje API pro animování a manipulaci jak s již před vytvořenými SVG, tak s SVG objekty generovanými pomocí samotné knihovny.
Za vytvořením Snap.svg stojí autor Dmitry Baranovskiy, který je také tvůrcem knihovny Raphaël a pracoval také na vývoji exportování SVG souborů z programu Adobe Illustrator. Snap.svg je navržen pro většinu dnešních moderních prohlížečů. Knihovna podporuje funkce jako jsou například maskování, ořezávání, barevné přechody, skupiny a mnoho dalších. [21]
Podobně jako u předešlých knihoven, Snap.svg lze nainstalovat hned několika způsoby:

- Bower: bower install snap.svg
- NPM: npm install snapsvg
- Nalinkováním do hlavičky v HTML souboru pomocí odkazu

Snap.svg je dostupný pod licencí Apache 2, což zaručuje jeho kompletní open-source povahu a zcela bezplatné využívání. Tato licence umožňuje uživatelům volně přizpůsobovat a distribuovat kód knihovny. [20]

4 VYBRANÉ PROGRAMY

Následující část práce zahrnuje vybrané programy a aplikace, které byly zvoleny a vyzkoušeny autorem práce.

4.1 Expressive Animator

Expressive Animator je progresivní webová aplikace navržená speciálně pro animování vektorové grafiky. Umožňuje vytvářet a exportovat plynulé a škálovatelné animace, které jsou ideální pro použití na webových stránkách, mobilních aplikacích a dalších digitálních médiích.

Jednou z unikátních vlastností Expressive Animator je možnost instalace jako progresivní webové aplikace, díky čemuž není potřeba stahovat další software. Tento přístup umožňuje uživatelům okamžitý přístup k funkcím bez nutnosti složité instalace, což zvyšuje přístupnost a flexibilitu aplikace pro uživatele, kteří preferují práci přímo v prohlížeči. Nevýhodou pro některé uživatele může být nutnost použití webového prohlížeče, který využívá jádro Chromium, jako je například Google Chrome, Opera nebo třeba Mozilla Firefox.

Expressive Animator je placená služba, avšak nabízí uživatelům možnost vyzkoušet si ji bezplatně po dobu 14 dní. Tato zkušební lhůta umožňuje uživatelům bez omezení prozkoumat všechny funkce a možnosti aplikace a rozhodnout se, zda je pro jejich potřeby vhodná. [22]

4.2 SVGator

SVGator je webová aplikace zaměřená na tvorbu vektorových animací. Nabízí bezplatnou verzi s omezenými funkcemi, která vyžaduje registraci pomocí e-mailové adresy. Mezi omezení patří maximální délka animace tři sekundy, limit tří exportů animací za měsíc, použití pouze základních typů animací, omezení možností při exportování a další.

SVGator nabízí širokou škálu dostupných tutoriálů, které uživatelům pomáhají lépe porozumět ovládání a funkcím aplikace k vytváření různorodým animacím. Tyto návody jsou k dispozici zcela zdarma. [23]

II. PRAKTICKÁ ČÁST

5 UKÁZKA VYBRANÝCH KNIHOVEN

Hlavní myšlenkou bylo vytvoření interaktivních SVG prvků, které by na základě použití JS knihoven reagovaly na akce prováděné uživatelem přímo na webové stránce.
Při tvorbě jsem se rozhodl využít dodatečně dva programy. Adobe Illustrator určený pro tvorbu vektorové grafiky a Visual Studio Code, který poskytuje prostředí pro psaní a editaci kódu. S oběma programy mám již několikaleté zkušenosti a tato kombinace mi umožnila efektivně vytvořit a integrovat interaktivní SVG prvky do webových stránek.

5.1 SVG.js

Následující část popisuje, jakým způsobem se autor rozhodl knihovnu SVG.js využít. Jakým způsobem při jejím implementování postupoval a uvažoval. Následně pak dochází ke zhodnocení finálního výtvoru.

5.1.1 Myšlenka animovaného prvku a jeho využití

Jeden z prvních konceptů pro vylepšení uživatelského zážitku na webových stránkách se zaměřuje na proces zadávání hesla v registračním formuláři. Běžně se setkáváme s jednoduchými hláškami, které informují uživatele o nesprávném formátu hesla, například "Heslo je příliš krátké" nebo "Heslo obsahuje nepovolené znaky". Rozhodl jsem se tedy tuto prostou hlášku obohatit o vizuální prvek, který se bude měnit v závislosti na délce zadávaného hesla.

5.1.2 Postup vytváření

Jako prvek pro tuto ukázku jsem zvolil SVG v podobě zámku jakožto symbolu zabezpečení. SVG bude mít celkem tři fáze, které se budou v závislosti na délce hesla vizuálně měnit.

5.1.2.1 Tvorba a export SVG

Pomocí základních objektů jsem v programu Adobe Illustrator vytvořil obrázek odemčeného zámku, který jsem následně obarvil a doladil drobnými detaily.

Obrázek 5.1 SVG základního zámku

K takto vytvořenému zámku jsem následně dodělal prvky, které jsem chtěl pomocí animací následně rozpohybovat.

Obrázek 5.2 Fáze animace zámku

Větší skupiny objektů, jako jsou ruce nebo zámek, jsem seskupil do vrstev a pojmenoval je pro snazší orientaci v XML kódu SVG souboru. Názvy vrstev se pak v kódu objevují jako atributy id. Z předchozích zkušeností jsem se rozhodl sloučit všechny tři fáze do jedné skupiny, především kvůli lepší manipulaci a orientaci v kódu. Takto sloučené objekty jsem následně vyexportoval jako soubor SVG.

Obrázek 5.3 Sloučené fáze SVG

5.1.2.2 Kódování

V programu Visual Studio Code jsem vytvořil základní HTML strukturu. V hlavičce <*html*> jsem pomocí URL adresy načetl knihovnu SVG.js ve verzi 3.0. Následně jsem v <*body*> vytvořil <*div*>, do kterého jsem inline vložil SVG tak, jak bylo vyexportováno z Adobe Illustratoru.

Nově jsem vytvořil JavaScriptový soubor, ve kterém jsem pomocí hlavní inicializační funkce *SVG()* a metody *addTo()* vytvořil oblast pro vykreslování SVG grafiky. Pro tuto oblast jsem zvolil element <*div*> v těle HTML dokumentu, ve kterém je SVG vloženo. Pro správnou funkčnost je nezbytné do této vytvořené oblasti předat naše SVG pomocí metody *add()*.

```
5. var draw = SVG().addTo('#svg').size(600, 300);
6. draw.add('#zamek');
```

Zdrojový kód 5.1 Vytvoření vykreslovací oblasti a předání SVG

Pomocí metody *opacity(0)* jsem skryl nepotřebné části SVG pro první fázi animace. Přechod SVG do druhé fáze jsem realizoval pomocí několika metod, které mi umožnily také uložit data do proměnných. Některé metody, jako jsou například *attr()*, *x()* nebo *y()*, které slouží pro získání a nastavení atributů konkrétního elementu, lze totiž použít jako getter a setter zároveň. Uložená data jsem poté aplikoval pro provedení animace z první fáze na druhou a obráceně.

```
111.      var patka = SVG('#patka');
112.     var patkaX = patka.x();
113.     var patkaY = patka.y();
121.     function playAnimationEfect() {
122.         //animace patky zamku
123.         patka.animate(300, 0, 'now').move(patkaX, 33);
```

Zdrojový kód 5.2 Ukázka uzamknutí zámku

V návaznosti jsem do HTML přidal vstupní pole pro heslo a v JavaScriptu jsem využil podmínek k určení, kdy se má animace přehrát na základě počtu znaků v poli hesla.

Obrázek 5.4 Zobrazení v prohlížeči

Pro přechod animace z druhé fáze na třetí jsem využil například metody *rotate()* pro otáčení elementu kolem středového bodu, který jsem získal pomocí metod *cx()* a *cy()*. Animace se následně spouští a obnovuje při splnění podmínek pomocí příslušných funkcí.

```
47.     else if (passwordLength >= 14 && previousLength < 14){
48.        playAnimationArms();
49.       }
50.
51.     else if (passwordLength < 14 && previousLength >= 14){
52.         animationArmsReset()
53.        }
```

Zdrojový kód 5.3 Ukázka podmínek pro zobrazení a skrytí třetí fáze animace

5.1.3 Zhodnocení knihovny a výsledku

Knihovna SVG.js je velmi přístupná i pro začínající vývojáře. Její syntaxe je jednoduše čitelná a často velmi stručná. Na oficiálních stránkách je dobře a přehledně zdokumentovaná, což usnadňuje vyhledání potřebných informací. Téměř každá metoda nebo funkce má uvedený konkrétní příklad kódu, což umožňuje rychle pochopit, jak daná část knihovny funguje. Většina metod funguje v kombinaci s metodou *animate()*, což umožňuje jednoduché animování libovolných prvků. Metoda přijímá celkem tři argumenty: dobu trvání animace, zpoždění a čas, kdy má animace začít.

Co se týká animování cest, SVG.js má pro ně pouze základní podporu. Metoda *plot()*, která slouží pro aktualizování cesty, dokáže animovat pouze cesty, které obsahují stejné příkazy. Pokud chceme tedy například vzhled polygonu pozměnit pomocí zakřivení některých cest, lze toho dosáhnout pomocí metody *plot()*. Avšak tuto změnu nelze animovat pomocí metody *animate()*, což může vyvolat chybovou hlášku v JavaScriptu.

Co se samotného projektu týká, podobá se tomu, jak byl původně zamýšlen. Animace jsou plynulé a fungují tak, jak byly navrženy. Jediným prvkem, který jsem původně chtěl implementovat, ale nedosáhl jsem toho, bylo ve třetí fázi animace. Plánoval jsem, že se sval na každé ruce lehce nadzvedne a vytvoří dojem zatnutého svalu. Avšak, jak bylo popsáno výše ve spojitosti s metodou *plot()*, tato funkcionalita nebyla proveditelná.

I přes tuto drobnost mi knihovna SVG.js přijde jako jednoduchý nástroj pro vytváření rychlých a nenáročných animací, které, při správném použití, dokážou vzbudit dojem větší komplexity.

5.2 Vivus

Následující část podrobně popisuje, jakým způsobem autor zvolil přístup k integraci knihovny Vivus do projektu. Po implementaci autor následně zhodnocuje, jak se mu s JavaScriptovou knihovnou pracovalo.

5.2.1 Myšlenka animovaného prvku a jeho využití

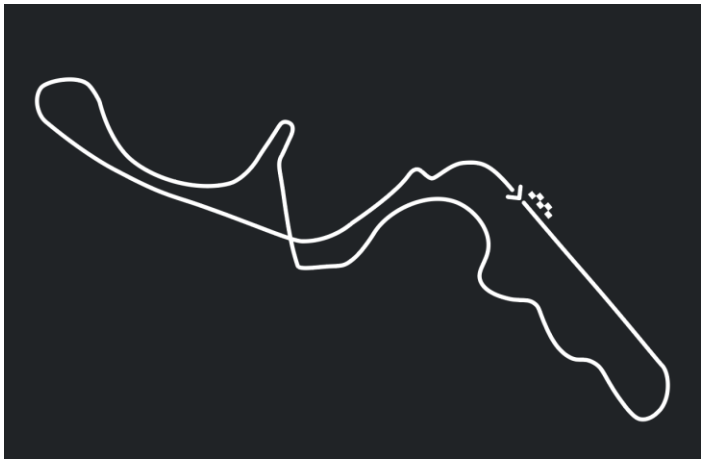
Vzhledem k tomu, že knihovna Vivus je primárně určena pro animaci vykreslování tahů, bylo nezbytné určit prostředí nebo scénář, ve kterém by tato knihovna mohla být efektivně prezentována ke svému využití. Pro tento účel se jako ideální volba ukázala schémata závodních okruhů, jejichž vlastnosti poskytly vhodné parametry pro použití knihovny.

5.2.2 Postup vytváření

Pro ukázku jsem vybral dva závodní okruhy, které jsou známé například ze závodů Formule 1, kde jsem se ostatně také inspiroval. Po načtení se základní schéma okruhu jednoduše vykreslí bílou barvou. U každého okruhu budou umístěna tlačítka, jež po kliknutí spustí animaci, která postupně vykreslí specifické detaily pro daný okruh. Kromě okruhů budou animovány i další dekorativní prvky, jako například vlajka země, ve které se okruh nachází.

5.2.2.1 Tvorba a export SVG

Z veřejně dostupných zdrojů jsem našel schémata okruhů a informace, které jsem chtěl do animace zahrnout. V Adobe Illustratoru jsem pomocí křivek okruhy překreslil včetně zmíněných informací, jako jsou speciální zóny a sektory trati. Vše jsem pečlivě uložil do pojmenovaných vrstev pro jednodušší manipulaci a orientaci v kódu.



Obrázek 5.5 Schéma japonského okruhu Suzuka vytvořeného pomocí křivek

Obrázek 5.6 Schémata vyobrazující konkrétní informace o trati

Obrázek 5.7 Kompletní schéma okruhu připravené pro export do SVG formátu

Kromě okruhů jsem také vytvořil dodatečné ozdobné prvky a vlajky zemí, k otestování animací objektů s výplní pozadí.

5.2.2.2 Kódování

Na rozdíl od předchozí knihovny, kterou jsem naimportoval pomocí URL adresy, jsem pro změnu knihovnu Vivus do projektu zahrnul přímým nainstalováním. Ve mnou vytvořené složce projektu jsem pomocí příkazu *npm install vivus* nainstaloval kompletní knihovnu. Pro lepší přehlednost jsem v projektu navíc vytvořil složky pro umístění stylů a SVG souborů. Pro lepší uživatelský zážitek jsem se se rozhodl implementovat jednoduché boční menu, které umožní uživateli zobrazovat v daném okamžiku pouze jeden okruh. K tomuto účelu jsem využil framework Bootstrap ve verzi 4.4.1, který jsem začlenil do hlavičky projektu. Framework obsahuje styly a pluginy sloužící k usnadnění vývoje stránek.

Stránku jsem následně rozdělil do dvou částí. Levá část pro zobrazení menu a pravá pro zobrazení závodních tratí. SVG okruhů jsem přímo vložil do HTML dokumentu, podobně jako při použití SVG.js. Avšak, v tomto případě jsem musel přesunout styly cest (*path*), které jsou obvykle součástí SVG tagu, do samostatných CSS souborů. Důvodem bylo, že některé cesty v různých SVG souborech měly po vyexportování stejné hodnoty v atributu *„class“*, což způsobovalo nesprávné vykreslování barev a dalších vlastností. Kromě toho jsem pomocí CSS skryl vrstvy, které se budou animovat až na základě stisknutí tlačítka.

```
#britanie{
    .cls-1, .cls-2, .cls-3 {
        stroke: #fff;
      }

      .cls-1, .cls-2, .cls-3, .cls-4, .cls-5, .cls-6, .cls-7, .cls-8 {
        fill: none;
      }

      .cls-1, .cls-2, .cls-4, .cls-5, .cls-6, .cls-7, .cls-8 {
        stroke-linecap: round;
        stroke-width: 3px;
      }
```

Zdrojový kód 5.4 Ukázka stylů tříd SVG

Dále jsem vytvořil JavaScriptový soubor, který má za úkol animovat a interagovat s prvky webové stránky. Knihovna Vivus nabízí tři hlavní typy animací:

- *Sync* – pro synchronní vykreslení tahů
- *Delayed* – každý další tah je vykreslen s lehkým zpožděním
- *OneByOne* – tahy jsou vykreslovány postupně jeden po druhém

Pro vykreslení základních SVG okruhů jsem využil všechny tyto tři typy.

```
    var direction = new Vivus('direction', { type: 'sync', duration: 20 });
    var track = new Vivus('track', { type: 'delayed', duration: 200});
    var flag = new Vivus('flag', { type: 'oneByOne', duration: 40 });
```

Zdrojový kód 5.5 Ukázka využití různých typů animací

Jak je z ukázky Zdrojový kód 5.5 zřejmé, knihovna Vivus má velmi jednoduchou syntaxi. Pomocí *new Vivus()* dojde k vytvoření nové instance objektu Vivus. Ten požaduje tři parametry:

- Identifikátor nebo objekt elementu
- Možnosti objektu
- Zpětné volání volající se na konci animace (nepovinné)

Existuje několik možností, kterými lze průběh animace ovlivnit. V projektu jsem aktivně využíval například:

- *duration*: Určuje dobu, po kterou se bude SVG vykreslovat
- *type*: Určení typu animace.
- *start*: Tato možnost určuje, za jakých okolností se má animace začít přehrávat.

Dále bylo nutné přidat dva ovládací prvky ve formě tlačítek, která by po kliknutí spustila animaci doposud skrytých vrstev SVG. Kromě toho bylo zapotřebí zajistit, že po opětovném kliknutí na tlačítko se vrstvy opět skryjí. K tomu posloužily přímo metody knihovny *play(), stop()* a *reset()*.

Obrázek 5.8 Ukázka SVG s tlačítky

```
14.     var sectors = new Vivus('sectors-lines',{ duration: 50, type: 'sync',
        start: 'manual' });
15.
16.     var isPlayingSec = false;
17.     function animateSectors() {
18.        sectors.play();
19.        if (!isPlayingSec) { //tlacitko je zakliknuto
20.             sectors.play();
21.             document.getElementById('button-sectors').classList.add
                ('active');
22.             document.getElementById('sectors-text').classList.add
                ('finished');
23.             isPlayingSec = true;
24.        } else {          //tlacitko neni zakliknuto
25.             sectors.stop().reset();
26.             document.getElementById('button-sectors').classList.
                remove('active');
27.             document.getElementById('sectors-text').classList.remove
                ('finished');
28.             isPlayingSec = false;
29.         }
30.     }
31.
32.     document.getElementById('button-sectors').addEventListener('click',
         animateSectors);
```

Zdrojový kód 5.6 Funkce pro zobrazení a skrytí animované vrstvy

Jak bylo dříve zmíněno, knihovna sice neposkytuje přímou podporu pro animaci výplní objektů, avšak autor v repozitáři knihovny naznačuje, že tento nedostatek lze částečně obejít pomocí chytrého využití tříd v CSS. Skript počká, až se animace pomocí knihovny dokončí, a poté nastaví třídu elementu, která musí obsahovat atribut *fill-opacity: 1*. Tento „hack“, jak autor naznačuje, jsem proto využil například k vykreslení vlajek států, které fungují jako doplňkový prvek stránky.

```
10. var flagJP = new Vivus('flagJP', { type: 'delayed', duration: 50 },
    function (obj) {
11.     obj.el.classList.add('finished');
12. });
```

Zdrojový kód 5.7 Nastavení stylu elementu na „*finished“* po dokončení animace

5.2.3 Zhodnocení knihovny a výsledku

I přesto, že knihovna Vivus nabízí omezené možnosti pouze v oblasti vykreslování tahů, je zřejmé, že má své využití. S velikostí knihovny do 150 kB je Vivus relativně malou a lehce integrovatelnou knihovnou. Její přehledná dokumentace a jednoduchá syntaxe přispívají k jednoduchosti použití. Výsledné animace pak působí svižně a plynule téměř při jakékoliv kombinaci typů animování. Oficiální stránky knihovny nabízejí dobře zpracovaná dema jednotlivých typů animací a umožňují uživatelům vyzkoušet různé kombinace pro lepší porozumění.

Pro správné fungování knihovny Vivus je nezbytné porozumět způsobu, jakým SVG cesty fungují. Během implementace jsem se osobně několikrát musel uchýlit k alternativním řešením, jelikož některých mnou zamýšlený prvků nebylo možné dosáhnout nebo jejich dosažení bylo příliš komplikované. Nemyslím si však, že by to markantně ovlivnilo zamýšlený vizuál webu.

Kromě knihovny existuje také online nástroj Vivus Instant. Tento nástroj umožňuje uživatelům nahrát SVG a animovat ho podobně jako pomocí JavaScriptové knihovny Vivus. Nicméně, výsledek animace lze vyexportovat pouze jako jednoúčelovou CSS animaci, kterou nelze dále ovládat ani ovlivňovat. Osobně jsem nástroj využil, abych pochopil, jakým způsobem funguje vykreslování cest v knihovně a mohl podle toho uzpůsobit finální podoby použitých SVG. [24]

5.3 Snap.svg

V následující sekci je popsáno, jaký autor práce vybral přístup k integraci knihovny Snap.svg do projektu. Zachycuje, jak během práce postupoval a po dokončení implementace autor následně hodnotí svou zkušenost s prací s tuto JavaScriptovou knihovnou.

5.3.1 Myšlenka animovaného prvku a jeho využití

Vzhledem k tomu, že většina SVG prvků v předchozích knihovnách reagovala na akce uživatele, rozhodl jsem se tuto funkcionalitu aplikovat i při použití knihovny Snap.svg. Záměrem bylo využít komponenty *slideru* v kombinaci s SVG objekty, které by v závislosti na poloze *slideru* měnily svoje atributy.

5.3.2 Postup vytváření

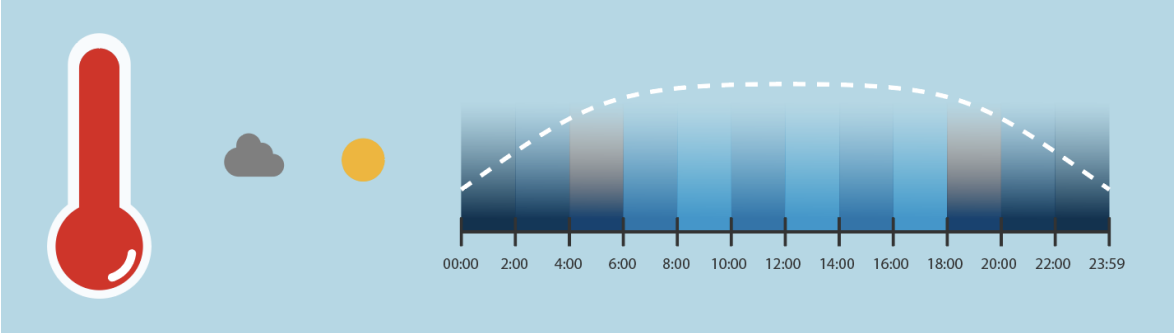
Pro demonstraci této funkcionality jsem se rozhodl implementovat jednoduchou komponentu počasí, která pomocí SVG vizualizuje teplotu a stav počasí v daném časovém úseku. Tato komponenta bude využívat SVG teploměru a časové osy s různými typy počasí.

5.3.2.1 Tvorba a export SVG

Jako první objekt, jehož parametry budou ovlivněny polohou *slideru* je teploměr, který má vizualizovat teplotu pro daný úsek. V programu Adobe Illustrator jsem tedy pomocí základních tvarů vytvořil a následně vyexportoval SVG teploměru.

Dále jsem určil, že pro ukázku zvolím dva stavy počasí: slunečno a zataženo. Pomocí tahů jsem vytvořil jednoduchou ikonu symbolizující zataženo (mrak). Tuto ikonu jsem poté zkopíroval a úpravou tahů a posouváním bodů jsem ji přeměnil na ikonu slunce, znázorňující druhý stav počasí. Důležité bylo zachovat stejný počet bodů, které objekty tvoří, což je klíčové pro správnou transformaci objektu na jiný pomocí JavaScriptu.

Jako poslední objekt jsem vytvořil časovou osu, která bude v kombinaci s ikonami počasí indikovat, na kterou část dne se zobrazované informace vztahují.



Obrázek 5.9 SVG objekty vytvořené pro práci s knihovnou Snap.svg

5.3.2.2 Kódování

Podobně jako u předchozí knihovny jsem využil možnost instalace pomocí příkazového řádku, konkrétně příkazem *npm install snapsvg*. Součástí instalace je také složka s ukázkovými projekty, které mohou uživateli sloužit jako pomůcka pro seznámení s knihovnou. Pro přehlednost jsem si vytvořil novou složku, ve které jsem si následně vytvořil soubory .html, .css a .js.

Jako první jsem se pustil do kódování *slideru*. Původně jsem plánoval vytvořit *slider* pouze pomocí objektů knihovny, avšak nepodařilo se mi přesně nastavit povolené souřadnice pro ovládací prvek *slideru*. Problém byl také s responzivitou. Z toho důvodu jsem se rozhodl využít generativní model AI ChatGPT a s jeho pomocí jsem vytvořil *slider* fungující na principu CSS a JavaScriptu. Následně jsem do HTML přidal SVG teploměru, které jsem pomocí *„id“* předal knihovně. Následně jsem si vytvořil pole pomyslných segmentů obsahujících náhodná data o počasí.

```
var s = Snap("#teplomer");

// Generovani segmentu
var segments = [];
var numSegments = 12;
for (var i = 0; i < numSegments; i++) {
    var temperature = Math.floor(Math.random() * (30 - (-10) + 1)) + (-10);
    // Random  teplota mezi -10 a 30
    var weather = Math.random() < 0.5 ? 'Sunny' : 'Cloudy'; // Random pocasi

    segments.push({
        numSegment: i + 1,
        temperature: temperature,
        weather: weather
    });
}
```

Zdrojový kód 5.8 Předání SVG knihovně Snap.svg a vytvoření segmentů

Následně jsem přiřadil jednotlivé segmenty na komponentu *slideru*. Při pohybu ovládacím prvkem *slideru* se tedy mění hodnoty v závislosti na jeho poloze a segmentu. V návaznosti jsem vytvořil funkci *updateColor()*, která využívá metodu *attr()* z knihovny Snap.svg pro změnu atributů prvku.

```
// Zmena barevnosti na zaklade teploty v segmentu
function updateColor(temperature) {
    var baseCircle = s.select("#base");
    if (temperature <= 8) {
        // modra
        baseCircle.attr({ fill: '#0b9fcc' }, 300);
        temp.attr({ fill: '#0b9fcc' }, 300);
    } else if (temperature > 8 && temperature <= 18) {
        // zluta
        baseCircle.attr({ fill: '#f7b30b' }, 300);
        temp.attr({ fill: '#f7b30b' }, 300);
```

Zdrojový kód 5.9 Část funkce pro změnu barevnosti teploměru

Kromě barevné vizualizace jsem chtěl změnu teploty vyobrazit i animovaným způsobem. Pomocí přepočtu teploty v segmentu na výšku vnitřního objektu teploměru, který

symbolizuje stupnici, a metody *animate()* jsem docílil toho, že se stupnice teploměru v závislosti na teplotě segmentu snižuje nebo naopak zvyšuje.

Jako další krok jsem implementoval časovou osu. Chtěl jsem, aby se stav počasí měnil v závislosti na poloze *slideru* po přerušované čáře, kterou je možné vidět na Obrázku 5.9, a tím znázorňovat aktuální stav počasí daného segmentu. Ikonu počasí jsem pomocí metody *transform()* umístil na počáteční bod cesty, který jsem získal pomocí metody *getPointAtLength(0)*. Tuto funkcionalitu jsem implementoval do funkce, která je následně umístěna v části kódu, která naslouchá události pohybu *slideru*. Pokud dojde k pohybu, ikona počasí se přesune po cestě, na kterou je přiřazena.

```
163. // Vypocet a umisteni stredu objektu na zacatek cesty
164. var weatherStateBBox = weatherState.getBBox();
165. var movePoint = weatherPath.getPointAtLength(0);
166. var dx = movePoint.x - weatherStateBBox.cx;
167. var dy = movePoint.y - weatherStateBBox.cy;
168. weatherState.transform('t' + dx + ',' + dy);
```

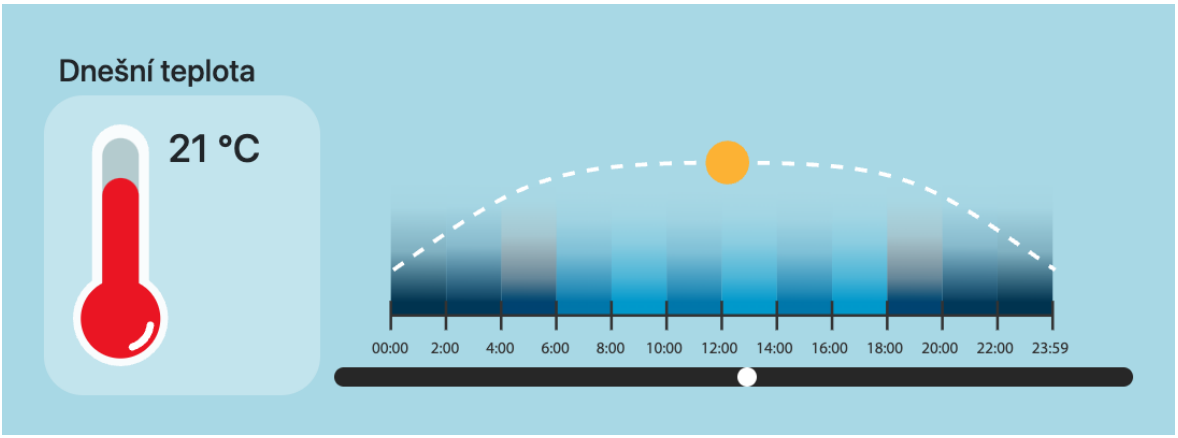
Zdrojový kód 5.10 Přiřazení ikony počasí na počáteční bod cesty

Poslední funkcionalitou, kterou jsem implementoval, byla změna ikony počasí v závislosti na stavu počasí v daném segmentu. Nejprve jsem uložil cesty ikon do proměnných. Poté jsem vytvořil funkci, která na základě podmínek a metod *animate()* a *attr()* změní ikonu podle počasí v segmentu na slunečno nebo zataženo.

```
// Funkce pro aktualizaci stavu pocasi v segmentu
function updateWeatherState(weather) {
    if (weather === "Sunny") {
        weatherState.animate({ path: sunPath }, 200).attr({
            fill: "#f7b30b" });
    } else {
        weatherState.animate({ path: cloudPath }, 200).attr({
            fill: "#7f7f7f" });
    }
}
```

Zdrojový kód 5.11 Funkce pro animovanou změnu ikony počasí

K závěru jsem využil metody *group()* a *append()* k seskupení objektů, jako je časová osa, ikona počasí a cesta, po které se ikona pohybuje do jedné skupiny. Tím jsem zajistil lepší možnost manipulace a responzivity na zobrazované stránce.



Obrázek 5.10 Ukázka kompletního projektu vytvořeného pomocí Snap.svg

5.3.3 Zhodnocení knihovny a výsledku

Na první pohled je knihovna Snap.svg podobná knihovně SVG.js. Názvy metod jsou často stejné nebo podobné a obě knihovny umožňují provádět podobné operace. Hezky zpracované demo ukázky názorně ukazují, čeho všeho lze s pomocí knihovny Snap.svg docílit.
Nicméně musím vytknout způsob, jakým je knihovna Snap.svg zdokumentována. Na oficiálních stránkách sice naleznete úvodní tutoriál, ten je však poměrně stručný a neposkytuje dostatek informací. Dokumentace pak obsahuje seznam všech metod, které knihovna nabízí, avšak ne vždy je zřejmé, jak přesně je implementovat do projektu. Například metoda *transform()* pracuje s parametrem, kterým je textový řetězec. Jakým způsobem tento textový řetězec správně formulovat už v dokumentaci uvedeno není. Z toho důvodu jsem byl nucen vyhledat externí stránku s návody, která některé z funkcí popisuje, a to včetně ukázky kódu, která je v oficiální dokumentaci knihovny jen u některých metod. [25]
Je však nutné dodat, že i přes zmíněné potíže jsem s výslednou podobou implementace knihovny Snap.svg spokojen.

6 UKÁZKA VYBRANÝCH PROGRAMŮ

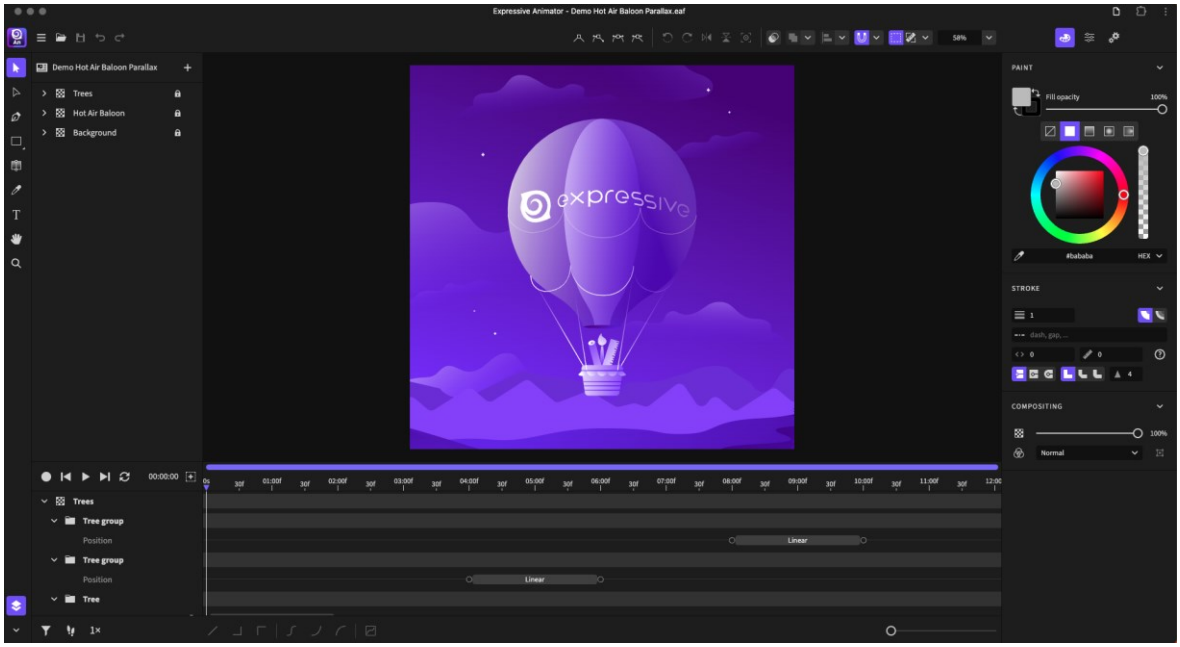
Primárním záměrem bylo prověřit možnosti tvorby animovaných SVG pomocí běžně dostupných aplikací a programů, bez ohledu na to, zda jsou zdarma či za poplatek. Vzhledem k omezeným možnostem exportu animovaných SVG z těchto nástrojů je vhodnější využívat je pro tvorbu opakujících se animací.
K implementaci animovaných SVG bylo opět využito programu Visual Studio Code.

6.1 Expressive Animator

Překvapením bylo zjištění, že Expressive Animator funguje na bázi progresivní webové aplikace. Na oficiálním webu je tato informace nepřímo naznačena a je poměrně snadno přehlédnutelná.
Aplikace vyžaduje měsíční poplatek ve výši 15 dolarů, avšak poskytuje i bezplatnou dvoutýdenní zkušební verzi, kterou jsem pro účely této práce využil. Po spuštění mi prohlížeč nabídl možnost instalace aplikace, což jsem pro pohodlnější používání také provedl.

6.1.1 Práce s aplikací

Aplikace na první pohled působí velmi přehledně. Na levé straně obrazovky se nachází menu nástrojů společně s rozdělením vrstev, dole časová osa, vpravo menu k nastavení vlastností objektu a uprostřed se nachází hlavní pracovní plocha pro vytváření animací.



Obrázek 6.1 Aplikace Expressive Animator s demo ukázkou

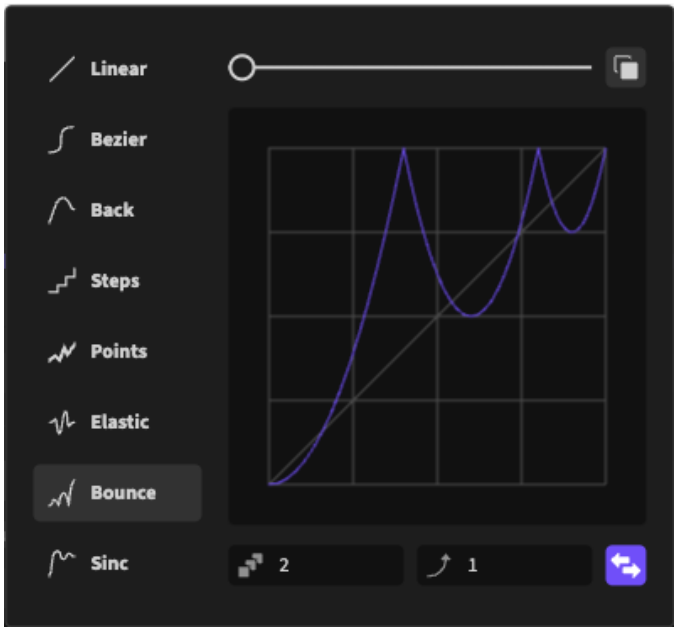
Na základě prvotního zkoušení aplikace jsem zjistil, že výsledné animace lze exportovat v několika různých formátech, včetně WebM, MP4, APNG a dalších. Pro účely této práce je však postačující export animací ve formátu animovaného SVG. U tohoto formátu je možné nastavit parametry animace, jako je počet opakování, rychlost přehrávání nebo třeba možnost přehrání animace pozpátku po jejím skončení. Osobně mě zaujala možnost spuštění animace jednoduchým kliknutím myši přímo na SVG.

Vznikl tedy nápad vytvořit animaci, která bude tento způsob spuštění přehrávání implementovat. Záměrem bylo, aby SVG před spuštěním animace vzbuzovalo dojem tlačítka, na které lze kliknout. Pomocí nástrojů jsem tedy vytvořil objekt připomínající tlačítko s ikonou pro kliknutí.

Obrázek 6.2 Počáteční stav animace

Následně jsem se rozhodl do animace zakomponovat logo a název Fakulty aplikované informatiky. Aplikace Expressive Animator nabízí velké množství typů animací. Pro ukázku byla využita možnost animace *Morph*, která dokáže měnit tvar cesty. Tímto způsobem se obyčejný kruh dokáže přeměnit do tvaru loga fakulty. Tento typ animace bylo poměrně obtížné docílit pomocí JavaScriptových knihoven, a z toho důvodu jsem si jej chtěl vyzkoušet právě zde.

Samotným typům animací, jako jsou právě *Morph*, *Rotate*, *Position* a další, lze nastavit způsob, jakým se mají přehrávat. U typu *Position*, který dokáže změnit polohu objektu, například lze nastavit možnost přehrávání *Bounce*. Toto nastavení zajistí, že animovaný objekt bude mít efekt poskočení a není tak nutné, jakkoliv dodatečně měnit cestu animace.



Obrázek 6.3 Okno sloužící k upravení průběhu animace

6.1.2 Zakomponování SVG do webu

Jak již bylo uvedeno v teoretické části práce, SVG lze do webových stránek implementovat několika způsoby. Nicméně, ne všechny jsou vhodné pro animovaná SVG. Například, i když je možné implementovat SVG pomocí elementu <*img*> nebo pomocí CSS *background-image*, animace v těchto případech není spustitelná.

Zakomponování animace pomocí <*iframe*> sice poskytuje podporu, nicméně některé webové prohlížeče standardně zobrazují <*iframe*> s ohraničením, což může být rušivé. Toto ohraničení je obvykle nutné odstranit pomocí CSS. Další možností, která je funkční, ale pro mnoho uživatelů pravděpodobně nevhodná a nevkusná, je vložení SVG přímo inline. Tím sice lze přehrát animaci, avšak množství kódu vytvořeného SVG může negativně ovlivnit přehlednost zdrojového kódu stránky. Pro představu, kód SVG vytvořené pro tuto ukázku obsahuje přes 50000 znaků.

Zbývají tedy možnosti využití elementů <*object*> a <*embed*>. Co se týká animace, obě možnosti fungují podobně a jsou tak ideální pro implementaci takto animovaných SVG souborů.

6.1.3 Zhodnocení

Aplikace disponuje uživatelsky příjemným a intuitivním rozhraním. Možnost kompletního vyzkoušení aplikace po dobu dvou týdnů zdarma nabízí uživatelům příležitost se seznámit s funkcionalitou a usnadňuje jim rozhodování, zda chtějí za poplatek aplikaci i nadále

využívat. Progresivní webová aplikace na rozdíl od klasické webové aplikace, umožňuje uživatelům pracovat bez nutnosti připojení k internetu, díky čemuž je aplikace dostupná i v prostředích s omezeným nebo žádným připojením.

Je nevhodné upravovat přímo kód animovaného SVG, například pro změnu vzhledu, protože kód bývá příliš obsáhlý a nepřehledný. Při exportování jsou také odstraněna veškerá pojmenování prvků a vrstev, což znesnadňuje jejich dohledání v kódu. Je proto lepší a pohodlnější upravit animaci zpětně v aplikaci, znovu ji vyexportovat a přehrát soubor v projektu.

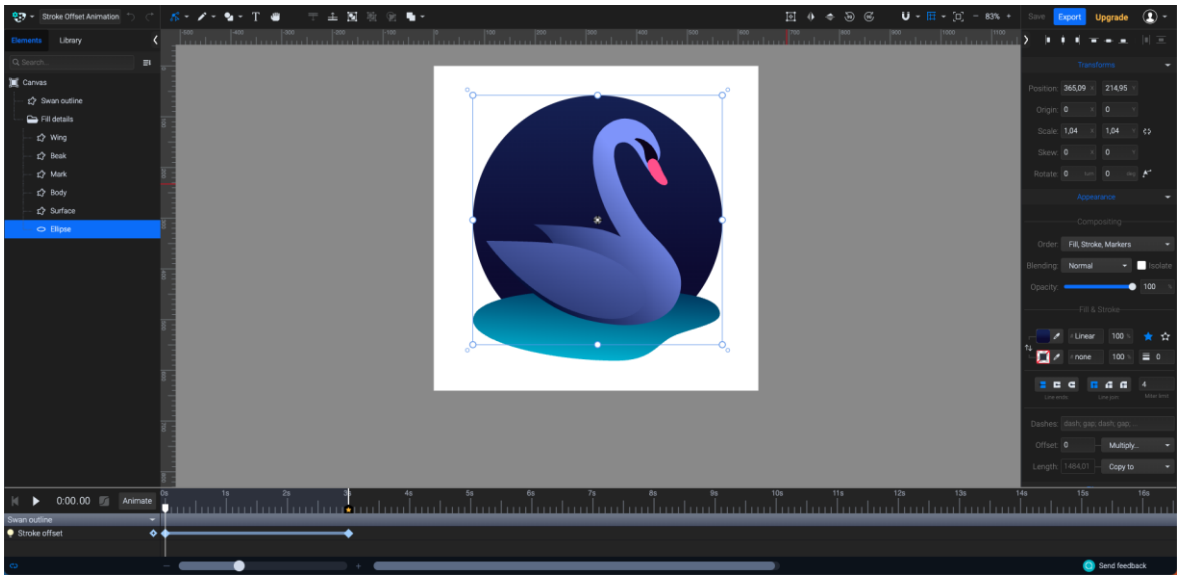
Pro zajímavost jsem vyexportoval demo, které aplikace při spuštění zobrazí, ve formátu animovaného SVG a ve formátu WebM. Výsledný soubor ve formátu WebM měl téměř 24krát větší velikost (10,7 MB) než SVG, přestože byl podroben poměrně velké kompresi. To jen podtrhuje výhodu SVG vzhledem k velikostem souboru.

6.2 SVGator

Na rozdíl od předchozí aplikace je SVGator čistě webovou aplikací, což znamená, že k jejímu používání je nezbytné trvalé internetové připojení.
Plná verze aplikace stojí měsíční poplatek ve výši 28 dolarů, což může být pro spoustu uživatelů odrazující částka. Pro méně náročné uživatele lze SVGator používat kompletně zdarma, avšak s velmi omezenými možnostmi.

6.2.1 Práce s aplikací

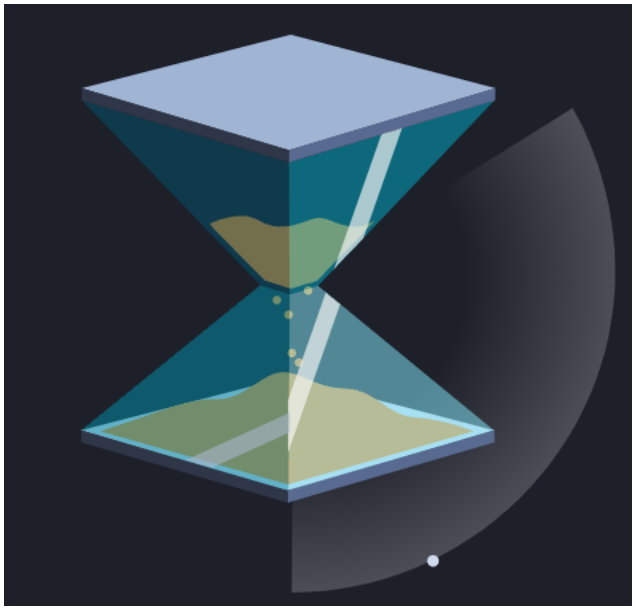
Po registraci aplikace přesměruje uživatele na stránku s uloženými projekty. Zde se nachází několik demo projektů, které obsahují různé typy animací. To umožňuje uživatelům například přiučit se, jakým způsobem animace fungují. Po vytvoření projektu vás aplikace pomocí informačních oken, seznámí se základním rozložením a funkcionalitou nástrojů a panelů, což je pro začínající uživatele, kteří s podobnými programy nebo aplikacemi doposud nepracovali, výhodou. SVGator na první pohled připomíná rozložením předchozí aplikaci Expressive Animator, avšak s drobnými odlišnostmi.



Obrázek 6.4 Aplikace SVGator s předpřipraveným demo projektem

Jelikož zkušební verze zdarma nabízí maximálně třísekundovou dobu animace, chtěl jsem vytvořit plynulou animaci, která by se neustále opakovala, s plynulým navazováním konce na začátek. Jako ukázku jsem pomocí nástrojů v aplikaci vytvořil vektorovou podobu přesýpacích hodin, které se neustále přesýpají díky možnosti animace *Position*. Nicméně to by

pro vyzkoušení programu bylo příliš málo. Pro zpestření jsem využil možnost maskování objektů a vytvořil jsem efekt odlesku, který se opakuje jednou za každé přehrání animace. Díky maskování se objekt zobrazuje pouze ve vymezeném prostoru, v tomto případě na „skle“ hodin.



Obrázek 6.5 Ukázka SVG hodin ve druhé sekundě animace

Animace se mi však stále zdála příliš nevýrazná. Do pozadí hodin jsem přidal kruh, rozdělený na tři stejné části. Pomocí možnosti animace *Opacity* jsem nastavil, aby se v každé sekundě změnila viditelnost a jedna po druhé se tak postupně zaktivovala. Také jsem přidal druhý malý kruh jako indikátor, který při spuštění animace obepisuje kružnici velkého kruhu. Toho jsem docílil jednoduše označením dvou objektů a zvolením možnosti *Set motion path*, která nastaví trasu animace horního objektu na trajektorii tahu spodního objektu.

6.2.2 Zakomponování SVG do webu

Podobně jako u aplikace Expressive Animator, i SVGator nabízí několik možností exportu animace. Nicméně ne všechny jsou dostupné ve verzi zdarma. Při exportu jako animované SVG máme možnost vybrat, zda se animace bude přehrávat na základě JavaScriptu nebo CSS. Soubor s animací pomocí CSS je sice menší, ale může nastat problém s podporou některých prohlížečů pro určité typy animací. [26]

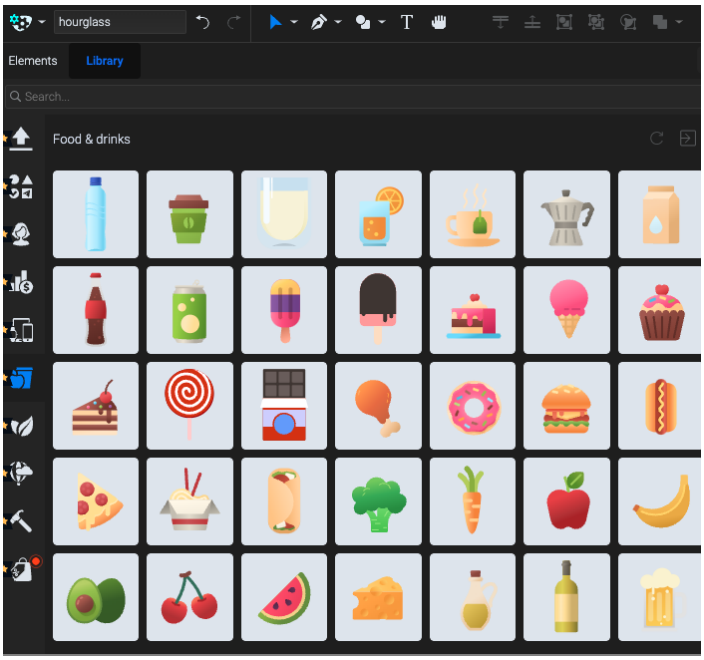
Dále je možné nastavit export *„id“* jednotlivých prvků tak, jak si je uživatel pojmenoval ve vrstvách. Ve vygenerovaném SVG souboru lze potom tyto prvky a vrstvy jednoduše

dohledat pomocí právě zmíněných *„id“*. V plné verzi aplikace pak jde při exportu nastavit například rychlost přehrávání, podmínka, kdy se má animace začít přehrávat a další parametry.

Vyexportované SVG lze poté, podobně jako u předchozí aplikace, jednoduše přidat do projektu a pomocí elementu <*object*> zobrazil.

6.2.3 Zhodnocení

S aplikací SVGator se pracovalo velmi dobře, i když některé klávesové zkratky pro ovládání nebyly podobné jako u jiných grafických programů a musel jsem si na ně chvíli zvykat. Pro uživatele, kteří by s tímto programem chtěli pracovat i nadále, určitě doporučuji plnou verzi aplikace, i když částka může být poměrně odrazující. SVGator má například k dispozici funkci knihovny, odkud lze do projektu přidat již před vytvořené prvky a usnadnit si tak práci. Kromě toho lze do knihovny také ukládat vlastní vytvořené prvky k opakovanému použití.



Obrázek 6.6 Ukázka knihovny předem generovaných objektů

Další zajímavou funkcí plné verze programu je export animace jako *Programmatic*. Díky tomu lze v kombinaci s SVGator JS API animace spouštět pomocí JavaScriptu přímo v kódu podle libosti. Pokud je tedy potřeba, můžeme jednotlivé části SVG animovat pomocí různých vstupů, jako je třeba kliknutí myši, stisk klávesy a podobně. [27]

Na oficiálních webových stránkách lze dohledat spousty informací a tipů, jak aplikaci využít. Všechny tyto informace jsou doprovázeny různorodými SVG animacemi, které jdou právě pomocí aplikace SVGator vytvářet.

ZÁVĚR

Cílem této bakalářské práce bylo seznámit čtenáře s možnostmi využití animovaných SVG na webových stránkách.

Teoretická část popisuje základní funkcionalitu formátu SVG a jeho výhody. Pomocí konkrétních ukázek kódů je znázorněno, jak lze SVG do webu přímo implementovat. Dále je také představen jazyk JavaScript, který například umožňuje uživatelské interakce, dynamické změny vzhledu stránky a manipulaci s jejím obsahem. Práce se především zaměřuje na animování obsahu webové stránky pomocí JavaScriptu v kombinaci s právě zmiňovaným SVG. Na závěr teoretické části jsou pro ukázku představeny JavaScriptové knihovny, které usnadňují manipulaci a implementaci animací SVG souborů do webu. Vedle těchto knihoven jsou také představeny aplikace, určené přímo pro export kompletních animovaných SVG souborů.

Následující praktická část se podrobněji věnuje knihovnám a aplikacím popsaných v teoretické části. Pro demonstraci možností, které toto téma nabízí, autor práce uvedl v potaz různé scénáře, ve kterých lze animované prvky SVG aplikovat. Tyto ukázky následně autor vytvořil. Autor práce vysvětluje, jakým způsobem uvažoval při vymýšlení a tvorbě konkrétních případů užití, s jakými problémy se při tvorbě potýkal, jak tyto problémy dokázal řešit nebo případně alternativně obejít.

Při vytváření této bakalářské práce, autor získal cenné zkušenosti v rámci práce s jazykem JavaScript a formátem SVG, které může kreativně zužitkovat při jeho soukromé tvorbě nebo v zaměstnání.

SEZNAM POUŽITÉ LITERATURY

[1] LARSEN, Rob. *Mastering SVG*. Online. Packt Publishing, 2018. ISBN 9781788621984. Dostupnéz: https://www.google.cz/books/edition/Mastering_SVG/3TJwDwAAQBAJ?hl=cs&gbpv=0&kptab=overview. [cit. 2024-02-11].

[2] BELLAMY-ROYDS, Amelia; CAGLE, Kurt a STOREY, Dudley. *Using SVG with CSS3 and HTML5*. Online. O'Reilly Media, 2017. ISBN 9781491921920. Dostupné z: https://www.google.cz/books/edition/Using_SVG_with_CSS3_and_HTML5/yWU6DwAAQBAJ?hl=cs&gbpv=0. [cit. 2024-02-11].

[3] *Adding vector graphics to the web*. Online. MND Web Docs. 2024. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/HTML/Multimedia_and_embedding/Adding_vector_graphics_to_the_Web. [cit. 2024-02-16].

[4] *SCALABLE VECTOR GRAPHICS (SVG)*. Online. *W3C*. 2010. Dostupné z: https://www.w3.org/Graphics/SVG/. [cit. 2024-02-21].

[5] *Obrázky na webu*. Online. *KIK*. 2015. Dostupné z: http://kik.osu.cz/moodle/mod/page/view.php?id=1713. [cit. 2024-03-01].

[6] YIP, Thomas. Comparing SVG and PNG File Sizes. Online. vecta.io. 2018. Dostupné z: https://vecta.io/blog/comparing-svg-and-png-file-sizes. [cit. 2024-03-01].

[7] *SVG files*. Online. Adobe. C2024. Dostupné z: https://www.adobe.com/creativecloud/file-types/image/vector/svg-file.html. [cit. 2024-03-01].

[8] ASIKPO, Edidiong. How to Use SVG Images in CSS and HTML – A Tutorial for Beginners. Online. *freeCodeCapm*. 2020. Dostupné z: https://www.freecodecamp.org/news/use-svg-images-in-css-html/. [cit. 2024-03-06].

[9] SOUEIDAN, Sara. Making SVGs Responsive with CSS. Online. *Codrops*. 2014. Dostupné z: https://tympanus.net/codrops/2014/08/19/making-svgs-responsive-with-css/. [cit. 2024-03-07].

[10] PICARDAL, Ryan. The Importance of Icons in Your Website Design. Online. *Power Digital*. 2018. Dostupné z: https://powerdigitalmarketing.com/blog/the-importance-of-icons-in-your-website-design/. [cit. 2024-03-14].

[11] *How To Add Icons*. Online. *Font Awesome Docs*. Dostupné z: https://docs.fontawesome.com/web/add-icons/how-to. [cit. 2024-03-14].

[12] MULRAJ SHAH, Rushabh. *Decoding JavaScript*. English Edition. Bpb Publications, 2021. ISBN 9789390684816. [cit. 2024-03-18].

[13] *JavaScript*. Online. *MND Web Docs*. 2024. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript. [cit. 2024-03-18].

[14] COLLINGRIDGE, Peter. Using Javascript with SVG. Online. *Peter Collingridge*. 2018. Dostupné z: https://www.petercollingridge.co.uk/tutorials/svg/interactive/javascript/. [cit. 2024-03-22].

[15] RASCIA, Tania. *Understanding the DOM — Document Object Model*. Online. DigitalOcean, 2020. ISBN 9780999773093. Dostupné z: https://www.google.cz/books/edition/Understanding_the_DOM_Document_Object_Mo/fNQBEAAAQBAJ?hl=cs&gbpv=0. [cit. 2024-03-26].

[16] MATTHEW, David. *Generative Art with JavaScript and SVG*. Online. Apress, 2024. ISBN 9798868800863. Dostupné z: https://www.google.cz/books/edition/Generative_Art_with_JavaScript_and_SVG/T9L8EAAAQBAJ?hl=cs&gb. [cit. 2024-03-26].

[17] *SVG.js*. Online. C2012-2021. Dostupné z: https://svgjs.dev/docs/3.0/. [cit. 2024-04-05].

[18] maxwellito / vivus. Online. *GitHub*. 2022. Dostupné z: https://github.com/maxwellito/vivus. [cit. 2024-04-21].

[19] *Vivus*. Online. Dostupné z: https://maxwellito.github.io/vivus/. [cit. 2024-04-21].

[20] *Snap.svg*. Online. Dostupné z: http://snapsvg.io/. [cit. 2024-04-25].

[21] *Dmitry Baranovskiy*. Online. Dostupné z: http://dmitry.baranovskiy.com/. [cit. 2024-04-25].

[22] *Expressive Animator*. Online. *Expressive*. C2024. Dostupné z: https://expressive.app/expressive-animator/. [cit. 2024-04-16].

[23] *SVGator*. Online. C2024. Dostupné z: https://www.svgator.com/. [cit. 2024-04-18].

[24] *Vivus instant*. Online. Dostupné z: https://maxwellito.github.io/vivus-instant/. [cit. 2024-04-21].

[25] *Snap.svg Tutorial*. Online. Dostupné z: https://svg.dabbles.info/. [cit. 2024-04-25].

[26] *Browser support for animated SVG*. Online. *SVGator*. C2024. Dostupné z: https://www.svgator.com/help/technical-issues/browser-support. [cit. 2024-04-25].

[27] *Animate Programmatically*. Online. SVGator. C2024. Dostupné z: https://www.svgator.com/help/getting-started/animate-programmatically. [cit. 2024-04-25].

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

SVG	Scalable Vector Graphics
GIF	Graphics Interchange Format
APNG	Animated Portable Network Graphics
JS	JavaScript
XML	Extensible Markup Language
HTML	Hypertext Makup Language
CSS	Cascading Style Sheet
PNG	Portable Network Graphics
JPEG	Joint Photographic Experts Group
AJAX	Asynchronous JavaScript and XML
WWW	World Wide Web
DOM	Document Object Model
MIT	Massachusettský technologický institut
NPM	Node Package Manager
URL	Uniform Resource Locator
AI	Artificial intelligence
API	Application programming interface

SEZNAM OBRÁZKŮ

SEZNAM ZDROJOVÝCH KÓDŮ