FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA INFORMATIKY
A VÝPOČETNÍ TECHNIKY

Bakalářská práce

Webové rozhraní pro překladač jazyka Blason

František Urban

PLZEŇ

2024

FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA INFORMATIKY
A VÝPOČETNÍ TECHNIKY

Bakalářská práce

Webové rozhraní pro překladač jazyka Blason

František Urban

Vedoucí práce
Ing. Richard Lipka, Ph.D.

PLZEŇ 2024

Citace v seznamu literatury:
URBAN, František. *Webové rozhraní pro překladač jazyka Blason*. Plzeň, 2024. Bakalářská práce. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky. Vedoucí práce Ing. Richard Lipka, Ph.D.

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**František URBAN**
Osobní číslo:	**A21B0305P**
Studijní program:	**B0613A140015 Informatika a výpočetní technika**
Specializace:	**Informatika**
Téma práce:	**Webové rozhraní pro překladač jazyka Blason**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se s principy a fungováním jazyka Blason pro český jazyk.
2. Seznamte se s nástrojem pro analýzu Blasonu vzniklým na KIV.
3. Navrhněte webové rozhraní, které umožní zadat popis v Blasonu a zobrazí vytvořené erbovní znamení.
4. Navržené řešení implementujte.
5. Hotové řešení otestujte s ohledem na přesnost, úplnost a uživatelskou přístupnost.

Rozsah bakalářské práce: **doporuč. 30 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

Dodá vedoucí bakalářské práce.

Vedoucí bakalářské práce: **Ing. Richard Lipka, Ph.D.**
Katedra informatiky a výpočetní techniky

Datum zadání bakalářské práce: **2. října 2023**
Termín odevzdání bakalářské práce: **2. května 2024**

L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 25. října 2023

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného akademického titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Západočeská univerzita v Plzni má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Plzni dne 1. května 2024

. . . . . .
František Urban

V textu jsou použity názvy produktů, technologií, služeb, aplikací, společností apod., které mohou být ochrannými známkami nebo registrovanými ochrannými známkami příslušných vlastníků.

Abstrakt

Heraldika umožňuje popisovat erbovní znamení formálním jazykem blazon. Tato bakalářská práce se zabývá vytvořením uživatelsky přívětivé webové aplikace pro generování vizualizace erbovních znamení na základě vstupního textu blazon. Aplikace využívá existující nástroj CBlazon, který byl vyvinut na Západočeské univerzitě a umožňuje překlad blazonu do strukturované textové podoby. Navržené řešení umožňuje generovat vizualizace znaků s minimální zátěží webového serveru, jelikož procesy generování probíhají na straně klienta. Výsledná aplikace umožňuje generovat jednoduché znaky popsané jazykem blazon.

Abstract

Heraldry makes it possible to describe armorial bearings in the formal language of blazon. This bachelor thesis deals with the creation of a user-friendly web application for generating a visualization of armorial signs based on the input blazon text. The application uses an existing tool CBlazon, which was developed at the University of West Bohemia and allows the translation of blazon into a structured textual form. The proposed solution allows generating sign visualizations with minimal load on the web server, as the generation processes are performed on the client side. The resulting application allows the generation of simple characters described by the blazon language.

Klíčová slova

blazon • generátor znaků • React • WebAssembly

Obsah

Úvod

Součástí současné lidské společnosti jsou různé symboly, reklamní plakáty, loga nebo třeba dopravní značky. Původ těchto grafických projevů vychází z heraldiky. Tato věda se zabývá studiem a souhrnem pravidel erbů, znaků a štítů. Jakýkoliv heraldický erb je možné popsat formálním slovním popisem erbu nazývaným *blazon*, někdy také *blason*.

V rámci své diplomové práce se autor Oto Šťáva zabýval vývojem nástroje, který umožňuje zpracovat slovní popis erbu, takzvaný *blazon*, v českém jazyce na strukturovaný datový popis erbu. Tento výstup je následně určen pro další zpracování. Autor nástroj implementoval v programovacím jazyce C a nazval jej *CBlazon*.

Cílem této bakalářské práce je navázat na práci sady programů CBlazon a využít její výstup pro vygenerování grafické podoby heraldických erbů. V rámci práce bude vytvořeno uživatelsky přívětivé grafické prostředí, které bude integrovat nástroje programů CBlazon. S ohledem na přístupnost je požadavkem, aby se jednalo o webovou aplikaci.

Automatizovaný generátor grafických znaků může být vhodný například ve školách, kde se vyučují základy heraldiky, nebo jako ukázka využití formálního překladače i jinde než v programovacích jazycích.

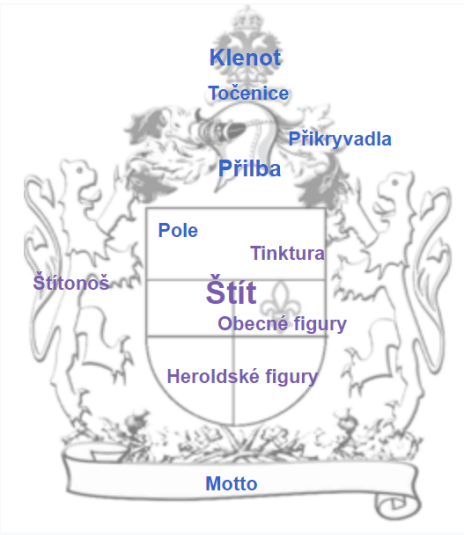
Blazon 1

Blazon je slovní popis znaku podle odborné terminologie a heraldických pravidel [1, s. 26]. Věda zabývající se studiem jejich pravidel a zvyklostí, se nazývá *heraldika*. Slovní popis znaku je dostatečně ustálený, aby kterýkoliv heraldik byl schopen z něj vytvořit grafickou reprezentaci [1].

Blazon se skládá z postupného popisu jednotlivých částí znaku, mezi které patří dělení znaku, použité „barvy“ (nazývají se tinktury), figury, přilba nebo například další vnitřní štítky . Systematický přístup umožňuje jasnou interpretaci a reprodukci znaku v grafické podobě [1].

Termín „znak“ a „erb“ se často zaměňuje. Erb se na rozdíl od znaku dědí v rodinném rodu. Erb se proto používá v souvislosti se šlechtickými rody, zatímco znak je spojen s městy a znaky států [1, s. 113, 451].

Znak se vždy musí skládat ze štítu, přičemž může obsahovat dodatečné prvky jako přilbu, klenot, přikrývky nebo točenici [1, s. 14]. Tyto dodatečné prvky se obvykle nevyskytují ve znacích, naopak jsou hojně využívány v erbech [1, s. 14].



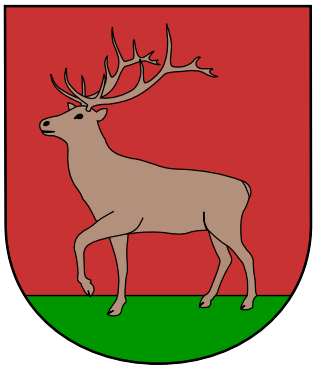
Obrázek 1.1: Heraldické části erbu[1]

Osoba, která znak drží v ruce, se nazývá *štítonoš* [1, s. 25]. Ten může, ale nemusí být součástí erbu. V jazyce *blazon* se levá a pravá strana posuzuje z pohledu štítonoše. Díváme-li se na nakreslený erb z pohledu pozorovatele je levá a pravá strana otočená [1, s. 16]. Tato terminologie byla chápána jako zdvořilost vůči nositeli erbu, který má přednost před pozorovatelem [1, s. 16].

1.1 Příklad

Pro představu uvedu příklad blazon znaku města Letohrad. Na tomto blazonu ukážu jednotlivé výstupy programů, které budou v této práci popsány.

> Blazon štítu města Letohrad:
> *V červeném štítě s trávníkem vykračující jelen přirozených barev*



Obrázek 1.2: Znak města Letohrad [2]

1.2 Tinktury

V heraldice se jednotlivé „barvy“ nazývají tinktury, ty se dělí na barvy, kovy a kožešiny [1, s. 16]. Heraldika je celosvětovým oborem, kde je rozeznáváno mnoho různých tinktur. V této části vycházím z publikace [1], kde jsou vyjmenovány používané tinktury na původním českém území. Základním pravidlem heraldiky je, že barva nesmí přijít na barvu [1, s. 16]. Mezi používané kovy patří zlato a stříbro. Původními základními barvami jsou jen červená, modrá, černá a zelená [1, s. 58]. V průběhu času počet barev přibýval, podle publikace [1, s. 58] jich je dohromady celkem deset. Jednou z nich je takzvaná *přirozená barva* [1, s. 58], která označuje přirozené zbarvení zvířete, rostliny, lidské pokožky a podobně. Používanými kožešinami jsou hermelín, kunina, sobol a popelčina [1, s. 16].

[1] zdroj (veřejná doména): `https://commons.wikimedia.org/wiki/File:Coat-elements.png`

1.3 Štít

Štít je hlavní částí erbu. Tvar štítu se nikdy v blazonu nepopisuje a závisí na umělci, jaký tvar zvolí [1, s. 17]. Zvolený tvar štítu by však měl respektovat umělecký sloh, například gotický nebo renesanční, a provincii, jako například italskou nebo španělskou [1, s. 17].

1.4 Figury

Obrazce, které se vkládají do štítu, se nazývají erbovním nebo znakovým znamením a nebo jen figurou [1, s. 18]. Figury se dělí na dvě skupiny: *heroldské* a *obecné*.

Heroldské figury jsou obrazce tvořené čárami a geometrickými obrazci. Jedná se o způsob, jak dělit jednotlivé pole. Mezi základní způsoby dělení štítu patří: polcení, dělení, klín, hlava štítu a mnoho dalších. Jednotlivé heroldské figury je navíc možné vzájemně kombinovat a vnořovat do sebe [1, s. 18-20]. Vzhledem k tomu, že ve své bakalářské práci vykresluji jednotlivé figury, zjišťoval jsem jaké jsou poměry stran geometrických obrazců. Autor Buben [1, s. 19] ve své publikaci vysvětluje, že v heraldice se již mnoho autorů snažilo vymezit šíře a velikosti heraldických figur. Nikomu se však nepovedlo rozměry globálně prosadit. Autor to odůvodňuje tím, že vzhledem k tomu, že lze umisťovat další figury do jednotlivých heroldických figur, musí umělci někdy například zvětšit výšku břevna, aby mohla být vnitřní figura zřetelně rozlišena [1, s. 18-20].

Stylizace dělící čáry nemusí být jen běžná čára, ale může být například vlnitá, pilová nebo cimburovitá [1, s. 18-20].

Obecné figury jsou všechna ostatní erbovní znamení. Na rozdíl od heroldských figur se nesmějí dotýkat okrajů štítu [1, s. 20-22]. Obecné figury se dále dělí podle toho, zda jsou živé, nadpřirozené nebo umělé.Tato rozdělení se dále dělí, až se nakonec dostaneme k samotné figuře. Konečné ztvárnění figury mohou různí autoři různé národnosti stylizovat odlišně [1, s. 20-22]. Každá kategorie má své možnosti, živá zvířata mohou mít různou polohu, například lev „ve skoku“, „kráčející“ nebo „sedící“. V případě, že poloha zvířete není uvedena, je zvyklostí, že zvíře hledí a kráčí doprava [1, s. 20-22].

CBlazon 2

Programy sady CBlazon umožňují převést český text popisu erbu (neboli blazon) do strukturovaného formátu. Celá práce byla vyvinuta autorem Otem Šťávou v rámci jeho diplomové práce na Fakultě Aplikovaných věd, Západočeské univerzity v Plzni [3]. Následující část je sepsána na základě textu jeho diplomové práce [3].

Architektura programu funguje na základě formálního překladače s využitím lemmatizace textu. Lemmatizace umožňuje určení druhu slova a jeho základního tvaru. Konkrétní lemmatizátor použitý v těchto programech je *Morphological Dictionary and Tagger* neboli zkráceně *MorphoDiTa* [4].

Sada CBlazon obsahuje čtyři spustitelné soubory. Pro účely bakalářské práce jsou zajímavé především dva z nich `cbltag` a `cblhtml`.

Program `cbltag` je hlavním nástrojem sady CBlazon. Jedná se o konzolovou aplikaci, která přijímá jako vstup text ve formátu blazon. Umožňuje generování různých výstupů, přičemž jeden z hlavních je *Abstract syntax tree*.

Program `cblhtml` je také konzolová aplikace, jejímž vstupem je opět text ve formátu blazon. Výstupem je soubor ve formátu `.html`, obsahující vizualizaci derivačního stromu, který je barevně zvýrazněn. V případě, že program nedokáže zpracovat nějakou část blazonu, je tato část označena podtržením. Soubor dále obsahuje jednoduchou vizualizaci dělení štítu ve formátu *Scalable Vector Graphics* (zkráceně *SVG*).

Autorovi se podařilo přeložit kód programu do formátu *WebAssembly*, čímž může být program spuštěn ve webové aplikaci na straně klienta bez zátěže serveru. Sada CBlazon však neobsahuje žádný spustitelný program určený přímo pro WebAssembly.

2.1 Výstupy programu

Struktura knihovny CBlazon je natolik univerzální, že umožňuje měnit výstupní formát bez nutnosti změny jádra aplikace. V následující sekci popíši výstupy spustitelných programů a pokusím se nalézt nejvhodnější formát pro mojí aplikaci.

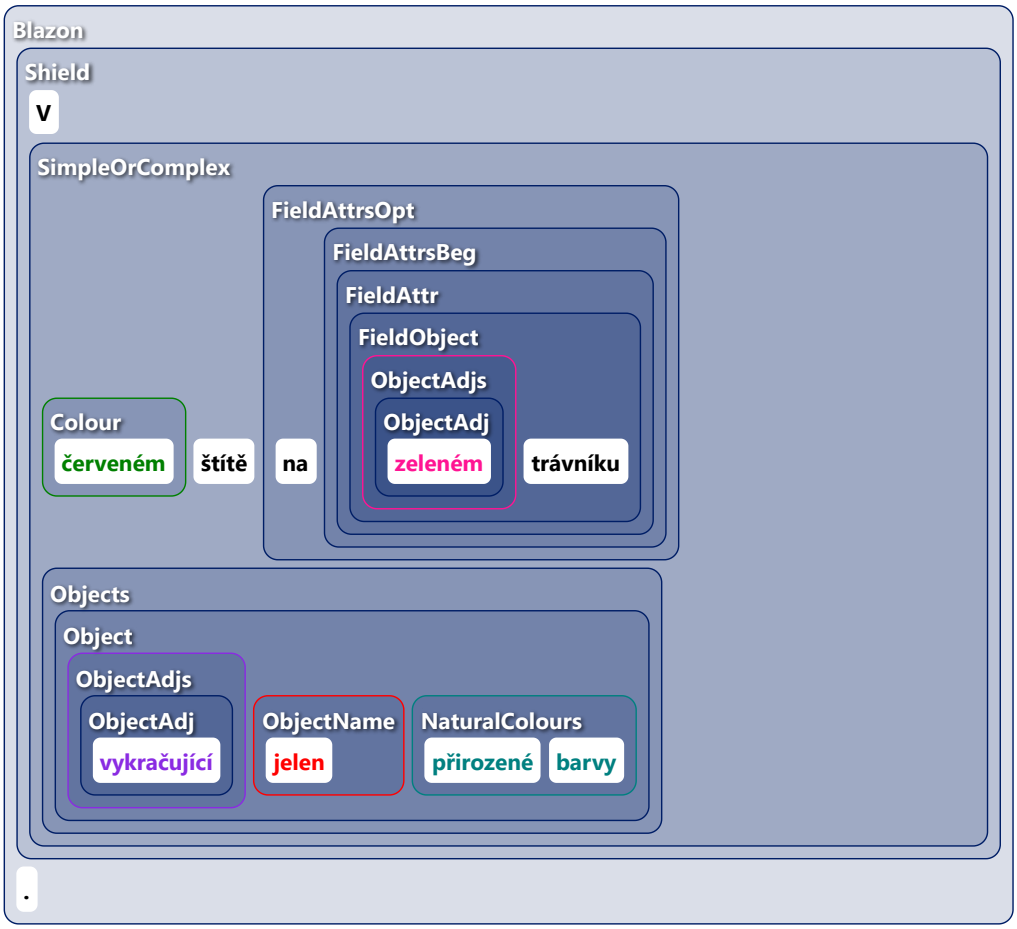
Program `cbltag` umožňuje pomocí přepínačů v argumentech vytvářet různé výstupy. Prvním přepínačem je `tag`, který umožňuje vypsání výsledku lemmatizace textu v textové podobě. Tento výstup je určený spíše pro testovací účely než pro další zpracování. Druhý přepínač `tokenize` funguje podobně, ale navíc obsahuje druh slova, pokud se jedná o slovo číslovky převede ji na číslo. Zajímavější je přepínač `tree`, který vypisuje strukturu derivačního stromu, ukazující, jak byl vstupní text zpracován analyzátorem. Tento výstup obsahuje mnoho informací, a připomíná standardní formát XML. I když se nejedná o validní XML, není problém upravit program `cbltag` tak, aby generoval standardní formát XML. Posledním přepínačem programu je `ast` výstup je ve formátu *Abstract syntax tree*, česky *syntaktický strom*, který připomíná formát JSON. Následující výstupní formát je pro moji aplikaci nejvhodnější.

Výpis 2.1: Výstup Abstract syntax tree programu cbltag

```
cbl_blazon {
  shield = cbl_obj_shield {
    splits = cbl_split_tree {
      > root = cbl_split {
        type = UNSPLIT
        style =
        where = CENTRE
        field_ix = 0
      }
    }
    fields = [
    cbl_field {
      colour = červený
      objects = [
      cbl_obj_figure {
        name = [ "trávník" ]
        tag_attrs = [ (empty) ]
        obj_attrs = [ (empty) ]
        natural_colours = false
      },
      cbl_obj_figure {
        name = [ "jelen" ]
        tag_attrs = [ "vykračující" ]
        obj_attrs = [ (empty) ]
        natural_colours = true
      },
      ]
    },
    ]
  }
  bearer = (null)
  helmet = (null)
```

```
33   valid = true
34 }
```

Program `cblhtml` umožňuje ze vstupního textu blazon generovat soubor `.html`, který obsahuje grafickou vizualizaci derivačního stromu a erbu. Vizualizace derivačního erbu mě zaujala, protože se jedná o kompletní výstup, který lze bez dodatečných úprav použít pro moji aplikaci. Grafická vizualizace znaku nastínila základní proces zpracování výstupu programu.



Obrázek 2.1: Grafická vizualizace derivačního stromu vytvořená v programu cblhtml

Návrh a architektura systému — 3

Cílem návrhu aplikace je zvolit základní technologie pro webovou aplikaci, která bude zpracovávat výstup programu ze sady CBlazon. Výsledkem běhu aplikace bude grafická vizualizace heraldického znaku.

3.1 Požadavky

Požadavky aplikace byly stanoveny vedoucím bakalářské práce. Hlavním požadavkem bylo, aby se jednalo o webovou aplikaci. Aplikace má být nasazena na školních serverech a architektura by měla co nejméně zatěžovat server. Předpokládá se nasazení aplikace na *Apache HTTP Serveru*. Pro aplikaci je možné použít školní databázový server. Z bezpečnostního hlediska nebyly na aplikaci kladeny žádné nároky, především proto, že by měla využívat volně dostupné materiály a s daty manipulovat lokálně na klientovi.

3.2 Architektura

V následující sekci popíšu jaké části bude má práce obsahovat a co se bude vykonávat na straně klienta a co na serveru.

Jednou z prvních věcí, kterou bylo potřeba vyjasnit, bylo, kde a jak se bude sada programů CBlazon spouštět. Logickým krokem bylo použití technologie *WebAssembly*, pro kterou byla již sada programů CBlazon částečně připravena. Tento krok byl vhodný vzhledem k požadavku na nízké nároky na serverové prostředky. Možností by bylo samozřejmě spouštět programy napsané v programovacím jazyce C na serveru, ale projekt CBlazon je open-source, a navíc je požadavkem, aby server byl co nejméně zatížen. Pro překlad programovacího jazyka C do WebAssembly jsem použil překladač *Emscripten* kvůli jeho rozšířenosti a podpoře.

Pro vytvoření vizualizace znaků jsou potřebná SVG data figur, která je třeba někde uchovávat. K tomuto účelu by mohl být vhodný databázový server s backendovou implementací a funkcemi CRUD na straně serveru. V případě této aplikace,

to ale nemusí být potřeba, protože jednotliví uživatelé s daty aplikace nemanipulují. Některá data navíc není nutné ukládat do databáze, neboť jejich počet není tak velký, aby si klient nemohl stáhnout všechny jako součást konstantních proměnných. Těmito daty jsou tinktury, heroldické figury a tvary štítu.

Jediná manipulace s daty ze strany uživatele by mohla být v přidávání nových obecných figur. Aby byla tato možnost použitelná, musela by klientská aplikace implementovat grafické rozhraní, ve kterém by bylo možné doplnit veškerá potřebná data k dané figuře, což jsem z časových důvodů nebyl schopen v dostatečné kvalitě implementovat. Proto jsem se rozhodl nepoužít databázi a vytvořit pouze aplikace klienta.

Jednotlivé figury ukládám na serveru roztříděné ve složkách. Ze serveru je stahuju podle popisů, které budou uloženy jako proměnné konstanty ve formátu JSON.

3.3 Zvolené Technologie

Ve webových aplikacích se na straně klienta používají technologie HTML, CSS a JavaScript. V následující sekci vyberu vhodné technologie pro aplikaci.

Pro zvýšení udržitelnosti kódu jsem se rozhodl využít programovací jazyk *TypeScript* [5], který se následně překládá do JavaScriptu. *TypeScript* je staticky typovaný jazyk, což znamená, že umožňuje odhalit mnoho chyb již během překladu kódu.

Klientská webová aplikace by měla být interaktivní, proto je vhodné použít knihovnu, která umožňuje interaktivitu. Jednou z takových knihoven je *React* [6], vyvíjená společností *Meta*. Tato knihovna umožňuje vytvářet "single-page"(zkráceně SPA) webové stránky, kde obsahuje pouze jeden `.html` soubor a interaktivita je dosažena změnou Document Object Model (zkráceně DOM), za běhu aplikace. React nezahrnuje základní komponenty, jako jsou například tlačítka, boxy, nebo gridy. Proto je vhodné zvolit knihovnu pro React, která tyto komponenty obsahuje. Jednou z takových knihoven je *Material UI* [1], která poskytuje již navržené grafické komponenty. Tato knihovna je vyvíjena společností *Google*, která ji používá ve svých aplikacích, což zaručuje dlouhodobou podporu i v budoucnu.

Instalaci balíčků a jejich verzí jsem spravoval pomocí nástroje *yarn*. Ze začátku jsem používal *npm*, který instaluje balíčky sekvenčně a je tudíž pomalejší než *yarn*, který je stahuje paralelně.

Heraldické znaky budou vytvářeny ve značkovacím vektorovém formátu *Scalable Vector Graphics*, zkráceně *SVG*. Grafická vizualizace se v tomto formátu vytváří pomocí elementů, které jsou zapsány ve formátu XML. Pro manipulaci s těmito elementy je vhodné zvolit dostatečně škálovatelnou knihovnu. Vhodnými kandidáty se nabízejí následující knihovny: Svg.js, Paper.js, d3.js a mnoho dalších. Nakonec jsem

[1] MUI Material UI: `https://mui.com/material-ui/`

zvolil knihovnu d3.js kvůli její univerzálnosti, rozšířenosti a dlouhodobé podpoře. Knihovna d3.js ale, na rozdíl od ostatních, neobsahuje nástroje pro práci s vektory, body a podobně. Na rozdíl od ostatních, tato knihovna umožňuje manipulovat s SVG elementy libovolně, aniž by musela implementovat konkrétní klíčová slova elementů a atributů. Ve své práci předpokládám použití více balíčků, tento byl ale jedním z nejdůležitějších.

3.4 Zdroje dat

Během analýzy jazyka blazon jsem objevil různé zdroje, ze kterých je možné sbírat testovací data a data pro databázi figur. Aby jednotlivá státní zřízení mohla chránit autorskými právy znaky měst a obcí, shromažďují státy obecní symboly. Česká republika není výjimkou a Poslanecká sněmovna Parlamentu České republiky k tomuto účelu vyhradila systém *Registr komunálních symbolů*, zkráceně *REKOS* [2]. Portál REKOS obsahuje znaky a vlajky měst a obcí České republiky. Součástí je textový popis blazonu a grafická vizualizace. Obrázky jsou ale bohužel v rastrovém formátu. Doplnění znaků do portálu REKOS je ze strany obcí a měst dobrovolné, i přesto ale obsahuje většinu znaků.

Figury, které budou použity pro vizualizaci, je nutné získat v nejlepší kvalitě, ideálně ve vektorovém formátu. Nejvhodnější variantou se zdají být webové stránky Wikipedie a jejich projekt úložiště otevřených obrázků *Wikimedia Commons*. *Wikimedia Commons* [7] se od roku 2009 zabývá problematikou heraldiky. Stanovilo, jak by se měly znaky pojmenovávat, a pokud je to možné, používá se formát vektorového formátu `.svg`. Každý obrázek navíc obsahuje licenci.

Použité technologie 4

4.1 Scalable Vector Graphics

Scalable Vector Graphics [8] (zkráceně SVG) je jazyk ve formátu XML, který popisuje dvourozměrnou vektorovou grafiku a používá příponu souborů `.svg`. Může být používán například pro vytvoření čar, cest a mnoho dalšího. Jednotlivá primitiva nebo skupiny primitiv lze přesouvat, otáčet, slučovat nebo škálovat. Formát byl vyvinut organizací spravující standardy webových technologií W3C. SVG je podporované všemi nejpoužívanějšími webovými prohlížeči. Elementy SVG lze vkládat přímo do HTML elementů a webový prohlížeč je bez problému zobrazí.

SVG dokument může využívat stylizaci pomocí kaskádových stylů, anglicky *Cascading Style Sheets* (zkr. CSS). Styly je možné používat v různých formách: inline formě, která se píše do elementu SVG jako attribut, globálního elementu `<style>` anebo externího CSS souboru. Kaskádové styly v SVG sice neumožňují stejné množství funkcí jako v HTML, ve své práci jsem se však s žádným omezením nesetkal [8].

```
<svg width="300" height="150"
  xmlns="http://www.w3.org/2000/svg">
  <style>
    .blueRect {
      width: 200px;
      height: 100px;
      x: 10;
      y: 10;
      rx: 20px;
      ry: 20px;
      fill: blue;
    }
  </style>

  <rect class="blueRect" />
</svg>
```

Zdrojový kód 4.1: Příklad použití stylu globálního elementu <style> v elementu SVG

Nakonec bych chtěl zmínit programy, ve kterých je možné SVG editovat a které jsem ve své práci využil. Program, který jsem použil na editaci figur je *Inkscape* [1], jedná se o open-source vektorový editor. V případě vytváření a editaci cest, kde jsem chtěl mít kontrolu na tím, jak bude cesta vypadat, se mi nejvíce osvědčil následující online webový nástroj *svg-path-editor* [2].

4.2 React

React [9] je JavaScript knihovna, která slouží pro vytváření webových grafických rozhraní. Knihovnu vyvinula společnost Meta a vydala projekt pod licencí open-source. Hlavním rozdílem Reactu oproti tradičním webovým stránkám, které jsou vytvářeny na serveru a posílány klientovi, je to, že webová stránka vytvořená pomocí Reactu je vykreslována přímo u klienta [10]. To přináší vylepšenou uživatelskou interaktivitu a rychlost odezvy [10].

Zdrojové kódy je možné v Reactu zapisovat pomocí rozšířené syntaxe JSX, která umožňuje psát kód tak, že kombinuje JavaScript s HTML značkami. Soubory JSX není možné použít ve webovém prohlížeči stejně jako soubory v syntaxi JavaScript. Před použitím je potřeba je překompilovat na JavaScript soubory. Syntaxe JSX se běžně ukládá do souborů s příponou `.jsx` nebo `.tsx` v případě použití TypeScriptu.

4.2.1 Webové nástroje

V následující sekci se budu věnovat nástrojům, které nejsou součástí Reactu, ale jsou s ním obvykle používány.

Pro správu balíčků v projektu React se často využívají správci balíčků jako *yarn* [3] a *npm* [4]. S nimi je možné instalovat a aktualizovat balíčky knihoven, a také jednoduše vytvářet nové projekty React.

Bundler, jako například *Webpack* [5], je nástroj používaný při vývoji webových aplikací. Jeho hlavním účelem je zpracování různých typů souborů, jako jsou JavaScript, CSS, obrázky a další, a jejich sloučení do jednoho nebo více balíčků pro efektivnější načítání v prohlížeči.

Babel [6] je překladač, transformuje zdrojový kód napsaný v novějších verzích JavaScriptu do formátu, který je kompatibilní s prohlížeči. V kontextu projektu React psaného v TypeScriptu se Babel používá k převodu kódu z TypeScriptu do standardního JavaScriptu, což umožňuje spouštění aplikací v prohlížečích. Babel také

[1] Inkscape: `https://inkscape.org`
[2] SvgPathEditor: `https://yqnn.github.io/svg-path-editor`
[3] Yarn: `https://yarnpkg.com/`
[4] npm: `https://www.npmjs.com/`
[5] Webpack: `https://webpack.js.org/`
[6] Babel: `https://babeljs.io/`

zpracovává JSX syntaxi, která je používána pro vytváření uživatelského rozhraní v React aplikacích.

4.2.2 Komponenty

Komponenty v Reactu jsou znovupoužitelné části kódu, které je možné vložit do elementů. Vlastní komponenty je možné vytvořit buď jako třídu nebo funkci. Podle dokumentace React je preferované použití funkcí.

```
import React from 'react';

interface PozdravProps {
  name: string;
}

const Pozdrav: React.FC<PozdravProps> = ({ name }) => {
  return (
    <div>
      <h2>Ahoj, {name}</h2>
    </div>
  );
}

export default Pozdrav;
```

Zdrojový kód 4.2: Příklad vytvoření vlastní komponenty v Reactu

4.2.3 Hook

Hooky umožňují komponentám uchovávat stav proměnných a při změně dat upravit i obsah webové stránky. Zabudovanými hooks v Reactu je více, popíšu zde pouze základní z nich.

Funkce `useState` je základní stavový hook v Reactu, který umožňuje komponentám ukládat stav a aktualizovat ho. Tento přístup umožňuje komponentám reagovat na události a změny stavu, což je klíčové pro dynamické chování uživatelského rozhraní.

```
import { useState } from 'react';
export default function Counter() {
  const [count, setCount] = useState(0);

  function handleClick() {
    setCount(count + 1);
  }

  return (
  <button onClick={handleClick}>
    Tlačítko bylo zmáčknuto: {count}
  </button>
  );
}
```

Zdrojový kód 4.3: Příklad hooks useState

Hook `useEffect` je funkce v Reactu, která umožňuje provádět vedlejší efekty v komponentách. Ve výchozím nastavení je volán po každém vykreslení komponenty. Jako druhý parametr je možné přidat takzvané *dependency array*, což je pole proměnných, funkcích nebo objektů. Pokud jsou v tomto poli uvedeny nějaké proměnné, bude se efekt spouštět pouze tehdy, když se některá z těchto proměnných změní. V případě, že je pole *dependency array* prázdné, provede se příkaz jen při prvním vykreslení komponenty.

```
useEffect(() => {
  // Provede se jen při prvním vykreslení
}, []);
```

Zdrojový kód 4.4: Příklad použití hook useEffect

4.3 WebAssembly

WebAssembly [11] (zkráceně WASM) je binární instrukční formát určený pro běh ve webových prohlížečích. WebAssembly byl vytvořen organizací W3C a vydán jako otevřený standard [11]. W3C je otevřená organizace, která se zaměřuje se na specifikaci webových technologií. Tradiční kompilované programovací jazyky, jako například C, C++ a Rust je možné přeložit do formátu WASM a ten následně spouštět ve webovém prohlížeči. Programy přeložené do formátu WebAssembly jsou obvykle výrazně rychlejší než ekvivalentní kódy v JavaScriptu. Formát WASM je nezávislý na platformě prohlížeče což umožňuje spouštění stejného kódu na různých zařízeních a v různých prohlížečích [11]. Díky tomu je formát podporován i na mobilních zařízeních [11]. Aktuálně (2024) je podporován všemi nejpoužívanějšími prohlížeči, tedy konkrétně Chrome, Edge, Firefox a Safari.

4.3.1 Specifikace

Specifikace WebAssembly stanovuje instrukční sadu, binární kódování, validaci, sémantiku provádění a textovou reprezentaci. Nedefinuje však, jak mohou programy WebAssembly pracovat s konkrétním prostředím, ve kterém se spouštějí, ani jak jsou z takového prostředí vyvolány [11]. Jinými slovy spuštění programu WebAssembly musí být provedeno pomocí jiného programovacího jazyka, na webových aplikacích se k tomu používá JavaScript.

Název technologie WebAssembly evokuje, že se bude jednat o jazyk symbolických adres (zkr. JSA), ovšem jde o dvě různé věci. Programovací jazyk WebAssembly je mezijazyk, který je uložen v binární podobě a lze jej jej přirovnat k *Java bytecodu*. Pro lepší čitelnost je definována textová reprezentace WebAssembly, nazývaná *Wat* [11]. Binární formát WebAssembly používá příponu souborů `.wasm` a textová reprezentace `.wat` [11]. Formát Wat se zapisuje pomocí takzvaných *S-výrazů* [12], anglicky *S-expression*,který je slovně popsán jako *symbolický výraz*. S-výrazy se používají například v programovacím jazyce Lisp, kde jsou použity jak pro zdrojový kód tak i pro data.

4.3.2 Překladač

Formát WebAssembly je možné psát přímo v binární nebo textové formě. Předpokládá se však, že program napíšeme například v programovacím jazyce C/C++ a ten přeložíme do WebAssembly [11].

Ve své práci potřebuji přeložit programovací jazyk C do WebAssembly. Jedním z nástrojů umožňujících překlad z tohoto programovacího jazyka je například *Emscripten* [13]. Záměrně jsem použil termín „nástroj“ místo „překladač“, protože *Emscripten* využívá existující překladače, které po sobě sekvenčně spouští a umožňuje jejich nastavení pomocí přepínačů. Tato sada nástrojů dokáže přeložit jakýkoliv programovací jazyk, který používá *LLVM* překladač [13]. Díky této vlastnosti je možné přeložit i jazyky jako C++ a Rust [13]. Více informací o tomto nástroji bude popsáno v sekci 4.4.

> *„LLVM je kolekce modulárních a znovupoužitelných překladačů a technologií toolchain. Navzdory svému názvu má LLVM jen málo společného s tradičními virtuálními stroji. Samotný název LLVM není zkratkou; je to celý název projektu."* [14]

4.3.3 Základní koncepty

Výpočetní model WebAssembly je postaven na zásobníkovém stroji, zvaném v angličtině *stack machine* [11].

Pro ukládání proměnných a dalších dat se používá *lineární paměť*, která je organizována v sekvenčním pořadí, kde každý byte má svou unikátní adresu. Hodnoty v lineární paměti je možné upravovat i po jejím vytvoření [11]. Lineární paměť je vytvořena s počáteční velikostí, ta se však může dynamicky zvětšovat podle potřeby během běhu programu. Pokud se pokusíme přistoupit k paměti, která je definována mimo rozsah, program reaguje vyhozením výjimky [11].

Modulem se v kontextu WebAssembly nazývá binární soubor s příponou `.wasm`, který obsahuje definice funkcí, tabulek a lineárních pamětí [11]. Pracovat s moduly lze ze spouštěného prostředí. Moduly umožňují exportovat funkce do spouštěného prostředí [15]. Je také možné importovat funkce z JavaScriptu do WebAssembly [15].

4.3.4 JavaScript API

V této sekci popíši, jak spustit program v binární podobě WebAssembly uvnitř JavaScriptu. Následující zdrojový kód (viz 4.5) jsem přeložil nástrojem Emscripten do binárního formátu WebAssembly.

```
1  int foo(int x) {
2    return x / 2;
3  }
```

Zdrojový kód 4.5: Příklad programu v programovacím jazyce C s ukázkou kompilace do jazyka WebAssembly

Zkompilovaný program ve formátu WebAssembly, nelze přímo spustit. WebAssembly totiž nepředepisuje prostředí, ze kterého je spouštěn [11]. Naštěstí však existuje v JavaScriptu API pro spouštění WebAssembly. Pro jednoduché načtení jsou potřebné tři kroky: získat soubor `.wasm`, zkompilovat jej pro konkrétní zařízení a načíst WebAssembly modul [15].

```
1  let deleni, pamet;
2  fetch("delicka.wasm")
3    .then(bytes => bytes.arrayBuffer())
4    .then(mod => WebAssembly.compile(mod))
5    .then(module => {return new WebAssembly.Instance(module)})
6    .then(instance => {
7      // Přístup k exportované funkci
8      deleni = instance.exports.foo;
9
10     // Přístup k exportované lineární paměti
11     pamet = new Uint8Array(instance.exports.memory);
12     // Nahrání hodnoty do lineární paměti na offset 0
13     pamet[0] = 42;
14   });
```

Zdrojový kód 4.6: Použití WebAssembly pomocí JavaScript API

Na zdrojovém kódu 4.6, je načten WebAssembly modul. Modul WebAssembly byl přeložen ze zdrojového kódu 4.5 napsaného v programovacím jazyce C. Program funguje tak, že nejprve stáhneme soubor `.wasm`, který obsahuje přeložený kód WebAssembly ze serveru, příkazem `fetch`. Jakmile je binární soubor stažen, je uložen do instance `ArrayBuffer`. Poté zkompilujeme novou instanci příkazem `WebAssembly.compile()` do modulu WebAssembly a po kompilaci vytvoříme jeho instanci [15].

Ve zdrojovém kódu (viz 4.6) na řádce 7, je získán přístup k exportované funkci `foo` [15].

Pokud se pracuje s větším programem, může být manipulace skrze JavaScript API komplikovaná. Proto existuje takzvaný *glue code*, který může být vygenerován překladačem a je naprogramován ve vysokoúrovňovém jazyce obvykle v JavaScriptu. S výsledným *glue code* lze obvykle pracovat jako s ES6 moduly, které jsou běžně používané v programovacím jazyce JavaScript [16].

4.4 Emscripten

Emscripten [13] je open-source překladač, umožňující zkompilovat libovolný programovací jazyk, který používá LLVM, do kódu WebAssembly. Dále umožňuje zkompilovat programovací jazyky, které spoléhají na interprety napsané v C/C++ jako například Python a Lua. Emscripten je primárně používán pro překlad programovacího jazyka C/C++. Tento nástroj umožňuje ve WebAssembly vykreslovat grafiku, přehrávat zvuky, zpracovávat soubory a mnoho dalšího.

Emscripten je již používán v mnoha komerčních aplikacích. Například v herních enginech je tento nástroj využit pro generování her do webového prostředí. Jako konkrétní příklady lze uvést v Unity a Unreal Engine [13].

4.4.1 Toolchain

Emscripten je sada nástrojů která obsahuje dva hlavní nástroje [17]. Vývojový balíček Emscipten je možné nainstalovat na operační systémy Windows, Mac OS a Linux [13].

- *Emscripten Compiler Frontend (emcc)* - Nástroj *emcc* je používaný pro volání emscripten překladače z příkazové řádky [17].

- *Emscripten SDK (emsdk)* - Nástroj *emsdk* se používá z příkazové řádky. Slouží pro správu celého nástroje Emscripten SDK. Umožňuje například stažení, aktivaci odstranění různých verzí překladačů [17].

4.4.2 Emscripten Compiler Frontend

Emscripten Compiler Frontend neboli zkráceně `emcc`, je překladač který umožňuje přeložit vysokoúrovňový programovací kód do WebAssembly [18]. Překladač však nedělá jen kompilaci, cílem Emscriptenu je nabídnou aplikacím, které nebyly určeny pro běh na webu, aby mohly plně běžet ve webovém prostředí [18]. Proto Emscripten emuluje celý POSIX operační systém, včetně virtuálního filesystému, síťového rozhraní a mnoho dalšího [18]. Emscripten zároveň podporuje použití OpenGL pro vykreslení aplikace na webovém rozhraní [18].

Emscripten pro překlad používá existující nástroje, které spouští sekvenčně po sobě. Jedním z nich je LLVM, což je modulární překladač, který programovací jazyk nepřeloží přímo do strojového kódu, ale překládá jej nejprve na mezijazyk [18]. Výhodou tohoto přístupu je, že lze použít již existující nástroj a přidat do něho jen nový backendový překladač pro WebAssembly. Díky tomu podporuje Emscripten všechny programovací jazyky, které jsou podporovány překladačem LLVM.

4.4.3 Emscripten Runtime Environment

Emscripten Runtime Environment řeší problémy, spojené s spuštěním aplikace, která nebyla určena pro webovou aplikaci. Mezi základní problémy patří manipulace se vstupy, výstupy, soubory a zvuky [19].

Aplikace naprogramované v programovacím jazyce C/C++ používají synchronní rozhraní pro přístup k souborům. Ve webovém prostředí je to ale problematické, protože se soubory načítají asynchronně skrze JavaScript [19]. Emscripten poskytuje vlastní implementaci virtuálního filesystému a knihoven *libc* a *libcxx*, které slouží jako rozhraní pro práci se soubory v programovacím jazyce C/C++ [19]. Emscripten podporuje více typů virtuálního filesystému, kde každá z nich se liší výhodami a nevýhodami [19]. Jako výchozí virtuální filesystém používá *MEMFS*, který ukládá data do operační paměti a její obsah je ztracen po obnovení okna prohlížeče [19].

4.4.4 Rozhraní API

Emscripten obsahuje některé knihovny pro programovací jazyk C/C++ a JavaScript. Tyto rozhraní umožňují upravit programovací kód, aby jej bylo možno včetně všech funkcionalit spustit i na webu [20]. Jedním z klíčových je *emscripten.h* pro programovací jazyk C/C++ [20].

Rozhraní zpřístupněné pomocí *emscripten.h* umožňuje upravit programovací kód tak, aby využíval funkce z JavaScriptu, exportoval funkce do JavaScriptu a mnoho dalšího.

4.4.5 Module objekt

Jak jsem již zmínil Emscripten umožňuje vytvořit takzvaný *glue code*, který ulehčuje interakci s WebAssembly kódem. Soubor obalovacího JavaScript kódu obsahuje jen globální proměnnou *Module* [21]. Tento soubor je možné importovat buď standardně pomocí atributu `<script>` do `.html` souboru, nebo obvykleji jako JavaScript *module* pomocí pojmenovaných exportů. V případě JavaScript modulu je potřeba navíc přidat do parametru překladu `MODULARIZE=1`.

Module je možné upravit jeho změnou příslušných vlastností dle dokumentace. Pokud při překladu je `MODULARIZE=0` je možné před importem vytvořit JavaScript module pod názvem *Module* a nastavit mu požadované vlastnosti. V případě překladu s `MODULARIZE=1` se předává nastavení modulu jako parametr konstruktoru.

```
var Module = {
  'print': function(text) { alert('stdout:' + text) },
  'printErr': function(text) { alert('stderr:' + text) }
};
```

Zdrojový kód 4.7: Ukázka úpravy globálního objektu Module [21]

4.4.6 Interakce s kódem

V následující sekci popíšu možnosti interakce kódu C/C++ s kódem JavaScript s použitím globálního Module. Existují v zásadě dvě možnosti jak interagovat s kódem C/C++. Je možné spouštět hlavní funkci `main()` a data předávat jako argumenty. Nebo je možné funkci `main()` nepoužívat a spouštět jednotlivé funkce kódu C/C++.

Spustit zkompilovanou funkci C/C++ z jazyka JavaScript, umožňují vlastnosti `cwrap()` a `ccall()` z globálního objektu *Module*. Funkce `ccall()` spustí funkci ihned. Naproti tomu `cwrap()` vrátí klasickou JavaScript funkci, kterou je následně možné spustit s příslušnými atributy. Ukázka zmíněné funkcionality je vidět na zdrojovém kódu **??**.

```
// ccall - (název funkce, návratový datový typ,
      datové typy attributů, argumenty)
var result = Module.ccall('foo',  'number', ['number'], [12]);

// cwrap - (název funkce, návratový datový typ,
      datové typy attributů)
var funcFoo = Module.cwrap('foo', 'number', ['number']);
result = funcFoo(12);
```

Zdrojový kód 4.8: Použití vlastností ccall a cwrap objektu Module pro kód **??**

Jestliže funkce `foo()` není v programu C/C++ nikdy volána, překladač ji do sestavení nevkládá. Je možné tomu, ale zamezit úpravou kódu C/C++ následujícím způsobem jako na zdrojovém kódu 4.9.

```
1  #include <emscripten.h>
2
3  int EMSCRIPTEN_KEEPALIVE foo(int x) {
4    return x / 2;
5  }
```

Zdrojový kód 4.9: Příklad programu v programovacím jazyce C s ukázkou kompilace do jazyka WebAssembly

Úprava implementace sady CBlazon — 5

Program CBlazon byl částečně upraven pro překlad do WebAssembly modulu. Nicméně bylo nezbytné upravit výstup programu, vytvořit spustitelný soubor vhodný pro opakované volání a najít vhodné přepínače pro překladač *Emscripten*.

5.1 Výstup formátu JSON

Programy sady *CBlazon* generují výstupy ve formě nevalidního XML nebo abstraktního stromu. Žádný z těchto výstupů není vhodný pro moji aplikaci, protože by vyžadoval vytvoření speciálního parseru. Proto jsem se rozhodl upravit generovaný výstup tak, aby odpovídal validnímu formátu. Vzhledem k tomu, že budu s daty pracovat v jazyce JavaScript, jsem zvolil formát JSON oproti XML.

Úpravy v programu pro generování formátu JSON byly provedeny rozdělením souboru *cblason.c*. Specifické funkce pro výstup abstraktního stromu zůstaly v souboru *lib/cblason.c*. Nově byly přidány dva další soubory: *lib/cbl_ast.c* a *lib/cbl_json.c*. Soubor *cbl_ast.c* využívá původní implementaci abstraktního stromu, zatímco druhý soubor *cbl_json.c* slouží k generování výstupu ve formátu JSON.

Výstup ve formátu JSON je nyní možné spustit pomocí spustitelného souboru *cbltag.c*. Upravený program dokáže generovat jak původní výstup, tak i nový formát JSON.

```
cbltag json <teststring>
```

Zdrojový kód 5.1: Generování JSON výstupu pomocí programu cbltag, který je součástí sady CBlazon

Výpis 5.2: Výstup programu programu cbltag, který je součástí sady CBlazon

```
Text:
v červeném štítě s trávníkem vykračující jelen
přirozených barev

JSON:
{
    "shield": {
        "splits": {
            "type": "UNSPLIT",
            "style": "",
            "where": "CENTRE",
            "place": "root",
            "field\_ix": "0",
            "splits": []
        },
        "fields": [
            {
                "colour": "červený",
                "objects": [
                    {
                        "name": ["trávník"],
                        "quantity": 1,
                        "tag\_attrs": [],
                        "obj\_attrs": [],
                        "natural\_colours": false
                    },
                    {
                        "name": ["jelen"],
                        "quantity": 1,
                        "tag\_attrs": ["vykračující"],
                        "obj\_attrs": [],
                        "natural\_colours": true
                    }
                ]
            }
        ]
    },
    "bearer": null,
    "helmet": null,
    "valid": true
}
```

5.2 Rozhraní cblweb

Sada CBlazon neobsahuje žádný spustitelný soubor umožňující zadat vstup a získat z něho pouze výstup bez dalších dat. V následující sekci popíšu vytvořený spustitelný soubor *cblweb.c*, který jsem do práce přidal. Tento soubor využívá implementaci specifickou pro překladač Emscripten a nelze jej přeložit pomocí jiných překladačů. Rozhodl jsem se nepoužívat hlavní funkci `main()` a volat zkompilované funkce jednotlivě. Nakonec obsahuje soubor pouze jednu funkci 5.3, která je spustitelná z JavaScriptu. Tato funkce přijímá text v jazyce blazon jako argument a vrací text ve formátu JSON.

```
char* cblazon_process(char *blazon_text) { ... }
```

Zdrojový kód 5.3: Hlavní funkce cblweb.c

Implementace knihovny CBlazon neumožňuje uložit výstup programu do proměnné, ale vypisuje se přímo do streamu typu `FILE*`. Naštěstí Emscripten umožňuje vytvořit virtuální filesystém. Výpis je proto zaznamenán do dočasného souboru, který následně čtu a vrátím jako proměnnou typu *char**. Typ *char** se v jazyce JavaScript automaticky převede na *string*.

Výpočetně nejsložitější je *morforizace* vstupního textu. Z toho důvodu jsem se rozhodl vytvořit jen jednu funkci, která vrátí veškeré možné výstupy programu. Formát výstupu funkce je zobrazen na zdrojovém kódu 5.4. Index `error` je vždy celé číslo, pokud je rovno 0, potom nenastala žádná chyba.

```
{
  "error": number,
  "error_msg": string,
  "json": string,
  "ast": string,
  "tag": string,
  "token": string,
  "tree_html": string
}
```

Zdrojový kód 5.4: Formát výstupu hlavní funkce cblweb.c

5.3 Použití pro WebAssembly

V následující sekci popíšu postupně přepínače překladače *Emscripten*, které bylo potřeba přidat do projektu. Mým požadavkem bylo použít globální ES6 WebAssembly *Module* a zároveň zajistit kompatibilitu s Webpackem, který jsem použil v projektu. Rozhodl jsem se nepoužívat hlavní funkci `main()` a volat zkompilované funkce

C/C++ individuálně. Informace k této sekci jsem primárně čerpal z následujícího návodu [22] a oficiální Emscripten dokumentace [23].

```
-s ENVIRONMENT='web'
```

Zdrojový kód 5.5: emsc přepínač Enviroment

Přepínač na zdrojovém kódu 5.6 nastavím, protože nechci používat prostředí WebView, Web worker ani Node.js, protože je nepotřebuji.

```
-s EXPORT_ES6=1
-s MODULARIZE=1
```

Zdrojový kód 5.6: emsc přepínač pro modularizaci a formát JavaScript kódu ES6

Přepínače na zdrojovém kódu 5.6 nastavím, protože chci aby JavaScript *glue code* byl ve formátu ES6 a aby se jednalo o JavaScript module, který je možné snadno importovat.

```
-s USE_ES6_IMPORT_META=0
```

Zdrojový kód 5.7: emsc přepínač pro nastavení meta.url

Přepínač na zdrojovém kódu 5.7 nastavím na hodnotu 0, protože výchozí použití `import.meta.url` v tomto nastavení způsobuje chybu ve Webpacku.

```
-sEXPORTED_RUNTIME_METHODS=ccall,cwrap
```

Zdrojový kód 5.8: emsc přemínač exported methods

Nastavením následujících přepínačů na zdrojovém kódu 5.8 exportuji funkce modulu `ccall` a `cwrap`.

```
-sALLOW_MEMORY_GROWTH
-sFORCE_FILESYSTEM
```

Zdrojový kód 5.9: emsc ostatní použité přepínače

Na zdrojovém kódu 5.9 nakonec povolím rozšiřování paměti za běhu a vynutím použití virtuálního filesystému *MEMFS*

V následující části popíšu nezbytné úpravy v JavaScriptu pro nasazení WebAssembly ve webové aplikaci. Překladač je nastaven a překladem vygeneruji soubory WebAssembly. Po dokončení překladu vznikne několik souborů, z nichž mě zajímají pouze *cblweb.wasm*, *cblweb.data* a *cblweb.js*. Soubory *cblweb.wasm* a *cblweb.data* přesunu do složky `public`. Díky tomu budou soubory dostupné z webového serveru. Soubor *cblweb.js* začlením do zdrojových souborů aplikace a později ho importuji.

Protože jsem přesunul soubory WebAssembly do složky *public*, je potřeba globálnímu modulu specifikovat, kde má tyto soubory hledat. Toto nastavení je možné nastavit změnou výchozích vlastností modulu pod klíčovým slovem `"locateFile"`.

Konkrétní nastavení globálního modulu je vidět v zdrojovém kódu 5.10. Ve výchozím nastavení se po načtení WebAssembly modulu spustí funkce `main()`, tuto možnost vypnu klíčovým slovem "`noInitialRun`".

```
const moduleConfiguration = {
  noInitialRun: true,
  locateFile: (path: string, scriptDirectory: string) => {
    // Prepend the root path to the filename
    return "/" + path;
  }
}
```

Zdrojový kód 5.10: Nastavení globálního modulu pro cblweb.js

Jak vyplývá z předchozího vysvětlení, není potřeba pouze přeložit aplikaci, ale také správně umístit soubory do projektu. Z tohoto důvodu jsem vytvořil bash skript, který usnadňuje proces překladu a přesunu souborů. Tento skript je umístěn v cestě `src/cblazon/_compile_emscripten_cblason.sh` a před prvním spuštěním je nutné tento soubor otevřít a upravit proměnnou obsahující cestu k projektu *CBlazon*.

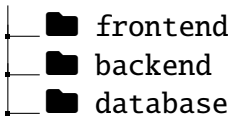
Implementace aplikace

6

V následující kapitole popíši vytvořenou implementaci klientské webové aplikace. Zdrojové soubory jsou k dispozici na Gitlabu Katedry Informatiky a výpočetní techniky, Fakulty aplikovaných věd, Západočeské univerzity [1].

6.1 Struktura projektu

Hlavní složka projektu je organizována na frontend, backend a databázi. V této bakalářské práci byla nakonec vytvořena pouze část frontendu, avšak vytvořená struktura může být využita v navazující práci. Projekt je s použitím technologie *Docker* rozdělený do jednotlivých kontejnerů, které je možné vytvořit příkazem: `docker compose up`. Tím se vytvoří dva image kontejnery jeden s *MariaDB* databází a druhý s *Apache HTTP Serverem*. Výhoda tohoto přístupu spočívá především ve snadném navázání na práci bez nutnosti konfigurace či instalace webového serveru a databáze. Kontejnery nakonec nebyly ve vývoji použity, neboť byla implementována pouze část frontendu.



Obrázek 6.1: Základní hierarchie projektu

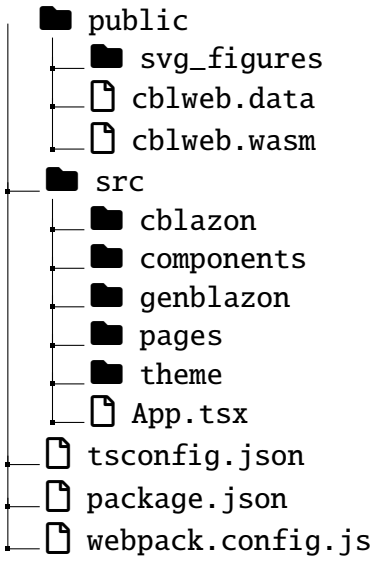
6.1.1 Frontend

Základní struktura projektu aplikace `frontend` byla vygenerována příkazem pro vytvoření nového projektu React. Složka `public` obsahuje veřejné soubory, které jsou dostupné z HTTP serveru. V této složce se nacházejí také přeložené soubory `cblweb`

[1] git projektu: `https://gitlab.kiv.zcu.cz/lipka/cblason-gui`

programu ze sady CBlazon a složka `svg_figures`, která obsahuje data jednotlivých figur.

Kořenová složka projektu obsahuje následující konfigurační soubory: soubor `package.json` obsahuje instalované balíčky knihoven v projektu, soubor `webpack.config.js` zahrnuje konfiguraci nástroje Webpack, zatímco `tsconfig.json` definuje nastavení kompilátoru pro programovací jazyk TypeScript.



Obrázek 6.2: Základní hierarchie projektu aplikace frontend

Zdrojové soubory frontend aplikace jsou ve složce `src`, ve které je složka `components`, kde jsou uloženy vlastní komponenty projektu. Složka `genblazon` umožňuje zpracování výstupu programu CBlazon pro generování grafické vizualizace heraldického znaku. Ve složce `pages` se nacházejí stránky, které se zobrazují uživateli. Složka `theme` obsahuje globální nastavení stylu webové aplikace. Vstupním souborem projektu je `App.tsx`.

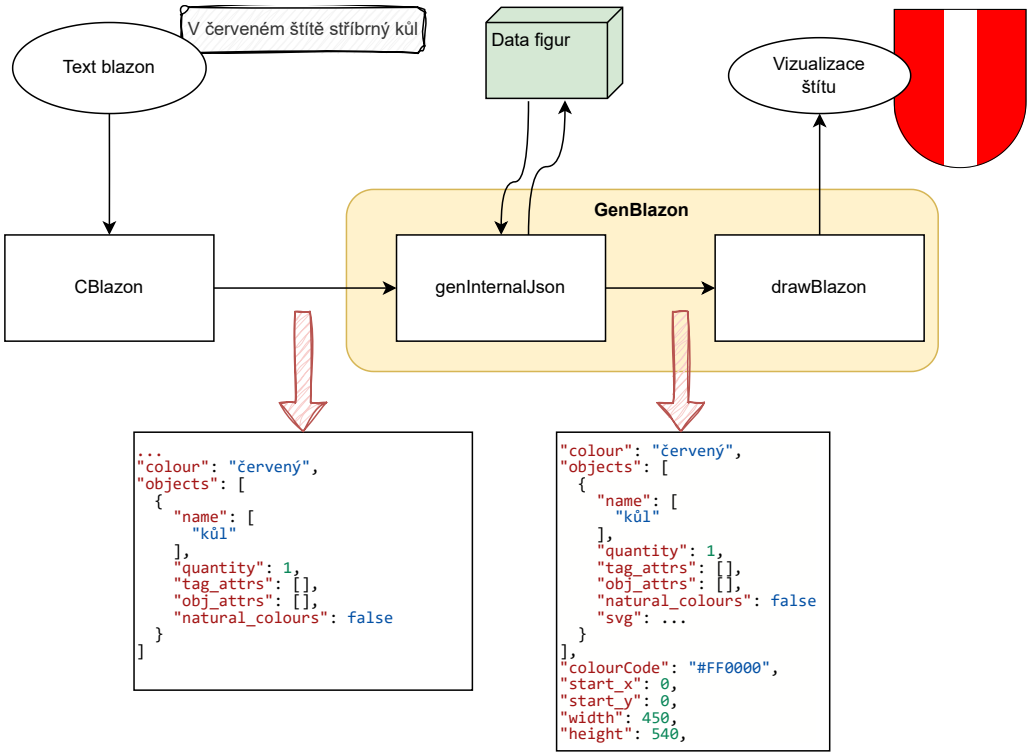
6.2 Proces zpracování textu blazon

V následující sekci popíšu proces zpracování vstupního textu blazon do grafické podoby. Tento proces je zobrazen na diagramu 6.3. Vstupem procesu je text blazon, který je nejprve zpracován programem CBlazon do podoby JSON. Následně je zpracován programem, který jsem nazval `GenBlazon`, jenž umožňuje z JSON popisu blazonu vytvořit grafickou vizualizaci. Tento program je rozdělen na dva samostatné celky: `genSpecificJson` a `drawBlazon`.

Program `genSpecificJson` doplní vytvořený JSON o data elementů SVG obecných a heraldických figur, počáteční a koncové souřadnice jednotlivých polí. V případě figur hledá nejvhodnějšího kandidáta podle metriky vzdálenosti $(0, +\infty)$, kde

0 vyjadřuje nejvhodnějšího kandidáta. Stejně tak v případě barev, kde se použije *Levenshteinova vzdálenost* dvou textových řetězců.

Program `drawBlazon` jako vstup přijme doplněný JSON, který již obsahuje všechna potřebná data pro vytvoření vizualizace štítu. Program manipuluje s elementy SVG a skládá postupně výslednou vizualizaci. Výstupem programu je SVG element, který obsahuje grafickou vizualizaci štítu.



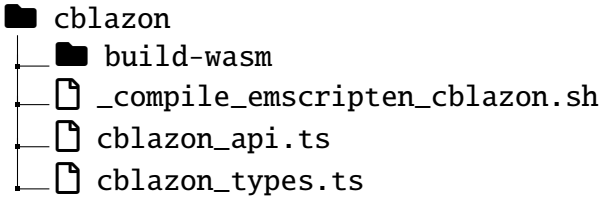
Obrázek 6.3: Proces zpracování textu blazon

6.3 CBlazon

Složka `cblazon` obsahuje soubor `cblazon_api.ts` ve kterém je vytvořeno rozhraní `CBlazonApi`, definované stejnými metodami jako exportované funkce z programu `CBlazon`. Implementace je ve stejném souboru ve třídě `CBlazonProcessor`. Implementace využívá funkce `cwrap` pro binding funkcí z programovacího jazyka C, což umožňuje jejich následné volání z JavaScriptu.

Důležitým souborem je `cblazon_types`, kde je definována struktura výstupu, což umožňuje překladači TypeScript provádět typovou kontrolu nad výstupním objektem.

Bash skript `_compile_emscripten_cblazon.sh` provede přeložení zdrojových souborů projektu CBlazon do složky `build-wasm` a vybrané soubory pak přesune do veřejného adresáře `public`.



Obrázek 6.4: Struktura části programu cblazon

6.4 GenBlazon

`GenBlazon` je klíčová část programu, která umožňuje z popisu znaku ve formátu JSON vytvořit grafickou vizualizaci.

6.4.1 Struktura projektu

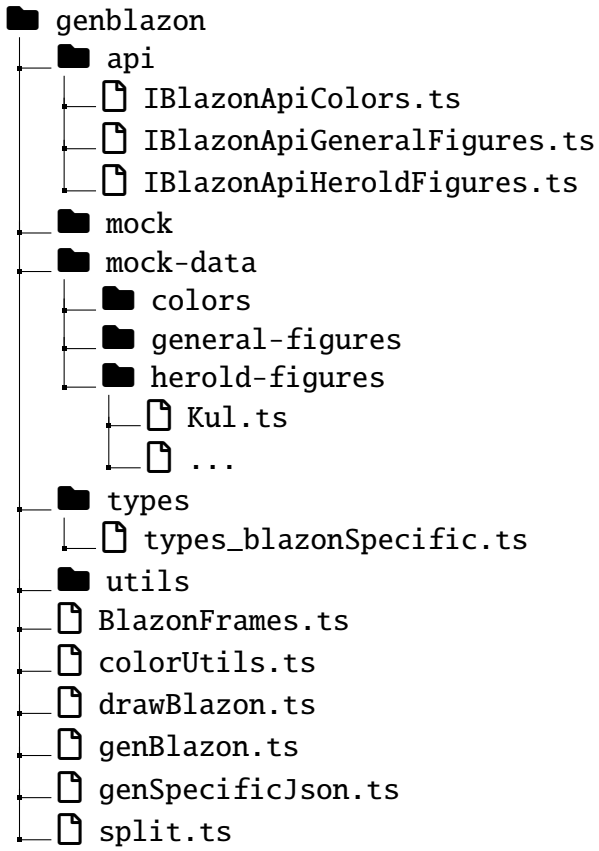
Exportované funkce, určené pro práci mimo složku `genblazon`, jsou v souboru `genblazon.ts`. Již zmíněné části, ze kterých se `GenBlazon` skládá jsou `genSpecificJson` a `drawBlazon`.

Složka `api` obsahuje rozhraní, které definuje strukturu přijímaných dat figur a barev. Takové rozdělení projektu umožňuje nahradit mock stahování barev a figur na backend serveru, který může být napojený na databázi. Složka `mock` obsahuje implementaci více rozhraní ze složky `api`. Data, která používá `mock` implementace, jsou ve složce `mock-data`.

Soubor `types_blazonSpecific.ts` ve složce `types` definuje formát `blazonSpecific`, což je rozšířený formát z JSON výstupu programu CBlazon.

Soubor `split` obsahuje funkce pro zpracování dělení použité v souboru `genSpecificJson`.

Generovat tvar štítu je možné pomocí statické třídy a stejnojmenného souboru `BlazonFrames.ts`.



Obrázek 6.5: Struktura části programu genblazon

6.4.2 Mock

Jednotlivé implementace služeb, které by bylo možné přesunout do backendové části programu, jsou ve složce `mock`. Data pro implementaci jsou uložena v adresáři `mock-data`.

6.4.2.1 Barvy

Implementace je schopná zpracovat pouze barvy a kovy, a nikoliv kožešiny. Záznam barvy se skládá z vlastností: `name`, což je český název barvy, a `color`, vyjadřující HTML hodnotu barvy. Data českých barev jsem získal z webové stránky *Wikipedia*.

Implementace je velice jednoduchá: metoda přijímá požadovanou českou barvu a návratovou hodnotou jsou tři nejpodobnější barvy. Podobnost je určena na základě *Levenshteinovy vzdálenosti* dvou textových řetězců.

```
1 interface IBlazonApiColors {
2   getMatchColors(czechName: string): Color[];
3 }
```

Zdrojový kód 6.1: Rozhraní barev

6.4.2.2 Obecné figury

Implementace rozhraní `IBlazonApiGeneralFigures` najde tři nejpodobnější obecné figury. Podobnost je určena na základě vypočtené vzdáleností dvou figur. Algoritmus porovnává jednotlivé názvy atributů a objektů. Pokud se neshodují s těmi, které jsou uloženy v datech `mock-data`, přičte stanovenou vzdálenost. Velikost přičtené vzdálenosti závisí na počtu zanoření daného objektu atributu.

```
interface IBlazonApiGeneralFigures {
  getMatchFigures(czechName: string[],
      czechTagAttributes: string[],
      czechObjAttributes: ObjectSpecific[],
      naturalColors: boolean): Promise<GeneralFigureEntry[]>;
}
```

Zdrojový kód 6.2: Rozhraní obecných figur

6.4.2.3 Heraldické figury

Implementace rozhraní `IBlazonApiGeneralFigures` vytvoří podle atributů objektu specifikovanou heraldickou figuru. Pokud název heraldické figury není nalezen, vrátí `null`. Implementace umožňuje vytvářet násobné heraldické figury, jako například „dva kůly“.

```
interface IBlazonApiHeroldFigures {
  getMatchHeroldFigure(names: string[],
    start_x: number, start_y: number,
    width: number, height: number,
    tags_attributes: string[],
    quantity: number): Promise<HeroldFigureData|null>;
}
```

Zdrojový kód 6.3: Rozhraní heraldických figur

6.4.3 GenSpecificJson

Program `genSpecificJson` je součástí programu `GenBlazon`. Obsahuje funkci `processBlazonGenerateSpecificJson`, jejímž vstupem je výstup programu CBlazon ve formátu JSON. Tato funkce zpracuje a rozšíří tento vstup o další informace nezbytné pro vykreslení znaku.

6.4.3.1 Pole

Program nejprve zpracuje jednotlivá pole, která byla rozdělena dělením a polcením. Tato pole jsou následně zpracována. Nejprve je český název barvy, převeden na hodnotu barvy HTML.

K jednotlivým polím jsou získány počáteční souřadnice `start_x` a `start_y` a jejich velikost `width` a `height`.

Každé pole je doplněno o počáteční souřadnice `element_x` a `element_y`, které určují obdélníkový prostor, do kterého lze umístit obecnou figuru. Obecná figura nemusí nutně vyplňovat celý tento prostor, ale neměla by z něho vyčnívat. Souřadnice jsou znázorněny na obrázku 6.6.

```
{
  start_x: number,
  start_y: number,
  width: number,
  height: number,

  element_x: number,
  element_y: number,
  element_max_width: number,
  element_max_height: number,

  colourCode: string    // barva HTML
}
```

Zdrojový kód 6.4: Vlastnosti pole

6.4.3.2 Objekty

Oproti výstupu z programu CBlazon ve formátu JSON jsou objekty rozšířeny o následující vlastnosti: `figure_type`, která vyjadřuje, zda se jedná o heraldickou figuru nebo obecnou figuru, a `colorCode`, která vyjadřuje HTML hodnotu barvy. V objektech, na rozdíl od polí, není česká barva obsažena pod klíčovým slovem `colour`, a je nutné tuto hodnotu získat z vlastnosti `obj_attr`, obsahující seznam řetězců. Barva je získána vypočtením vzdálenosti podobnosti textů nad jednotlivými českými názvy barev. Pokud je vzdálenost mezi slovy menší nebo rovna jedné, považuje se hodnota z pole `obj_attr` za barvu a přesune se do klíčového slova `colour`.

```
{
  coulour: string
  colourCode: string    // barva HTML
}
```

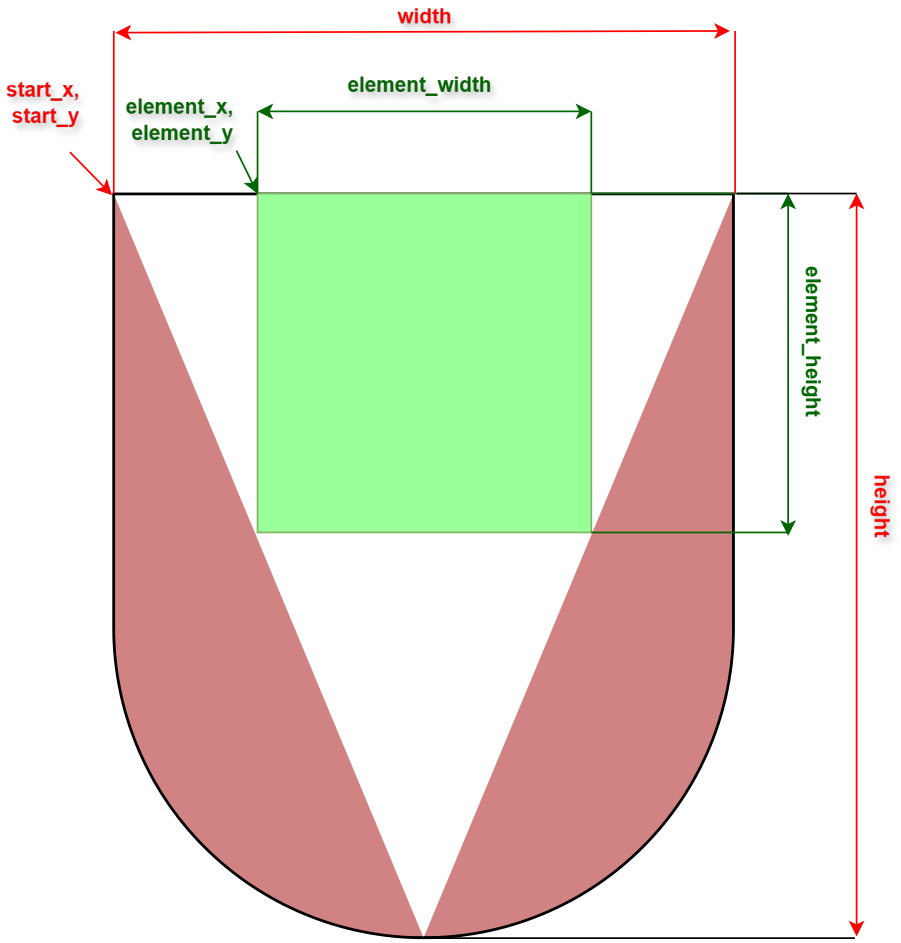
Zdrojový kód 6.5: Vlastnosti objektu

6.4.3.3 Heraldické figury

Heraldické figury fungují obdobně jako pole. Stejně jako do polí je možné vkládat do heraldických figur další figury. Navíc však obsahují vlastnost `svg`, která reprezentuje vizualizaci heraldické figury ve formátu SVG.

```
{
  // + stejné vlastnosti jako pole
  figure_type: FigureType.HEROLD,
  svg: string,
}
```

Zdrojový kód 6.6: Vlastnosti heraldické figury



Obrázek 6.6: Rozměry heraldické figury „klín“

6.4.3.4 Obecné figury

Obecné figury mají následující vlastnosti: `figure_type`, `figureIndex` a `figureData`. Vlastnost `figureData` obsahuje pole dat nejpodobnějších obecných figur. Toto pole obsahuje pouze malé množství figur získaných z rozhraní obecných figur, a to maximálně tři. Vlastnost `figureIndex` určuje index aktuálně vybrané obecné figury

z pole `figureData`. Díky tomu je uživatel schopen změnit konkrétní figuru, která bude ve výsledné vizualizaci zobrazena.

```
{
  figure_type: FigureType.GENERAL,
  figureData: FigureDataSpecific[],
  figureIndex: number,
}
```

Zdrojový kód 6.7: Vlastnosti obecné figury

6.4.4 DrawBlazon

Program `drawBlazon` je součástí programu `GenBlazon`. Obsahuje funkci `generateSvgBlazon`, jejímž vstupem je formát `SpecificJson` a výstupem je textová SVG vizualizace. Program používá knihovnu *d3.js* pro manipulaci s SVG elementy.

Prvním krokem je vytvoření hlavního SVG prvku, za nímž následuje definice tvaru štítu. Následně program prochází jednotlivá pole, která jsou vybarvena podle specifikované barvy, a do nich jsou vloženy figurální prvky, které daná pole obsahují.

6.4.4.1 Tvary štítu

Tvar štítu se vytváří pomocí souboru a stejnojmenné statické třídy `BlazonFrames.ts`. Třída obsahuje metodu `getSquareByType`, která umožňuje vytvoření tvaru štítu podle typu. Návratovou hodnotou je objekt `BlazonFrame`, který obsahuje velikost štítu a cestu štítu ve formátu `path`, který se používá v SVG.

Štít je vykreslován jako obdélníkové pole, které je oříznuto pomocí masky do tvaru štítu. Příklad použití masky je na ukázkovém kódu 6.8.

```
<svg width="450" height="540">
  <defs>
    <mask id="shield_frame">
      <path d="M 0 315 L 0 0 L ..." fill="white"/>
    </mask>
  </defs>

  <!-- Pole 1 --!>
  <rect x="0" y="0" width="450" height="540" fill="red"
      mask="url(#shield_frame)"/>
</svg>
```

Zdrojový kód 6.8: Ukázka aplikace tvaru červeného štítu s použitím masky

6.4.4.2 Figury

Vykreslení figury probíhá odlišným způsobem v závislosti na tom, zda se jedná o heraldickou nebo obecnou figuru. V případě heraldické figury se v SVG přesune pomocí tagu <g> na počáteční pozici figury (`start_x` a `start_y`). Poté se vloží vlastnost objektu pod klíčovým slovem `svg` do nově vytvořeného elementu. Obarvení elementu je provedeno přidáním atributu `fill` s příslušnou barvou do elementu <g>. Výsledek je znázorněn na ukázkovém kódu 6.9.

```
<svg width="450" height="540">
  ...
  <!-- Pole 1 --!>
  ...

  <-- Heraldicka figura (bily klin) 1 --!>
  <g x="0" y="0" fill="white">
    <svg>...</svg>
  </g>
</svg>
```

Zdrojový kód 6.9: Ukázka heroldické figury ve výstupu SVG

Pokud se má vložit obecná figura, nejprve se provede přejmenování všech identifikátorů (ID) z SVG dat obecné figury, aby se zabránilo duplicitním názvům identifikátorů. Identifikátory není v některých případech možné smazat, protože na ně mohou být vázány odkazy, které jsou použity například v maskách. Poté je potřeba vypočítat měřítko, které je nutné aplikovat na figuru, aby se vešla do vymezeného prostoru u objektů uložených pod klíčovým slovem `element_<...>`. Nakonec jsou obarveny části figury podle názvu CSS stylů, které byli uloženy společně s figurou v `mock-data` datovém úložišti.

Zdrojový kód 6.10 demonstruje výpočet měřítka, které je nutné aplikovat na obecnou figuru, aby se vešla do definovaného prostoru. V tomto kódu jsou proměnné `scaleX` a `scaleY` určeny jako podíl maximální výšky a šířky rodičovského prvku a výšky a šířky obecné figury. Nakonec je zvoleno menší z těchto dvou měřítek jako konečné měřítko pro transformaci SVG prvku.

```
const scaleX = parent.element_max_height / figureHeight;
const scaleY = parent.element_max_width / figureWidth;
const scale = Math.min(scaleX, scaleY);
```

Zdrojový kód 6.10: Ukázka vypočtení měřítka pro obecnou figuru

6.5 Stránka

Jednotlivé vizualizace stránek jsou definovány ve složce `pages`. Kromě toho obsahuje složku `documentation`, kde jsou uloženy podstránky sekce dokumentace.

Styl webové stránky je nastaven v souboru `theme/theme_heraldic.ts`. Pomocí něho je možné globálně upravit grafický styl webové stránky.

6.5.1 Komponenty

Webová stránka obsahuje znovupoužitelné komponenty ve složce `components`. Komponenta zajišťující rozložení stránky je v podsložce `page-wrapper`. Hlavní navigační menu je v podsložce `header-navbar`.

6.5.1.1 Editor dat JsonSpecific

K aplikaci jsem vytvořil jednoduchý editor formátu JsonSpecific, který umožňuje: nastavení barev polí a figur, změnu obecné figury. Implementace se nachází podsložce `blazon-editor-json-specific` složky `components`.

Obrázek 6.7: Editor formátu JsonSpecific

6.5.1.2 Blazon runner

Pro jednoduché otestování a zobrazení více textů blazon byla vytvořena minimalistická komponenta, která umožňuje zobrazit vstupní text blazon a výstupní grafickou podobu. Implementace komponenty se nachází v podsložce `blazon-runner`.

Testování 7

V následující sekci popíšu výsledky testování. Aplikace byla testována s ohledem na přesnost, úplnost a uživatelskou přístupnost. Přesnost a úplnost jsem testoval manuálně. Uživatelskou přístupnost testovali dobrovolníci, kteří výsledek testování sepsali do připraveného formuláře.

7.1 Testovací data

Abych mohl provést testování pouze mnou vytvořeného programu `GenBlazon`, zahrnul jsem do testovacích dat pouze ta, která byla úspěšně zpracována programem CBlazon. Tato testovací data jsem čerpal z portálu REKOS [2]. Záznamem testovacích dat je text blazon a jejich odpovídající grafická vizualizace znaku. Pro lepší přehlednost je následující seznam testovacích dat vypsán pouze s názvy měst a obcí.

Testovací data: Mariánské Lázně, Letohrad, Vráž (okr. Písek), Bavorov, Velehrad, Běleč, Besednice, Frymburk, Kalenice, Kunžak, Roseč

7.2 Data figur

V době testování obsahuje program poměrně malé množství dat figur. Konkrétně datový model aplikace obsahuje následující heraldické figury: kůl, břevno, pata, hlava, klín a špice. Aplikace obsahuje celkem 25 obecných figur.

7.3 Limitace

Úplnost programu jsem testoval manuálně a vytvořil jsem seznam věcí, které program GenBlazon není schopen správně vizualizovat. Testování jsem rozdělil do dvou částí. Nejprve jsem testoval omezení, která lze vyřešit bez změny programu CBlazon, a poté jsem se zaměřil na změny, které by bylo potřeba provést v programu CBlazon, aby bylo možné tyto omezení řešit.

V následujícím seznamu jsou sepsány limitace testovacích dat, které je schopný CBlazon úspěšně zpracovat, ale nikoliv program `GenBlazon`.

- Chybějící vykreslení většího počtu obecných figur než 1.
- Chybějící vykreslení zarovnání obecné figury na stranu pole (např. vyrůstající = dole):
- Chybějící vykreslení natočení obecných figur (např. levošikmé).
- Chybějící vykreslení obecné figury s vazbou na libovolné předměty (např. lev držící zmrzlinu).
- Chybějící vykreslení obecné figury s množstevními atributy (např. zkřížené meče).
- Chybějící vykreslení obecné figury s atributy typu polovina figury (např. půlená orlice).
- Pole určené pro umístění figury je vždy obdélníkové.
- Pole určené pro umístění figury nelze rotovat.

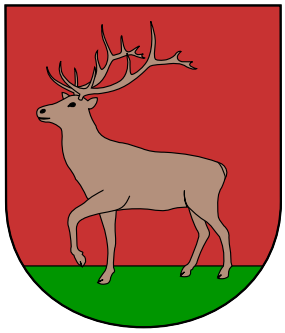
Tímto jsem shrnul limitace, kterým program GenBlazon obsahuje, ale předchozí program CBlazon zvládne zpracovat. V následujícím seznamu jsou sepsány limitace, které nejsou vyřešeny ani v programu CBlazon a tudíž je ani program GenBlazon neumí zpracovat.

- Chybějící podpora v datovém modelu pro upřesňující polohu objektů předložky „nad“, „pod“, a podobně (např. nad kůlem).
- Chybějící podpora pro srdeční štíty.
- Chybějící podpora pro vložení textového řetězce do názvu (např. uvnitř pole písmeno „W“).
- Chybějící podpora pro atributy pozice heraldických figur (např. „kůl vlevo“ nebo „kůl vpravo“).
- Chybějící podpora pro některé atributy objektů (např. „zkrácený kůl“).

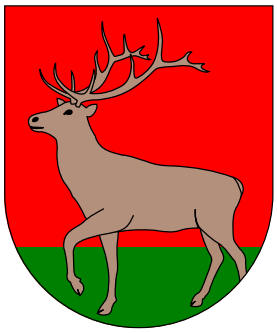
7.4 Přesnost

V následující kapitole popíšu přesnost vytvořeného generátoru vizualizace znaků. Na obrázku 7.1 je vidět porovnání originálního a vygenerovaného znaku města Letohrad. Z obrázků je patrné, že se liší barvy použité ve znaku. Použitá figura ve vygenerovaném znaku je totožná s originálním znakem. Je zřejmé, že figura ve vygenerovaném znaku je větší než by měla být, což způsobuje, že není vidět celá zadní noha jelena. Heraldická figura paty neboli trávníku je o něco vyšší než originální.

Blazon štítu města Letohrad:
V červeném štítě s trávníkem vykračující jelen přirozených barev



(a) originální znak Letohrad [2]



(b) vygenerovaný znak Letohrad

Obrázek 7.1: Srovnání vygenerovaného znaku města Letohrad s originálním znakem

Generované znaky jsou v zásadě velmi podobné originálním znakům, pokud jsou použity stejné figury. Obvykle se však liší velikost objektů a barvy. Protože generátor neumí umístit figury k horní straně pole, jsou figury vždy vycentrovány ke středu, i když by někdy neměly.

7.5 Uživatelské testování

Pro účely testování byl vytvořen testovací dotazník, jehož účelem bylo zjistit uživatelskou přístupnost a chyby aplikace. Znění formuláře je v příloze B. Testery vyplněné dotazníky jsou v příloze C. Dohromady aplikaci otestovalo 9 dobrovolníků, z nichž všichni jsou vysokoškolští studenti.

7.5.1 Reakce na dotazníky

V první otázce z dotazníku mělo být zjištěno, jak se uživatelům s aplikací obecně pracovalo. Ohlasy na ovládání aplikace jsou podle dotazníků celkem pozitivní. Jen tester č. 2 zřejmě vůbec nepochopil, jak aplikaci ovládat. Po analýze, kde se stala chyba, jsem zjistil, že tester č. 2 vložil do textového pole dva blazon texty zároveň, které byly napsány v instrukcích formuláře. S tímto scénářem jsem vůbec nepočítal, možná jsem měl více oddělit jednotlivé texty blazon v instrukcích dotazníku.

Testerům č. 3, 5 a 8 se nelíbilo, že nevěděli, co mohou psát do textu blazon. Tento problém je obtížné ovlivnit, snažil jsem se ho ale alespoň částečně řešit příklady textů blazon. Příklady a použití webové aplikace jsou již v době testování uvedeny v sekci dokumentace. Tester č. 5 navrhl, že by bylo vhodné do sekce dokumentace přidat

také barvy, které je možné do textu blazon zadat. Tento návrh mi přijde vhodný, a proto jsem do dokumentace v aplikaci přidal sekci tinktur.

Vizualizace stromové struktury dat byla oceněna téměř všemi testery. Je zřejmé, že funkcionalita editace dat JSON byla vhodně zvolena.

Dotazník od testera č. 1 byl nejobsáhlejší a obsahuje mnoho užitečných informací. Tento tester zmínil, že na titulní straně musí uživatel přejít až na konec stránky, aby viděl výsledný erb. Tento problém zmínil i tester č. 6. Nejvhodnějším řešením by bylo zvětšit šířku okna a zobrazit vizualizaci štítu na pravé straně. Nakonec jsem tuto změnu neudělal, protože bych nebyl schopen opravené řešení před finálním odevzdáním dostatečně otestovat.

Tester č. 1 zmínil, že mu chyběla možnost zadávání dat erbu ve formátu JSON na titulní straně, aniž by musel zadávat text blazon. Zmíněná funkcionalita však nebyla záměrně vložena na titulní stranu, protože většina uživatelů nebude umět vytvářet erb přímo ve formátu JSON.

Testerovi č. 1 vadilo, že se štít automaticky nezmění při editaci stromu s daty JSON. Problém jsem, ale nakonec z časových důvodů nezměnil, i když by byla zmíněná funkcionalita užitečná.

Testerovi č. 4 se nelíbilo vkládání atributů, ale z důvodu nedostatku informací jsem nezjistil, proč.

7.5.2 Nalezené chyby pomocí dotazníků

Opravil jsem chyby, které zmínil tester č. 1. Prvním z nich byl checkbox „Show tree“, který nezůstal v předchozím stavu po opětovném generování nového štítu. Dále jsem doplnil titulní stranu v sekci dokumentace, aby zde nebyla pouze prázdná stránka.

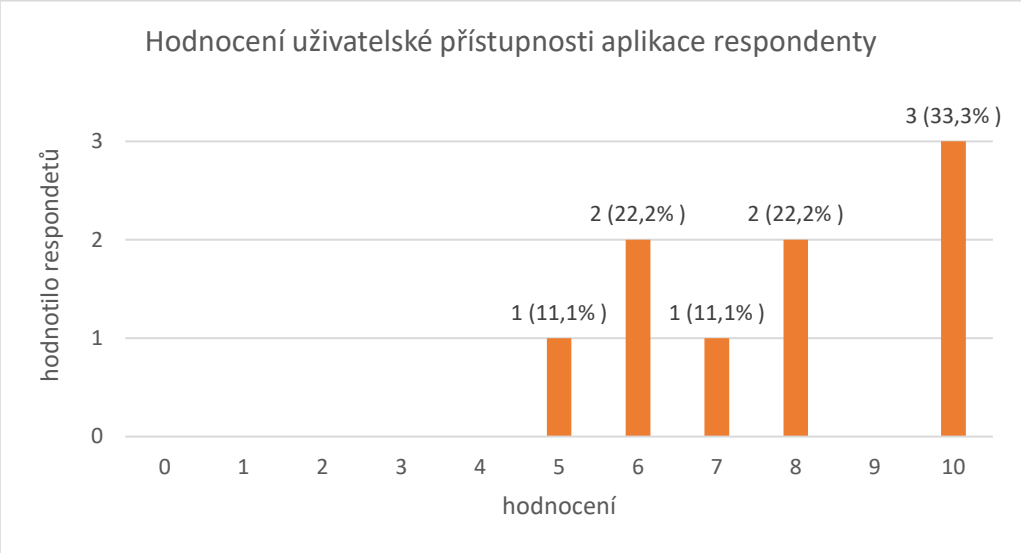
Figura „býk“ byla špatně zadána do programu. Díky testerovi č. 1 byla chyba odhalena a následně opravena. Chyba způsobovala, že tato figura měla vždy červené rohy.

Tester č. 1 nalezl chybu ve špatném vykreslování dělících čar vytvořených při půlení nebo dělení. Tento problém byl způsoben tím, že heraldické figury se překreslovaly přes tyto dělící čáry. Problém byl vyřešen posunutím dělících čar před objekty figur.

Nejzávažnější chybou v aplikaci bylo nesprávné vybírání figury v případě kříže a jeho variant. Chyba se nacházela v algoritmu pro výpočet vzdálenosti mezi atributy figury. Problém způsobila nevhodná velikost konstant vyjadřujících důležitost atributů a podobjektů. Tato chyba byla opravena ihned po obdržení zpětné vazby od testera č. 1, takže další testery již na tuto chybu nenarazily.

7.5.3 Vyhodnocení dotazníku

Na následujícím grafu 7.2 je zobrazeno celkové hodnocení aplikace dobrovolníky. Uživatelé měli hodnotit od 0 do 10, jak se jim s aplikací pracovalo, přičemž 0 znamená nejhorší hodnocení. Hodnocení aplikace se udrželo v rozmezí hodnot od 5 do 10. Průměrné hodnocení je `7,78` a medián je `8`. Velikost hodnocení však může být zkreslena kvůli osobním vztahům s respondenty. Celkově si ale myslím, že k tomu v mnoha případech nedocházelo, protože se hodnocení pohybuje v průměru na úrovni 7.



Obrázek 7.2: Hodnocení uživatelské přístupnosti respondenty

Respondenti hodnotili aplikaci celkově pozitivně. Uživatelům však chyběly některé funkce, což mohlo snížit hodnocení. Během testování aplikace objevili drobné chyby, které by jinak bylo obtížné najít. Někteří respondenti navrhovali různá vylepšení v rozložení uživatelského rozhraní. Z dotazníků jsem si uvědomil, že většině respondentů se líbilo více upravovat texty blazon v sekci dokumentace než na hlavní stránce, i když ta nabízí více možností. Proto by bylo vhodné více upravit domovskou stránku, aby byla pro uživatele přívětivější. Například využitím šířky stránky, aby uživatelé nemuseli příliš posouvat obsah stránky. Tato úprava však nebyla zpracována do finální verze aplikace.

Závěr

Cílem bakalářské práce bylo vytvořit uživatelské webové rozhraní pro generování erbovních znamení podle zadaného popisu erbu nazývaného „blazon“. Nejprve jsem provedl analýzu heraldiky a jazyka blazon, kde jsem se zaměřil především na grafické části znaku.

Seznámil jsem se s vytvořeným nástrojem CBlazon, který slouží k analýze textu blazon. Bylo zjištěno, že nástroj je velice dobře navržen z hlediska datového modelu a množství funkcionalit. I přesto se ale ukázalo, že formát výstupu není vhodný pro navazující práci. Nástroj CBlazon navíc neobsahoval spustitelný program, který by byl vhodný pro webovém prostředí na straně klienta.

Specifikoval jsem požadavky aplikace a stanovil jsem architekturu aplikace. Na základě těchto požadavků byly vybrány vhodné technologie pro vytvoření webové aplikace. Nástroj CBlazon je spouštěn na straně klienta pomocí WebAssembly, a webová aplikace využívá knihovnu React. Bylo rozhodnuto, že backendová část aplikace nebude součástí této bakalářské práce.

Jak již bylo zjištěno z analýzy program CBlazon bylo potřeba upravit. Program byl upraven tak, aby generoval výstup ve formátu JSON, byl vytvořen spustitelný soubor vhodný pro WebAssembly a provedena jeho integrace do projektu s knihovnou React.

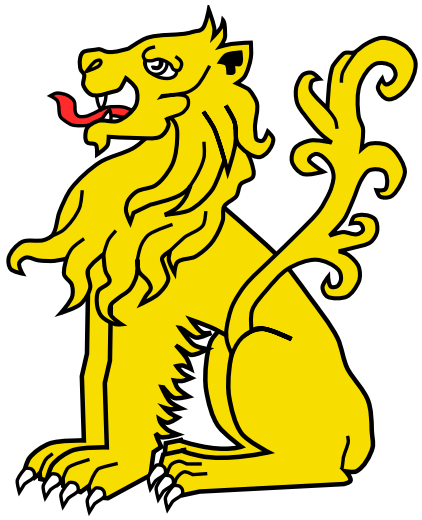
Hlavní částí práce byla implementace webové aplikace, která umožňuje generovat grafickou vizualizaci erbovních znamení podle zadaného popisu blazon. V této části byl vytvořen program nazvaný GenBlazon, který využívá výstup programu CBlazon k vytvoření grafické vizualizace znaku.

Během testování úplnosti bylo zjištěno, že aplikace umožňuje generovat pouze omezené množství erbovních znamení měst a obcí. Ačkoliv aplikace generuje téměř identická erbovní znamení, generované znaky se obvykle liší v použitých odstínech barev a velikostí figur. Uživatelská přístupnost byla testována pomocí dobrovolníků, kteří webovou aplikaci vyzkoušeli na základě dodaných instrukcí. Poté vyplnili testovací dotazník, ve kterém zhodnotili, jak se s aplikací pracovalo, a zda našli nějaké chyby. Respondenti přispěli k nalezení několika chyb v aplikaci. Z výsledků vyplněných dotazníků vyplývá, že se jim s aplikací pracovalo dobře, avšak uvítali by některé změny, které by zlepšily uživatelský zážitek.

Vytvořená webová aplikace položila základní pilíře pro vizualizaci heraldických erbů na základě vstupního textu blazon. I když vzniklé řešení zatím umožňuje generovat pouze malé množství erbů, grafický charakter výstupu je srozumitelný i nezasvěceným uživatelům, kteří by se mohli dozvědět, co je možné vytvořit pomocí formální gramatiky. Aplikace bude použita jako příklad použití formálního překladače pro předmět „Formální jazyky a překladače“, který vyučuje vedoucí práce. Výslednou práci je možné rozšířit z hlediska jednoduššího nahrávání nových obecných figur a vylepšení vizualizace erbů.

Návod pro přidání obecných figur

1. Nalezněte a stáhněte soubor SVG figury, například z webové stránky `https://commons.wikimedia.org/`.

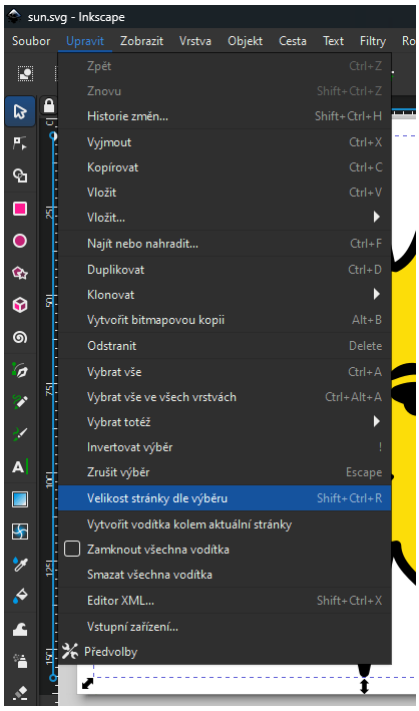


Obrázek A.1: Příklad obecné figury „lev sedící“ [1]

2. Otevřete soubor v programu `Inkscape`. Pokud obrázek obsahuje přebytečné data jako například štít, další figury, odstraňte je.

3. Upravte velikost plátna na velikost figury. Lze udělat jednoduše pomocí `Upravit → Velikost stránky dle výběru`

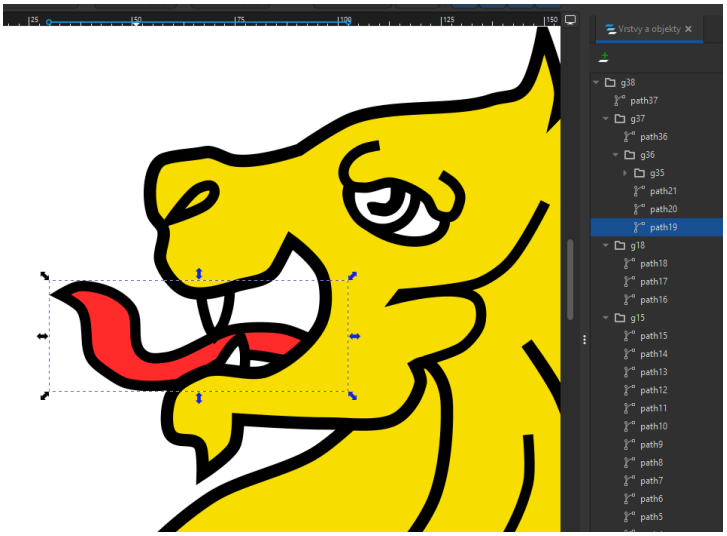
[1] zdroj (veřejná doména): `https://commons.wikimedia.org/wiki/File:Lion_Sejant.svg`



Obrázek A.2: Upravení velikost plátna

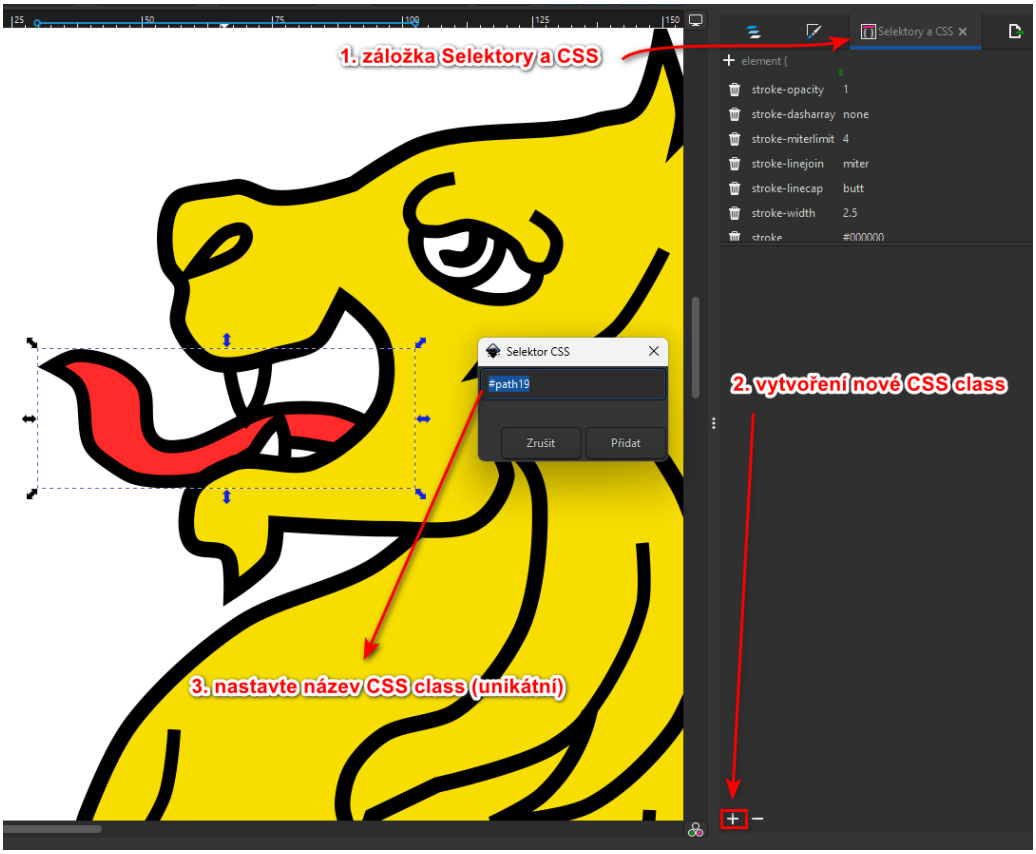
4. Následující část se může lišit podle složitosti erbu. Je nutné vybrat jednotlivé části, které figura obsahuje (například srst, zbroj, ocas, zobák, pařát)

   a) Vyberte části, které chcete zahrnout do objektu. V tomto příkladě například jazyk. Lze si pomoct pravým navigačním polem pro výběr objektů.



Obrázek A.3: Výběr elementu jazyka

b) Vytvoření CSS class pro element jazyk. Pokud chcete vytvořit zbroj (spojené drápy, jazyk, atd.), vytvořte CSS class zbroj do které vložte všechny tyto elementy.



Obrázek A.4: Přidání elementu jazyka do CSS class

c) Opakujeme proces pro všechny objekty, nezapomeneme na srst (hlavní barvu) objektu.

5. Zavřeme aplikaci `Inkscape` a zkopírujeme výsledný soubor do příslušné složky `public` v projektu `cblazon-gui/frontend`.

6. Vytvoříme nový záznam v souboru `src/genblazon/mock-data/general-figures/figures.ts` v projektu `cblazon-gui/frontend`. Jednotlivé atributy jsou v pádu lemmat a jednotném čísle.

```
export const generalFiguresData: GeneralFigure[] = [
  ...

    {
    name: [ "lev" ],
    filename: "animals/lion/Lion_Sejant.svg",
```

```
    tags: [ "sedící" ],
    color_css_name: "srst",
    possible_objects: [
      {
        name: ["zbroj"],
        css_name: "zbroj",
        tags: []
      },
      {
        name: ["dráp"],
        css_name: "drapy",
        tags: []
      },
      {
        name: ["ocas"],
        css_name: "ocas",
        tags: []
      },
      {
        name: ["zub"],
        css_name: "zuby",
        tags: []
      },
    ],
    source_url: "https://commons.wikimedia.org/.../.svg",
  },
]
```

Zdrojový kód A.1: Příklad použití stylu globálního elementu <style> v elementu SVG

Možné problémy

Formát SVG, ve kterém jsou uloženy data obecných figur, obsahuje velké množství typů elementů, hlaviček a podobně. Z tohoto důvodu se mohou někdy objevit neočekávané problémy.

Pokud se objeví problém, že se figura nevykresluje správně, jedním z možných řešení je vytvořit nový SVG soubor a zkopírovat elementy v programu Inkscape do tohoto nově vytvořeného souboru.

Testovací dotazník B

Vaším cílem je otestovat aplikaci pro generování vizualizace heraldických znaků z hlediska uživatelské přístupnosti. To znamená zhodnotit, jak se vám s aplikací pracovalo, zda něco nefungovalo, zda se program choval předvídatelně a podobně.

Heraldika umožňuje popisovat znaky nebo erby formálním jazykem blazon. Na Západočeské univerzitě v Plzni vznikl projekt CBlazon, který umožňuje překlad blazonu do strukturované textové podoby. Pro tento překladač jsem vytvořil v rámci své bakalářské práce webové rozhraní, které generuje grafickou vizualizaci znaků z textu blazon.

Instrukce

Webová aplikace je dostupná na URL adrese: ...

1. Vyzkoušejte zadat jeden či více textů blazon z následujícího seznamu:
   - V červeném štítě český lev se zlatou korunou a zlatou zbrojí.
   - V bílém štítě zelený klín v něm zlatý býk s černými rohy.
   - V prvním stříbrném poli polceného štítu černá plamenná orlice. Ve druhém červeném poli tři bílé břevna.
   - V prvním bílém poli děleného a nahoře polceného štítu čtyři modré kůly. Ve druhém červeném poli bílý kříž. Ve třetím stříbrném poli s zeleným trávníkem hnědý kůň s růžovou zbrojí se zlatým ocasem.
2. Vyzkoušejte upravit text blazon, aby program generoval figury s jinými barvami.
3. Podívejte se na fázi „2. fáze - Stažení dat k erbu“, kde je pomocí stromu možné upravovat vizualizaci znaku. Vyzkoušejte změnu barvy a změnu figury.
4. Vyzkoušejte jeden z textu blazon, který není schopný program CBlazon zpracovat.

- V modrém štítě na černém trojvrší kvádrovaná hradba, uprostřed mezi dvěma válcovými kvádrovanými věžemi s cimbuřím prolomená prázdnou bránou s vytaženou mříží, vše stříbrné.

5. Podívejte se do horního navigačního menu do položky „Dokumentace“ a prohlédněte si jaké znaky umožňuje generátor generovat.

6. Podívejte se do Dokumentace -> Heraldické figury -> Kůl, vyzkoušejte podle dokumentace upravit formát JSON, aby generoval jiný počet kůlů.

Otázky

1. Jak jste byli spokojení s uživatelským rozhraním, na škále 0 až 10? (0 = nespokojení)

2. Co přesně se vám líbilo?

3. Co přesně se vám nelíbilo, či vám připadalo nepřehledné?

4. Zaznamenali jste nějaké chyby uživatelského rozhraní, pokud ano jaké?

5. Která obecná figura (jako například lev, orel, a podobně) vám v aplikaci chybí?

6. Máte ještě nějakou připomínku k aplikaci?

Vyplněné testovací dotazníky

C

Tester 1

1. **Jak jste byli spokojení s uživatelským rozhraním, na škále 0 až 10?**

   10

2. **Co přesně se vám líbilo?**

   aplikace působí velmi příjemně a uceleně

   Dokumentace se mi líbí, protože obsahuje již vygenerované erby a můžu je rovnou editovat v blazonu nebo JSONu. Na aplikaci je vidět velká interaktivita.

3. **Co přesně se vám nelíbilo, či vám připadalo nepřehledné?**

   Asi by se mi víc líbilo, kdyby hned pod polem blazonu byl vykreslen obrázek a ne stromy fáze 1 a 2.

   Jako hlavní výsledek očekávám erb a ne vysvětlení toho, co jsem zadal.

   Ocenil bych, kdyby se erb měnil ihned po výběru figury/zavření colorpickeru.

   Když odkliknu checkbox Show tree tak se strom stejně ukáže po vygenerování nového erbu.

   Po kliknutí na dokumentaci se vykreslí prakticky prázdná stránka s trochou textu (který by klidně mohl být footer). Byl bych radši, kdyby se už rovnou ukázaly nějaké erby. Ušetří to kliknutí.

4. **Zaznamenali jste nějaké chyby uživatelského rozhraní, pokud ano jaké?**

   V dokumentaci u posledního štítu chybí blazon, je pouze poskytnutý JSON soubor.

   Po načtení hlavní stránky chybí možnost zadat erb pomocí editoru JSON, můžu zadávat pouze blazon. Editor se objeví až po úspěšném vykreslení erbu.

5. **Která obecná figura (jako například lev, orel, a podobně) vám v aplikaci chybí?**

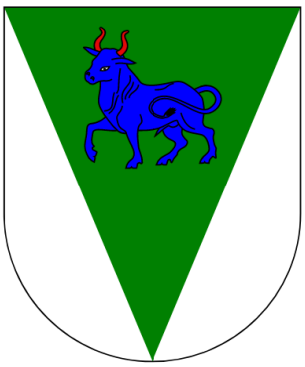
   V modro-zeleně kosmo děleném štítě nahoře kosmá zlatá přirozená lilie na stonku, dole stříbrná hornická kladívka na topůrkách přirozené barvy. (Ražice)

6. **Máte ještě nějakou připomínku k aplikaci?**

   Aplikace je svým vzhledem velmi příjemná. Líbí se mi, že podporuje i různé velikosti okna. Přestože zcela nechápu význam atributů u figur, je i dokumentace přehledná a obsahuje nejspíš všechno co by měla.

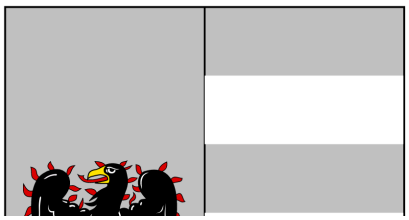
7. **Příloha**

   Barva rohů je pořád červená. "V bílém štítě zelený klín v něm modrý býk s černými rohy."



Obrázek C.1: Chyba vykreslení figury býk, nalezená testerem č. 1

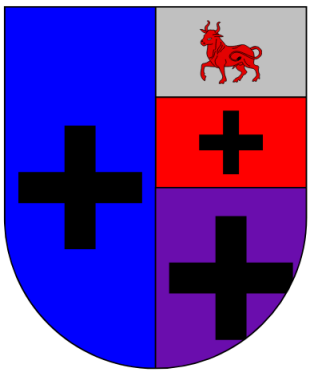
Asi se jedná o detail, ale bílé břevna zasahují do dělící čáry veprostřed. Vypadá potom užší. "V prvním stříbrném poli polceného štítu černá plamenná orlice. Ve druhém stříbrném poli tři bílé břevna."



Obrázek C.2: Chyba vykreslení dělících čáry, nalezená testerem č. 1

U dokumentace nechápu význam atributů. Když zadám černý dvojramenný kříž, očekával bych vykreslení třetího kříže. Zkusil jsem různé variace zadání. Naopak výchozí býk je kráčející i přestože zadám pouze býk.

"V prvním stříbrném poli polceného a vlevo děleného a nahoře děleného štítu býk. Ve druhém červeném poli dvojramenný černý kříž. Ve třetím modrém poli dvojramenný kříž. Ve čtvrtém purpurovém poli černý dvojramenný kříž."



Obrázek C.3: Chyba vykreslení figury kříž, nalezená testerem č. 1

Tester 2

1. **Jak jste byli spokojení s uživatelským rozhraním, na škále 0 až 10?**

   5

2. **Co přesně se vám líbilo?**

   .

3. **Co přesně se vám nelíbilo, či vám připadalo nepřehledné?**

   .

4. **Zaznamenali jste nějaké chyby uživatelského rozhraní, pokud ano jaké?**

   Zadal jsem první dvě věty a hodil mi to tenhle error.

5. **Která obecná figura (jako například lev, orel, a podobně) vám v aplikaci chybí?**

   .

Obrázek C.4: Error testera 2

Tester 3

1. **Jak jste byli spokojení s uživatelským rozhraním, na škále 0 až 10?**

   8

2. **Co přesně se vám líbilo?**

   velmi pomohla vizualizace stromové struktury dat

3. **Co přesně se vám nelíbilo, či vám připadalo nepřehledné?**

   je trochu frustrující snažit se přijít na to, jak přesně má být složena věta - třeba "na zeleném štítu"místo "v zeleném štítu"systém nedokáže vyhodnotit, ano, technicky vzato je "v zeleném štítu"česky správněji než "na zeleném šítu", ale to člověku hned nedojde

4. **Zaznamenali jste nějaké chyby uživatelského rozhraní, pokud ano jaké?**

   ne

5. **Která obecná figura (jako například lev, orel, a podobně) vám v aplikaci chybí?**

   slunce

Tester 4

1. **Jak jste byli spokojení s uživatelským rozhraním, na škále 0 až 10?**

   10

2. **Co přesně se vám líbilo?**

   Rychlá odpověď rozhraní

3. **Co přesně se vám nelíbilo, či vám připadalo nepřehledné?**

   Možnosti měnění atributů

4. **Zaznamenali jste nějaké chyby uživatelského rozhraní, pokud ano jaké?**

   Ne

5. **Která obecná figura (jako například lev, orel, a podobně) vám v aplikaci chybí?**

   Lev

Tester 5

1. **Jak jste byli spokojení s uživatelským rozhraním, na škále 0 až 10?**

   6

2. **Co přesně se vám líbilo?**

   Vykreslování erbů. UI je přehledný a jednoduchý.

3. **Co přesně se vám nelíbilo, či vám připadalo nepřehledné?**

   Trošku nepřehledné je co všechno do textu blazon můžu napsat. Chybí slovní barev.

4. **Zaznamenali jste nějaké chyby uživatelského rozhraní, pokud ano jaké?**

   Když zadám býk tak se mi vykreslí kráčející býk.

5. **Která obecná figura (jako například lev, orel, a podobně) vám v aplikaci chybí?**

   Had, Přilba(helma)

Tester 6

1. **Jak jste byli spokojení s uživatelským rozhraním, na škále 0 až 10?**

   8

2. **Co přesně se vám líbilo?**

   Stromový rozklad a vysoká kvalita vygenerovaného obrázku.

3. **Co přesně se vám nelíbilo, či vám připadalo nepřehledné?**

   Asi bych zvolil širší zobrazení na stránce a využil místo - například rozdělit stránku na dva obdélníky. Pro levý ukázat rozklad, na pravém mít grafické výstupy. Uživatel aktuálně musí hodně scrollovat.

4. **Zaznamenali jste nějaké chyby uživatelského rozhraní, pokud ano jaké?**

   Ne

5. **Která obecná figura (jako například lev, orel, a podobně) vám v aplikaci chybí?**

   Hradba nebo hrad

Tester 7

1. **Jak jste byli spokojení s uživatelským rozhraním, na škále 0 až 10?**

   7

2. **Co přesně se vám líbilo?**

   generování a úprava erbů

3. **Co přesně se vám nelíbilo, či vám připadalo nepřehledné?**

   malý výběr figur

4. **Zaznamenali jste nějaké chyby uživatelského rozhraní, pokud ano jaké?**

   ne

5. **Která obecná figura (jako například lev, orel, a podobně) vám v aplikaci chybí?**

   oheň/plamen, kolo, kopec

Tester 8

1. **Jak jste byli spokojení s uživatelským rozhraním, na škále 0 až 10?**

   6

2. **Co přesně se vám líbilo?**

   Variabilita barev a možnost úpravy po generovaní, možnost stažení

3. **Co přesně se vám nelíbilo, či vám připadalo nepřehledné?**

   Chce to chvilku zvyku než se člověk naučí správně sestavovat věty, ale nic strašného.

4. **Zaznamenali jste nějaké chyby uživatelského rozhraní, pokud ano jaké?**

   Občas si neví rady se slovem "polcené"pokud je jinde ve větě, než je uvedeno v příkladu. Diagonálně dělený erb se mi nepodařilo vytvořit, stejně tak horizontálně dělený bez jakéhokoliv znaku v dané polovině. Problém s pojmy "levá"a "pravá"

5. **Která obecná figura (jako například lev, orel, a podobně) vám v aplikaci chybí?**

   Osobně by se mi líbil pták kivi, ale to je hodně specifické a neobvyklé, takže to jako mínus nepovažuji.

Tester 9

1. **Jak jste byli spokojení s uživatelským rozhraním, na škále 0 až 10?**

   10

2. **Co přesně se vám líbilo?**

   přehlednost, jasnost

3. **Co přesně se vám nelíbilo, či vám připadalo nepřehledné?**

   nic

4. **Zaznamenali jste nějaké chyby uživatelského rozhraní, pokud ano jaké?**

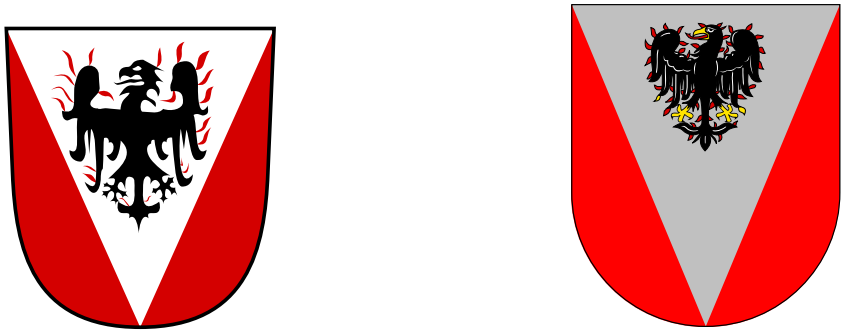
   ne

5. **Která obecná figura (jako například lev, orel, a podobně) vám v aplikaci chybí?**

   liška

Ukázka generovaných znaků z aplikace

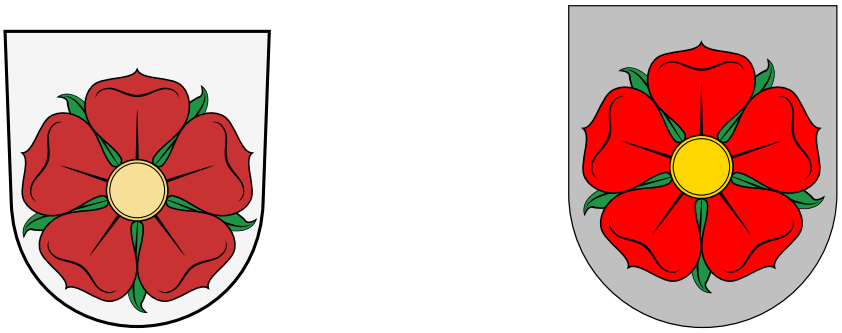
D



(a) originální znak [2]

(b) vygenerovaný znak

Obrázek D.1: Srovnání vygenerovaného znaku obce Vráž (okres Písek) s originálním znakem

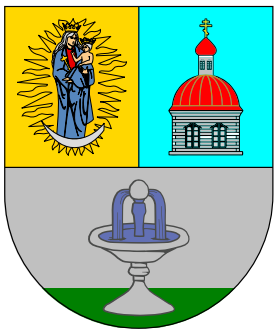


(a) originální znak [2]

(b) vygenerovaný znak

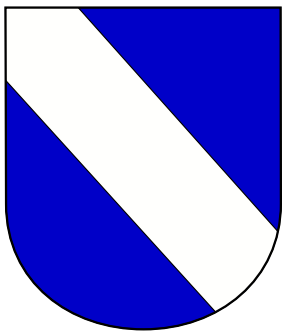
Obrázek D.2: Srovnání vygenerovaného znaku obce Bavorov (okres Strakonice) s originálním znakem

(a) originální znak [2]

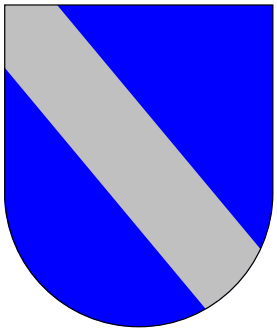


(b) vygenerovaný znak

Obrázek D.3: Srovnání vygenerovaného znaku města Mariánské lázně s originálním znakem



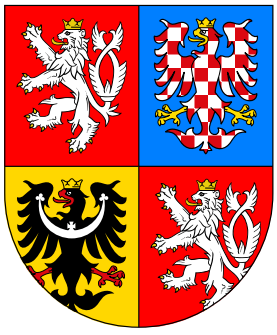
(a) originální znak [2]



(b) vygenerovaný znak

Obrázek D.4: Srovnání vygenerovaného znaku obce Bělá nad Svitavou (okres Svitavy) s originálním znakem

(a) originální znak [2]



(b) vygenerovaný znak

Obrázek D.5: Srovnání vygenerovaného velkého českého státního znaku s originálním znakem

Bibliografie

1. BUBEN, Milan. *Encyklopedie heraldiky*. 1997.
2. POSLANECKÁ SNĚMOVNA, České republiky. *Registr komunálních symbolů*. 2024. Dostupné také z: https://rekos.psp.cz/. [cit. 2024-04-10].
3. ŠŤÁVA, Oto. Automatic Blazon compiler. 2023. Dostupné z DOI: 10.13140/RG.2.2.22666.64969.
4. STRAKOVÁ, Jana; STRAKA, Milan; HAJIČ, Jan. Open-Source Tools for Morphology, Lemmatization, POS Tagging and Named Entity Recognition. In: *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Baltimore, Maryland: Association for Computational Linguistics, 2014, s. 13–18. Dostupné také z: http://www.aclweb.org/anthology/P/P14/P14-5003.pdf.
5. TEAM, TypeScript. *TypeScript*. 2023. Dostupné také z: https://www.typescriptlang.org/. [cit. 2024-04-10].
6. *React*. 2024. Dostupné také z: https://react.dev/. [cit. 2024-04-10].
7. COMMONS, Wikimedia. *WikiProject Heraldry and vexillology*. 2009. Dostupné také z: https://commons.wikimedia.org/wiki/Commons:WikiProject_Heraldry_and_vexillology. [cit. 2024-04-10].
8. FOUNDATION, Mozilla. *SVG Tutorial, Introduction*. 2023. Dostupné také z: https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorial/Introduction. [cit. 2024-04-10].
9. *React*. 2024. Dostupné také z: https://github.com/facebook/react?tab=readme-ov-file. [cit. 2024-04-10].
10. MIKSU, Vojtech. *React - Úvod*. 2024. Dostupné také z: https://www.dzejes.cz/react-uvod.html. [cit. 2024-04-10].
11. ROSSBERG, Andreas (ed.). *WebAssembly Core Specification*. 2022-04-19. Version 2.0. W3C. Dostupné také z: https://www.w3.org/TR/wasm-core-2/.

12. DOCS., MDN Web. *Understanding WebAssembly text format*. 2024. Dostupné také z: https://developer.mozilla.org/en-US/docs/WebAssembly/Understanding_the_text_format. [cit. 2024-04-10].
13. CONTRIBUTORS, Emscripten. *About Emscripten*. 2024. Dostupné také z: https://emscripten.org/docs/introducing_emscripten/about_emscripten.html. [cit. 2024-04-10].
14. LLVM.ORG. *The LLVM Compiler Infrastructure*. 2024. Dostupné také z: https://llvm.org/. [cit. 2024-04-10].
15. WEBASSEMBLY.ORG. *Understanding the JS API*. 2024. Dostupné také z: https://webassembly.org/getting-started/js-api/. [cit. 2024-04-10].
16. SELVATICI, Marco. *WebAssembly Tutorial*. 2020. Dostupné také z: https://marcoselvatici.github.io/WASM_tutorial/. [cit. 2024-04-10].
17. EMSCRIPTEN. *Tools Reference*. 2015. Dostupné také z: https://emscripten.org/docs/tools_reference/index.html. [cit. 2024-04-10].
18. SURMA. *Compiling C to WebAssembly without Emscripten*. 2019. Dostupné také z: https://surma.dev/things/c-to-webassembly/index.html. [cit. 2024-04-10].
19. EMSCRIPTEN. *Emscripten Runtime Environment*. 2015. Dostupné také z: https://emscripten.org/docs/porting/emscripten-runtime-environment.html. [cit. 2024-04-10].
20. EMSCRIPTEN. *API Reference*. 2015. Dostupné také z: https://emscripten.org/docs/api_reference/index.html. [cit. 2024-04-10].
21. EMSCRIPTEN. *Module object*. 2015. Dostupné také z: https://emscripten.org/docs/api_reference/module.html. [cit. 2024-04-10].
22. CHEN, Bobbie. *React C/C++ WASM demo*. 2022. Dostupné také z: https://github.com/bobbiec/react-wasm-demo. [cit. 2024-04-10].
23. EMSCRIPTEN. *Emscripten Compiler Frontend (emcc)*. 2015. Dostupné také z: https://emscripten.org/docs/tools_reference/emcc.html.

Seznam obrázků

Seznam výpisů

Seznam zkratek

CRUD (Create, Read, Update, Delete)

DOM Document Object Model

ES6 ECMAScript 2015

HTTP Hypertext Transfer Protocol

JSON JavaScript Object Notation

JSA Jazyk symbolických adres

MorphoDita Morphological Dictionary and Tagger

REKOS Registr komunálních symbolů

SVG Scalable Vector Graphics

W3C World Wide Web Consortium

WASM WebAssembly

XML Extensible Markup Language