Střední průmyslová škola a Vyšší odborná škola, Liberec, příspěvková organizace

VanGogh Painter

Maturitní práce

Autor **jan.holy.021**

Obor **Technické lyceum**

Vedoucí práce **Ing. Tomáš Kazda DiS.**

Školní rok **2024/2025**

Počet stran **15**

Počet slov **1116**



Anotace

Práce se zabývá spojením VanGogh Painter robota s webovým rozhraním. Vychází z nekompletního a uživatelsky neideálního projektu. Přináší nové webové rozhraní s intuitivním rozhraním a vylepšení pro pohodlnější a širší možnosti použití, širší veřejnosti.

Summary

This work deals with connecting the VanGogh Painter robot with a web interface. It is based on an incomplete and user-ideal project. It brings a new web interface with an intuitive interface and enhancements for more convenient and broader possibilities of use to a wider audience.

Čestné prohlášení

Prohlašuji, že jsem předkládanou maturitní práci vypracoval sám a uvedl jsem veškerou použitou literaturu a bibliografické citace.

V Liberci dne

jan.holy.021

Obsah

[Úvod 1](#_Toc190905641)

[1 Technologie a prostředí 2](#_Toc190905642)

[1.1 WebStorm 2](#_Toc190905643)

[1.2 React 2](#_Toc190905644)

[1.3 Micro:bit Web Bluetooth 2](#_Toc190905645)

[1.3.1 Micro:bit Web Components 2](#_Toc190905646)

[1.4 SVG Editor 3](#_Toc190905647)

[1.5 VanGogh Extension 3](#_Toc190905648)

[1.6 Figma 3](#_Toc190905649)

[1.7 Převod z SVG do Turtle Graphics 3](#_Toc190905650)

[2 Webová aplikace 6](#_Toc190905651)

[2.1 UI/UX 6](#_Toc190905652)

[2.2 Mechaniky 6](#_Toc190905653)

[2.2.1 Připojení robota k webu 6](#_Toc190905654)

[2.2.2 Protokol komunikace 10](#_Toc190905655)

[2.2.3 Zobrazení cesty robota v reálném čase 10](#_Toc190905656)

[2.2.4 Editor SVG souborů 10](#_Toc190905657)

[2.2.5 Škálování SVG 10](#_Toc190905658)

[3 Problémy 11](#_Toc190905659)

[3.1 Bluetooth připojení 11](#_Toc190905660)

[3.2 Editor SVG 11](#_Toc190905661)

[Závěr 13](#_Toc190905662)

[Seznam zkratek a odborných výrazů 14](#_Toc190905663)

[Seznam obrázků 15](#_Toc190905664)

[Použité zdroje 16](#_Toc190905665)

[A. Seznam přiložených souborů I](#_Toc190905666)

Úvod

Cíl této práce je vytvořit webovou aplikaci pro VanGogh Painter robota. Web je schopen přeložit SVG soubory na Turtle Graphics pokyny, kterým může robot rozumět. Také je přímo v aplikaci možnost jednoduchých úprav daného SVG souboru a škálování reálného výstupu podle potřeby pomocí pravítka, které ukazuje jeho reálnou velikost. Aplikace je uživatelsky přívětivá a jednoduchá na použití. Průběh vykreslení je graficky znázorněn zobrazením celé cesty robota a zvýrazněním již vypracované části.

Celá aplikace byla nejprve navrhnuta v programu Figma a poté naprogramována převážně v TypeScript Reactu v prostředí WebStorm.

# Technologie a prostředí

Práce je vypracována v TypeScript React v prostředí WebStorm od společnosti JetBrains. K spojení a komunikaci byla použita knihovna Micro:bit Web Bluetooth a její rozšíření Micro:bit Web Components. Pro úpravu SVG souborů byl použit již existující webový SVG editor. Design byl navrhnut v aplikaci Figma.

## WebStorm

WebStorm je výkonné integrované vývojové prostředí (IDE) vyvinuté společností JetBrains, které je optimalizované pro vývoj v JavaScriptu a souvisejících technologiích, včetně Reactu. Nabízí pokročilé funkce jako automatické doplňování kódu, integrovanou podporu pro verzovací systémy a ladicí nástroje, což vývojářům usnadňuje práci a zvyšuje jejich produktivitu.

## React

React je open-source JavaScriptová knihovna vyvinutá společností Meta (dříve Facebook) pro tvorbu uživatelských rozhraní. Používá komponentově orientovaný přístup, což umožňuje vývojářům vytvářet znovupoužitelné prvky aplikace. Díky využití virtuálního DOM (Document Object Model) je React velmi efektivní při aktualizaci a vykreslování uživatelského rozhraní, čímž zvyšuje výkon webových aplikací.

## Micro:bit Web Bluetooth

Micro:bit Web Bluetooth je technologie umožňující bezdrátovou komunikaci mezi zařízením micro:bit a webovou aplikací pomocí protokolu Bluetooth Low Energy (BLE). Díky této technologii mohou vývojáři vytvářet interaktivní webové aplikace, které komunikují s micro:bitem přímo v prohlížeči bez potřeby instalace dodatečného softwaru. To otevírá možnosti pro vzdělávací a IoT aplikace s jednoduchou konektivitou.

### Micro:bit Web Components

Micro:bit Web Components je sada webových komponent navržených pro usnadnění integrace funkcionality micro:bit do webových aplikací. Komponenty umožňují snadné ovládání micro:bitu, přístup k jeho senzorům a interakci s externím hardwarem prostřednictvím standardních webových technologií. Díky použití Web Components mohou vývojáři jednoduše přidávat micro:bit funkce do svých projektů bez nutnosti hlubšího porozumění nízko úrovňové komunikaci s hardwarem.

## SVG Editor

Tento webový SVG editor je jednoduchý nástroj pro tvorbu a úpravu vektorové grafiky přímo v internetovém prohlížeči. Umožňuje uživatelům vytvářet základní tvary intuitivním způsobem bez nutnosti instalace specializovaného softwaru. Editor podporuje základní funkce jako kreslení čar, křivek a tvarů, přičemž umožňuje jejich přesné umístění. Díky tomuto editoru lze přímo na webové aplikaci tvořit a upravovat vektorovou grafiku přímo v prohlížeči s možností exportu pro další využití.

## VanGogh Extension

Micro:bit VanGogh Extension je rozšíření pro Microsoft MakeCode, které umožňuje interaktivní využití pro micro:bit při práci s kresbou a grafikou. Rozšíření poskytuje nástroje pro ovládání micro:bit LED displeje a umožňuje programování jednoduchých animací a vzorů. Je vhodné pro vzdělávací účely a podporuje kreativní programování v prostředí MakeCode.

## Figma

Figma je cloudový nástroj pro návrh uživatelského rozhraní, který umožňuje týmovou spolupráci v reálném čase. Díky svému intuitivnímu rozhraní a široké škále funkcí, jako jsou vektorové kreslení, komponenty a automatizované layouty, je ideální pro návrh moderních webových a mobilních aplikací. Figma umožňuje snadné sdílení návrhů a zpětnou vazbu, což usnadňuje iterativní vývoj a zlepšuje efektivitu designového procesu.

## Převod z SVG do Turtle Graphics

Tento kód slouží k převodu vektorových cest ve formátu SVG na sadu příkazů, kterým rozumí micro:bit robot v prostředí **Turtle Graphics**. Díky tomu je možné převést digitální kresbu do pohybů robota a následně ji fyzicky vykreslit na papír. Celý proces zahrnuje několik kroků – nejprve se načtou a analyzují příkazy z SVG souboru, poté se vypočítají správné pohyby, a nakonec se výsledná data převedou do formátu, kterému robot rozumí.

Kód zpracovává několik základních **SVG příkazů** – například **M (Move)** pro přesunutí pera bez kreslení, **L (Line)** pro nakreslení čáry mezi dvěma body, **H (Horizontal Line)** a **V (Vertical Line)** pro pohyb pouze v jedné ose. Kromě jednoduchých čar podporuje i složitější **Beziérovy křivky**, které jsou rozděleny na menší segmenty a vykresleny jako posloupnost krátkých přímek, aby robot dokázal přesně napodobit jejich tvar.

Správné fungování robota závisí na přesném výpočtu vzdáleností a úhlů mezi jednotlivými body. Tento výpočet zajišťuje funkce calcRightTriangle(a, b), která pomocí Pythagorovy věty určí délku přepony trojúhelníku a spočítá úhel, pod kterým se robot musí otočit. Po stanovení úhlu funkce deltaWrite(deltaAngle, output) rozhodne, zda se robot otočí doleva nebo doprava a o kolik stupňů.

Všechny zpracované příkazy jsou následně převedeny do **Turtle Graphics**, kde každý příkaz odpovídá konkrétnímu pohybu robota. Například **[1, X]** znamená pohyb vpřed o X jednotek, **[2, X]** otočení doleva a **[3, X]** otočení doprava. Robot také může položit pero na papír (**[4, 0]**) nebo ho zvednout (**[5, 0]**). Pokud například robot dostane příkaz **L 100 100**, otočí se směrem k souřadnici (100,100), položí pero a vykreslí přímku.

Kromě jednoduchých čar kód podporuje i Beziérovy křivky, které je potřeba rozdělit na menší části. Funkce cubicBezier(P0, P1, P2, P3) provádí tento rozklad tak, že spočítá sadu bodů na křivce a převede je na úsečky, které robot dokáže nakreslit. Celý proces tedy umožňuje převést i složitější tvary do podoby, kterou micro:bit robot dokáže vykreslit přesně a plynule.

Funkce quadBezier(P0, P1, P2) zpracovává kvadratickou Beziérovu křivku pomocí jednoduššího výpočtu. Křivka je nejprve rozdělena na menší úseky, které jsou pak postupně převedeny na přímé čáry. Tyto úseky robot vykreslí pomocí příkazů Turtle Graphics, čímž se vytvoří plynulé zakřivení.

Dalším složitějším prvkem jsou eliptické oblouky, které jsou implementovány ve funkci ellipticalArc(rx, ry, xAxisRotation, largeArcFlag, sweepFlag, x, y, currentX, currentY). Tato funkce pracuje na principu maticových transformací, které umožňují vypočítat přesné body na elipse. Výstupem je seznam bodů, kterými se robot postupně pohybuje, aby vytvořil hladké křivky. Jelikož micro:bit nedokáže vykreslovat skutečné křivky, oblouk se skládá z několika krátkých přímých úseků, které dohromady vytvářejí přibližný tvar elipsy.

Výstup celého programu je seznam číselných příkazů v Turtle Graphics formátu, který micro:bit robot interpretuje jako pokyny pro pohyb. Například SVG cesta definovaná jako M 50 50 L 100 100 se převede na následující sadu příkazů:

[

[3, 45], // Otočení o 45 stupňů doprava

[1, 70], // Pohyb vpřed o 70 jednotek

[3, 45], // Otočení zpět o 45 stupňů

[1, 70] // Pohyb vpřed o 70 jednotek

]

Tento výstup říká robotovi, že se má otočit o 45 stupňů doprava, poté se posunout o 70 jednotek vpřed, následně se vrátit zpět do původního směru a znovu se posunout o stejnou vzdálenost. Tímto způsobem micro:bit vykreslí danou SVG cestu na papír.

Celý algoritmus umožňuje efektivní převod vektorových obrázků do podoby, kterou robot dokáže fyzicky vykreslit. Díky rozdělení složitých křivek na malé úseky lze přesně napodobit i složitější tvary. Tento proces probíhá v několika fázích – nejprve se analyzují příkazy SVG, poté se vypočítají správné úhly a vzdálenosti, následuje rozklad křivek na segmenty, a nakonec se vytvoří výstupní příkazy pro robota. Výsledkem je systém, který umožňuje převést digitální vektorový obrázek do skutečné kresby na papíře pomocí micro:bit robota.

# Webová aplikace

## UI/UX

Návrh webové aplikace byl vytvořen ve Figmě a slouží jako vizuální i funkční rozvržení celého systému. Hlavním cílem designu je vytvořit moderní a přehledné uživatelské rozhraní, které bude intuitivní a snadno ovladatelné. Struktura webu je rozdělena do jasně definovaných sekcí a používá konzistentní barevné schéma, aby bylo vše vizuálně sladěné.

Důležitou součástí návrhu je využití komponentového a modulárního přístupu, což znamená, že jednotlivé prvky webu lze snadno upravovat a opakovaně používat. Tento způsob usnadňuje rozšiřování webu v budoucnu a zajišťuje, že celý design zůstane jednotný. Návrh také respektuje principy UX/UI, tedy pravidla pro správnou použitelnost a vizuální přehlednost, což zajišťuje snadnou navigaci a pohodlnou práci s webovou aplikací.

## Mechaniky

### Připojení robota k webu

Jedním z klíčových prvků při práci s micro:bit robotem je jeho připojení k webové aplikaci. To je umožněno pomocí technologie Bluetooth Low Energy (BLE), která umožňuje komunikaci mezi micro:bitem a prohlížečem bez nutnosti instalace speciálních ovladačů. V této kapitole bude vysvětlen kód, který zajišťuje připojení micro:bitu k webové aplikaci, a způsob, jakým je spravováno jeho spojení a data.

Funkce handleConnect je hlavní částí kódu, která se stará o připojení micro:bitu k webové aplikaci. Tento proces probíhá asynchronně, protože samotné připojení přes BLE může trvat několik sekund. Nejprve je nutné ověřit, zda už není nějaké zařízení připojeno. Pokud ano, spojení se ukončí, zařízení se odpojí a stav aplikace se resetuje.

const handleConnect = async () => {

if (device) {

if (device.gatt && device.gatt.connected) {

await device.gatt.disconnect();

}

setDevice(null);

microbitStore.empty();

dispatch({ type: 'DISCONNECT\_CONNECT', payload: false });

return;

}

V této části kódu se kontroluje, zda je už nějaké zařízení připojeno (if (device)). Pokud ano, zkontroluje se stav připojení přes device.gatt.connected. Pokud je micro:bit připojen, pomocí device.gatt.disconnect() se odpojí a následně se proměnná device nastaví na null. Aby byla data správně spravována, funkce microbitStore.empty() vymaže uložené informace o zařízení. Také se pomocí dispatch aktualizuje stav aplikace, aby bylo jasné, že žádné zařízení není připojeno.

Pokud zařízení připojeno není, funkce pokračuje požadavkem na výběr micro:bitu.

const newDevice = await requestMicrobit(window.navigator.bluetooth);

if (newDevice) {

setDevice(newDevice);

microbitStore.update("device", newDevice);

const services = await getServices(newDevice);

microbitStore.update("services", services);

Tato část kódu umožňuje uživateli vybrat micro:bit pomocí requestMicrobit(), což je funkce, která otevře nativní dialog pro výběr dostupných BLE zařízení. Pokud je micro:bit úspěšně vybrán, uloží se do proměnné newDevice. Následně se provede aktualizace dat v microbitStore, což je objekt sloužící k ukládání informací o připojeném zařízení.

Další klíčovou částí je načtení BLE služeb:

if (services.deviceInformationService) {

const deviceInformation = await services.deviceInformationService.readDeviceInformation();

microbitStore.update("deviceInformation", deviceInformation);

}

Zde se ověřuje, zda micro:bit podporuje službu **Device Information Service**, což je jedna ze základních BLE služeb, která poskytuje informace o zařízení, jako je výrobce nebo verze firmwaru. Pokud je služba dostupná, načtou se informace o zařízení a uloží se do microbitStore.

Poslední část funkce handleConnect nastavuje událost pro případné odpojení zařízení a přesměrovává uživatele na stránku pro nahrávání dat do micro:bitu.

newDevice.addEventListener("gattserverdisconnected", handleConnect);

dispatch({ type: 'DISCONNECT\_CONNECT', payload: true });

navigate("/upload");

Při nečekaném odpojení micro:bitu (například pokud se vybije nebo se vzdálí z dosahu) je nutné spojení znovu navázat. To je zajištěno tím, že se na micro:bit nastaví událost gattserverdisconnected, která při odpojení zavolá znovu funkci handleConnect. Poslední dva řádky zajistí, že stav aplikace se aktualizuje (dispatch) a uživatel je přesměrován na stránku /upload, kde může s micro:bitem dále pracovat.

Pro uložení informací o připojeném zařízení slouží objekt microbitStore, který využívá třídu Store.

export interface MicrobitStore {

device: BluetoothDevice;

services: Services;

deviceInformation: DeviceInformation;

}

export const microbitStore = new Store<MicrobitStore>();

Vytvořením rozhraní MicrobitStore je definováno, jaká data o zařízení budou uchovávána. device obsahuje informace o BLE zařízení, services představuje seznam dostupných služeb a deviceInformation ukládá podrobnosti o micro:bitu.

Třída Store umožňuje dynamické ukládání a správu dat souvisejících s připojeným micro:bitem.

export class Store<T> {

private data: { [key: string]: T[keyof T] | undefined } = {};

private listeners: Partial<T>[] = [];

private \_update<K extends keyof T>(key: K, value: T[K] | undefined) {

this.data[key as string] = value;

this.listeners.forEach(listener => {

if (listener[key] !== undefined) {

listener[key] = value;

}

});

}

Data jsou ukládána do objektu data, kde je každá hodnota přístupná podle svého klíče. Funkce \_update umožňuje změnu hodnoty v data a zároveň informuje všechny komponenty, které dané hodnoty sledují.

Kromě ukládání nových dat umožňuje třída Store i přidání posluchačů, kteří jsou upozorněni na změny hodnot.

public addListener(listener: Partial<T>) {

this.listeners.push(listener);

Object.keys(this.data).forEach(key => {

if (listener[key as keyof T] !== undefined) {

listener[key as keyof T] = this.data[key];

}

});

}

Díky této funkci je možné zajistit, že když dojde ke změně například připojeného zařízení, všechny součásti aplikace, které na této informaci závisí, budou automaticky aktualizovány.

Pro vymazání všech uložených dat je v třídě Store definována funkce empty:

public empty() {

Object.keys(this.data).forEach(key => {

this.\_update(key as keyof T, undefined);

});

}

Tato funkce je klíčová při odpojení microbitu, protože zajistí, že v aplikaci nebudou zůstávat zastaralé informace o dříve připojeném zařízení.

### Protokol komunikace

Po úspěšném připojení Micro:bitu k webové aplikaci je dalším důležitým krokem odesílání dat do zařízení. To umožňuje předat Micro:bitu instrukce k vykreslení určitého tvaru na základě SVG příkazů, které jsou přeloženy do Turtle Graphics. Kód pro odesílání dat je implementován v asynchronní funkci sendout, která přijímá pole příkazů typu OutputCommand[] a následně je odesílá do Micro:bitu prostřednictvím služby **UART**.

Na začátku funkce se získá reference na službu uartService, která je součástí microbitStore a umožňuje komunikaci se zařízením přes **Bluetooth Low Energy (BLE)**. Pokud je služba k dispozici, začne se s přípravou dat. Data jsou reprezentována jako pole SVG příkazů, které jsou nejprve upraveny do formátu vhodného pro použití:

let textOut = '[';

input.forEach((x) => (textOut += `[${x[0]}, ${x[1]}],`));

textOut += '[4,0]]';

console.log("text out", textOut);

Tento kód vytvoří řetězec odpovídající seznamu příkazů, kde každý příkaz je uzavřen v hranatých závorkách. Poslední příkaz [4,0] je přidán automaticky a slouží k signalizaci konce přenosu dat.

Jakmile Micro:bit obdrží data, může poslat zpětnou vazbu, která informuje aplikaci o průběhu zpracování. K tomu slouží následující část kódu, která naslouchá událostem přijatým z Micro:bitu:

service.uartService.addEventListener("receiveText", (e: CustomEvent) => {

const { detail } = e;

if (detail.startsWith("%dr")) {

const x = (parseInt(detail.slice(3)) / (input.length + 2)) \* 100;

console.log(`Drawing ${Math.round(x)}`);

setProgress(x);

}

switch (detail) {

case "%rsta":

console.log("Uploading");

break;

case "%rend":

console.log("Uploaded");

break;

case "%dsta":

console.log("Drawing");

setProgress(0);

break;

case "%dend":

console.log("Finished");

setProgress(100);

break;

}

});

Micro:bit může poslat různé odpovědi označené speciálními kódy:

* %rsta – Začátek nahrávání dat do Micro:bitu.
* %rend – Úspěšné dokončení nahrávání.
* %dsta – Začátek kreslení podle přijatých dat.
* %dend – Dokončení kreslení.

Pokud Micro:bit pošle zprávu začínající %dr, znamená to, že probíhá kreslení a aplikace může vypočítat, jaká část obrázku už byla vykreslena. Tento údaj se následně použije k aktualizaci ukazatele průběhu (setProgress(x)).

Po přípravě dat je třeba je odeslat do Micro:bitu. K tomu slouží smyčka, která postupně posílá jednotlivé části zprávy.

const sendText = '%' + textOut + '%';

console.log("send text", sendText);

for (let i = 0; i < sendText.length; i += 14) {

console.log("sending to microbit", sendText.slice(i, i + 14) + '$');

await service.uartService.sendText(sendText.slice(i, i + 14) + '$');

}

Každá zpráva začíná a končí znakem %, což slouží jako označení začátku a konce přenosu. Protože BLE přenosy mají omezenou velikost datového paketu, je zpráva rozdělena na části po **14 znacích**. Každá část je odeslána pomocí service.uartService.sendText(), přičemž se na konec přidává znak $, který pomáhá Micro:bitu správně rozpoznat jednotlivé části přenosu. Pokud služba uartService není dostupná, vypíše se do konzole chybová hláška, která pomáhá identifikovat situaci, kdy Micro:bit není připojen nebo není správně inicializován.

### Zobrazení cesty robota v reálném čase

Bude doděláno brzy

### Editor SVG souborů

SVG editor, převzat z existujícího řešení, umožňuje uživatelům vytvářet a upravovat vektorovou grafiku přímo v prohlížeči. Je postaven na knihovně **Fabric.js**, která zajišťuje manipulaci s objekty na plátně.

Hlavní komponenta CanvasApp spravuje plátno pomocí React hooků useState a useRef. Po načtení aplikace se inicializuje plátno o velikosti **500x500 pixelů** a na něj nahraje SVG soubor, se kterým se momentálně pracuje. Editor ho načte a vykreslí pomocí fabric.loadSVGFromString, což umožňuje jeho další úpravy.

Editor obsahuje podporu pro **pohyb objektů**, při kterém se zarovnávají pomocí vodicích čar (handleObjectMoving). Po dokončení úpravy se vodicí čáry odstraní (clearGuidelines). Nechybí ani funkce **exportu do SVG**, která převede obsah plátna na XML formát a umožní stažení výsledného souboru.

Uživatelé mohou přidávat nové prvky (AddElements), jako například křivky nebo jiné základní tvary, měnit několik základních vlastností (Settings) anebo měnit rozměry plátna (CanvasSettings). Díky propojení s Reactem je editor snadno integrovatelný do webových aplikací a umožňuje základní práci se SVG bez potřeby externího softwaru.

# Problémy

## Bluetooth připojení

Jedním z největších problémů při implementaci bylo navázání spojení mezi webovou aplikací a micro:bitem pomocí technologie Bluetooth Low Energy (BLE). I přes opakované úpravy kódu a kontrolu nastavení se nedařilo připojení stabilně navázat, což komplikovalo testování i další vývoj aplikace.

První krok při řešení problému spočíval v kontrole správné inicializace Bluetooth adaptéru v prohlížeči a ověření, zda micro:bit vysílá BLE signál a je v režimu párování. Pro testování byly použity různé prohlížeče, především Google Chrome a Microsoft Edge, protože tyto prohlížeče mají nativní podporu Web Bluetooth API. Přestože micro:bit byl aplikací rozpoznán, připojení se buď ihned přerušilo, nebo vůbec neproběhlo.

Další analýza naznačila, že problém může souviset s omezeními Web Bluetooth API, která se mohou lišit podle verze prohlížeče a operačního systému. Ukázalo se, že některé systémy vyžadují speciální oprávnění k přístupu k BLE zařízením, což mohlo být jedním z důvodů nefunkčního připojení. Kromě toho byla provedena kontrola firmwaru micro:bitu, protože správná verze softwaru je klíčová pro podporu BLE komunikace.

Nakonec se problém podařilo vyřešit díky doporučení vedoucího práce, který navrhl drobné úpravy v kódu a konfiguraci micro:bitu. Po těchto změnách a dodatečném ladění BLE komunikace se podařilo připojení úspěšně navázat. Tento problém ukázal, že práce s BLE technologií může být složitá a vyžaduje přesné nastavení jak softwaru, tak samotného zařízení. Kromě správné inicializace je nutné brát v úvahu také verzi prohlížeče, oprávnění systému a kompatibilitu firmwaru.

## Editor SVG

Při implementaci editoru SVG souborů se objevil zásadní problém, protože dostupná řešení byla velmi omezená a většina z nich nevyhovovala požadavkům na integraci do webové aplikace. Výběr vhodného editoru provázelo několik komplikací, které ztěžovaly jeho přímé využití v projektu.

Jednou z hlavních překážek byla nekompatibilita většiny dostupných editorů s Reactem, který byl použit pro vývoj aplikace. Mnoho existujících SVG editorů bylo navrženo jiným způsobem, což znamenalo, že je nebylo snadné přizpůsobit pro použití jako React komponenty. Často chyběla podpora pro správu stavu aplikace, což způsobovalo problémy při aktualizaci a synchronizaci SVG souborů s uživatelským rozhraním. Některé editory navíc neumožňovaly okamžité aktualizace, což negativně ovlivňovalo plynulost práce a uživatelskou přívětivost.

Další komplikací byla složitá integrace existujících řešení do webové aplikace. I když některé open-source SVG editory nabízely široké možnosti úprav, jejich začlenění do projektu by vyžadovalo rozsáhlé změny kódu, což by mohlo narušit celou architekturu aplikace. Mnoho těchto editorů nebylo modulárních, takže nešlo jednoduše přidat potřebné funkce nebo je efektivně upravovat.

Také stav dostupných editorů nebyl ideální. Některé z nich byly zastaralé, nepodporovaly moderní technologie a jejich užitečnost byla omezená. Jiné sice nabízely pokročilé funkce, ale chyběla jim kvalitní dokumentace nebo už nebyly aktivně vyvíjeny, což ztěžovalo jejich údržbu a přizpůsobení potřebám aplikace.

Po zvážení všech těchto problémů se jako nejlepší řešení ukázalo vytvořit vlastní SVG editor. Tento přístup umožnil mít plnou kontrolu nad jeho implementací a upravit ho přesně podle potřeb projektu. Základem se stal online návod, který poskytl základní strukturu editoru, ale bylo nutné ho dále upravovat a optimalizovat, aby byl plně funkční. Výsledný editor byl navržen tak, aby byl kompatibilní s Reactem, umožňoval plynulé zpracování SVG souborů a snadno se integroval do celého systému aplikace.

Celý proces ukázal, že hotová řešení nejsou vždy nejlepším řešením a někdy je efektivnější vytvořit vlastní nástroj, který přesně odpovídá požadavkům projektu. Tento přístup umožnil nejen lepší kontrolu nad výsledkem, ale také zajistil budoucí rozšiřitelnost a flexibilitu, což usnadní další vývoj aplikace.

Závěr

Vytvoření webu bylo ve výsledku úspěšné, i přes mnohé strasti a komplikace.

Seznam zkratek a odborných výrazů

HTML

HyperText Markup Language – značkovací jazyk používaný pro tvorbu webových stránek.

React

Seznam obrázků

Použité zdroje

1. **Stehlík, Michal.** *Návod k maturitním pracím 2020.* Liberec : Albatros, 2020.

1. Seznam přiložených souborů

Na přiloženém datovém nosiči se nacházejí následující soubory a složky:

* **MP2010-Novák-Jan-L4-Tepelné\_čerpadlo.docx** – editovatelná verze dokumentace maturitní práce
* **MP2010-Novák-Jan-L4-Tepelné\_čerpadlo.pdf** – tisknutelná verze dokumentace maturitní práce
* **Výkresy** – kompletní výkresová dokumentace
* **Aplikace** – zdrojové kódy