VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

VEKTOROVÝ GRAFICKÝ VÝSTUP Z HTML RENDEROVACÍHO STROJE

VECTOR GRAPHICAL OUTPUT FROM AN HTML RENDERING ENGINE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. TOMÁŠ CHOCHOLATÝ**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. RADEK BURGET, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2020/2021

Zadání diplomové práce

Student:	**Chocholatý Tomáš, Bc.**
Program:	Informační technologie
Obor:	Informační systémy a databáze
Název:	**Vektorový grafický výstup z HTML renderovacího stroje** **Vector Graphics Output from an HTML Rendering Engine**
Kategorie:	Web

Zadání:

1. Prostudujte grafické vektorové formáty SVG a PDF a knihovny pro jejich generování na platformě Java.
2. Seznamte se s existujícím renderovacím jádrem CSSBox a možnostmi generování SVG a PDF výstupu.
3. Navrhněte unifikované řešení pro generování vektorového grafického výstupu z knihovny CSSBox nezávisle na konkrétním výstupním formátu.
4. Implementujte navržené řešení a využijte je pro jednotnou implementaci výstupu ve formátech SVG a PDF.
5. Proveďte testování implementovaného řešení na reálných webových stránkách.
6. Zhodnoťte dosažené výsledky.

Literatura:

- Projekt CSSBox: https://github.com/radkovo/CSSBox
- CSSBox PDF: https://github.com/radkovo/CSSBoxPdf
- CSSBox SVG: https://github.com/radkovo/CSSBoxSvg
- Nguyen, H. D. Transformace webových stránek do vektorové grafiky. Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií.
- Šafář, M.. Transformace dokumentů HTML na vektorovou grafiku SVG. Brno, 2016. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií.
- Červinka, Z.: Generování PDF dokumentů z webových stránek, bakalářská práce, Brno, FIT VUT v Brně, 2015
- Scalable Vector Graphics, The Wold Wide Web Consortium, http://www.w3.org/Graphics/SVG/

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Burget Radek, Ing., Ph.D.**
Vedoucí ústavu:	Kolář Dušan, doc. Dr. Ing.
Datum zadání:	1. listopadu 2020
Datum odevzdání:	19. května 2021
Datum schválení:	22. října 2020

Abstrakt

Tato práce se věnuje problematice vykreslení webových stránek pomocí vektorové grafiky. Je zde popsán experimentální zobrazovací stroj CSSBox a jeho již existující knihovny pro vykreslení vektorové grafiky v PDF a SVG. Cílem je navrhnout společnou strukturu pro tyto dvě knihovny a co nejvíce sjednotit postup vykreslování webových stránek ve vektorové grafice. V práci je proveden rozbor chybně implementovaných částí existujících řešení a popsány nedostatky, které bude nezbytné doimplementovat tak, aby výsledná vektorová grafika splňovala standard CSS3. Dále zde bude popsán proces implementace včetně opravy všech nefungujících původních řešení a bude popsán princip sjednocení jednotlivých části pro generování obou dvou vektorových formátů. V závěru práce budou zhodnoceny výsledky vlastního testování a výstupy z generování reálných webových stránek.

Abstract

This thesis focuses on the issue of rendering web pages using vector graphics. The experimental CSSBox display engine and existing libraries for rendering vector graphics in PDF and SVG will be described here. The goal of the thesis is design a common structure for these two libraries and unify the process of rendering web pages in vector graphics as much as possible. Analysis of incorrectly implemented parts of existing solutions will be performed here and shortcomings, which will be necessary to implement to the resulting vector graphic meet the standard CSS3, will be describe. Furthermore, the implementation process, including the repair of all non-functioning original solutions, and the principle of unification of individual parts for the generation of both vector formats will be described. The conclusion is dedicated the results of self-testing and outputs from generating real websites.

Klíčová slova

CSSBox, JStyleParser, Vektorová grafika, PDF, SVG, CSS3, Java, Apache Batik, Apache PDFBox

Keywords

CSSBox, JStyleParser, Vector graphics, PDF, SVG, CSS3, Java, Apache Batik, Apache PDFBox

Citace

CHOCHOLATÝ, Tomáš. *Vektorový grafický výstup z HTML renderovacího stroje*. Brno, 2021. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Radek Burget, Ph.D.

Vektorový grafický výstup z HTML renderovacího stroje

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Radka Burgeta Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Tomáš Chocholatý
15. května 2021

Poděkování

Rád bych poděkoval svému vedoucímu práce Ing. Radkovi Burgetovi, Ph.D za veškerou pomoc během práce, cenné rady a vstřícnost při konzultacích.

Obsah

Kapitola 1

Úvod

Tato práce popisuje generování vektorového výstupu z HTML renderovacího stroje. V současné době stále narůstá množství webových stránek. Společně s tím vznikají i požadavky na možnost uložení webových stránek a požadavky na přístup k nim bez připojení k internetu nebo možnost převedení webové stránky na formát pro tisk. K tomuto účelu je vhodná právě vektorová grafika, která neztrácí kvalitu zobrazované stránky při změně velikosti. Export do PDF umožňuje například přípravu stránek pro tisk. Formát SVG je vhodný na další případné editace výstupu v grafických editorech. Cílem této práce je převádět webové stránky do těchto dvou formátů.

V této práci bude popsáno rozšíření pro již existující experimentální projekt CSSBox, který je používaný pro testování algoritmů pro analýzu webových stránek. Generování vektorového výstupu z tohoto nástroje již bylo implementováno v předchozích bakalářských a diplomových pracích. [18] [7] [19]

Existující řešení používají různé přístupy pro generování vektorového výstupu. Ukázalo se, že by bylo možné obě řešení spojit, přičemž velká část generování by mohla být společná pro oba dva přístupy. V aktuálním stavu mají obě dvě řešení nevyhovující strukturu implementace, která je nepřehledná, neúplná a v některých případech také chybná.

Cílem této práce bude proto navrhnout co nejvíce společnou strukturu pro generování vektorové grafiky do formátů PDF a SVG a zlepšit celkovou strukturu tohoto projektu. Dalším z cílů je korekce všech chybně implementovaných funkcí, které generují nesprávný výstup a doplnění nových funkcí tak, aby byla zachována specifikace CSS3.

Kapitola 2

Počítačová grafika

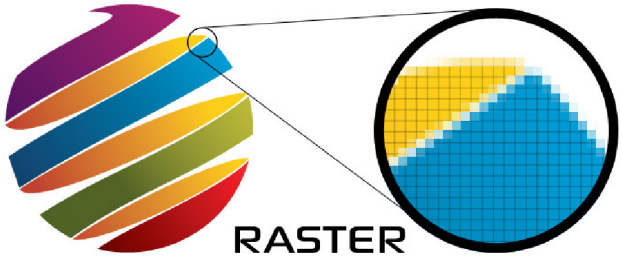
Digitální grafické soubory jsou v počítačové grafice reprezentovány dvěma základními způsoby. Jedná se o rastrovou a vektorovou grafiku. V této kapitole bude popsán princip reprezentace obrázků u rastrové a vektorové grafiky, výhody a nevýhody obou způsobů reprezentace a dva z nejvýznamnějších formátů vektorové grafiky.

2.1 Popis rastrové a vektorové grafiky

Rastrová grafika je tvořena mnoha malými čtverečky zvanými pixely. [5] [14] Tyto pixely jsou uspořádány do mřížky neboli bitmapy. Každý pixel je možné identifikovat na základě jeho souřadnic, které určují pozici pixelu v mřížce. Každý pixel obsahuje mimo jiné informace o barvě či odstínu. Protože většina moderních zobrazovacích zařízení je také rastrová, zobrazení obrázku vyžaduje pouze dekomprimaci a přenesení na obrazovku.

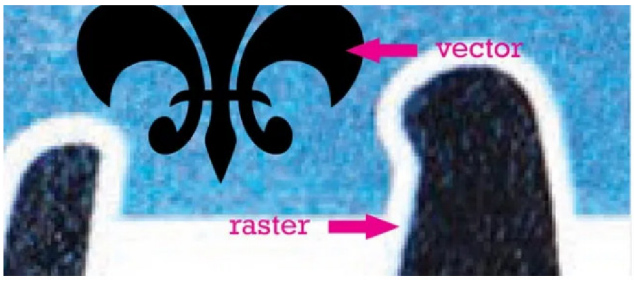
Rozlišení obrázku určuje počet pixelů, které obsahuje. Čím vyšší je rozlišení, tím více pixelů obrázek obsahuje a tím více detailů je v obrázku zachyceno. Rozlišení obrázku se udává na základě počtu pixelů, které upravují šířku a výšku obrázku. Kvalitu obrázku také udává hodnota PPI (Pixel per inch), která informuje o tom, kolik pixelů je zobrazeno na ploše o velikosti jednoho palce. Tato reprezentace obrázků pomocí pixelů má jednu hlavní nevýhodu a tou je degradace obrazu při změně jeho velikosti. [6] Pokud budeme obrázek zvětšovat, budou čáry a okraje nepřesné a zubaté, jelikož po přiblížení uvidíme jednotlivé pixely, které tvoří obraz. Toto je znázorněno na obrázku 2.1. Tento jev se dá zmírnit metodou interpolace, která zajistí, že neuvidíme jednotlivé pixely. Výsledný obraz bude ale rozmazaný a nebude mít tak ostré kontury jako originál. Při opačném procesu zmenšování vzniká problém, kdy jsou obrázky méně ostré, protože zmenšování obrázku poškozuje jeho kvalitu.

Vektorová grafika je na rozdíl od rastrové grafiky založena na matematických výrazech, které definují primitivní geometrické objekty, jako jsou čáry, křivky, kruhy a mnohoúhelníky. Má velkou výhodu oproti rastrové grafice. Lze ji škálovat bez ztráty kvality. Vytváří ostrý a jasný okraj i při úpravě velikosti obrázku. Jak je možné vidět na obrázku 2.2, vektorová grafika je schopna se nekonečně zvětšovat nebo zmenšovat bez ztráty kvality, takže si obraz zachovává svou ostrost. Oproti tomu zvětšení rastrového obrazu povede k rozmazání obrazu. Díky tomu, že je obraz popsán pomocí matematických výrazů, lze snadněji analyzovat obsah nebo pracovat s prvkem nezávisle na ostatních. Na rozdíl od rastrové grafiky však vektorová grafika obvykle nedosahuje realismu fotografií. Vektorová grafika je mnohem všestrannější a flexibilnější. Vektorové obrázky jsou rychle a dokonale škálovatelné, což je výhodou oproti



Obrázek 2.1: Problém degradace kvality rastrových obrázků při zvětšování. [6]

rastrové grafice. Rovněž velkou výhodou vektorové grafiky je, že nemá žádné pevně dané rozlišení. Zobrazení se přizpůsobí jakémukoli výstupnímu zařízení, které jí vykresluje. Protože si vektorová grafika nemusí pamatovat údaje pro miliony pixelů, bývají tyto soubory menší. Pokud bude ale vykreslovaný objekt velmi složitý s velkým počtem vektorových elementů, může naopak jejich velikost přesáhnout i velikost rastrové reprezentace. Nevýhodou vektorové grafiky pak může být velikost výsledného souboru u složité grafiky a náročnost na výpočetní zdroje počítače. Vytvoření vektorové grafiky je na rozdíl od rastrové grafiky náročnější.



Obrázek 2.2: Ukázka výhody ostrosti vektorové grafiky při změně velikosti obrázku. [14]

Převádění vektorové grafiky do rastrové je jednodušší. K tomuto procesu nám stačí pouze proces rasterizace. Vektorizace, převod rastrové reprezentace do vektorové, je mnohem složitější, protože se musí jednotlivé objekty analyzovat a pak se musí matematicky popsat.

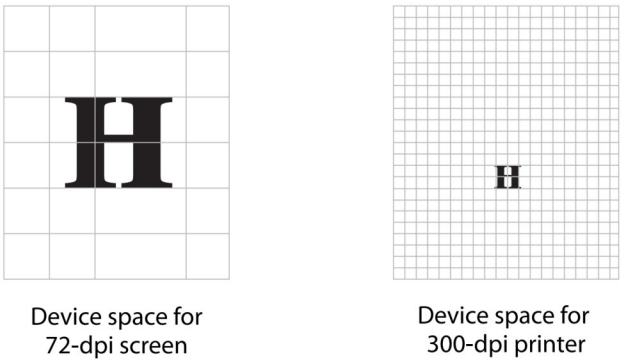
2.2 Dvojdimenzionální souřadnicové systémy

S ohledem na fakt, že oba formáty, kterými se bude tato práce zabývat, používají různé souřadnicové systémy, bude vhodné je v této kapitole popsat. Budou zde popsány dva dvojdimenzionální souřadnicové systémy *user space* a *device space* a jejich hlavní rozdíly v používání.

2.2.1 Souřadnicový systém device space

Tento souřadnicový systém je závislý na výstupním zařízení, na kterém je zobrazován. Zařízení se mohou značně lišit v integrovaných souřadných systémech, které používají k adresování pixelů v jejich zobrazovatelných oblastech. [1] Různá zařízení mohou mít různá

rozlišení. Některé mohou mít různé rozlišení ve vertikálním a v horizontálním směru. Také prostory zařízení mohou být orientovány odlišně. Například hodnoty souřadnice y se mohou na některých zařízeních zvyšovat od horní části stránky a na některých od spodní části stránky. Tyto všechny vlastnosti vedou k tomu, že zobrazovaný objekt se může na různých zařízeních zobrazovat odlišně. Na obrázku 2.3 je vidět, že pokud jsou souřadnice zadány v souřadném systému *device space*, jsou závislé na zařízení, na kterém se zobrazují. Obrázek ukazuje, jak stejný objekt, vykreslovaný pouze na jiném zařízení, může vypadat odlišně.



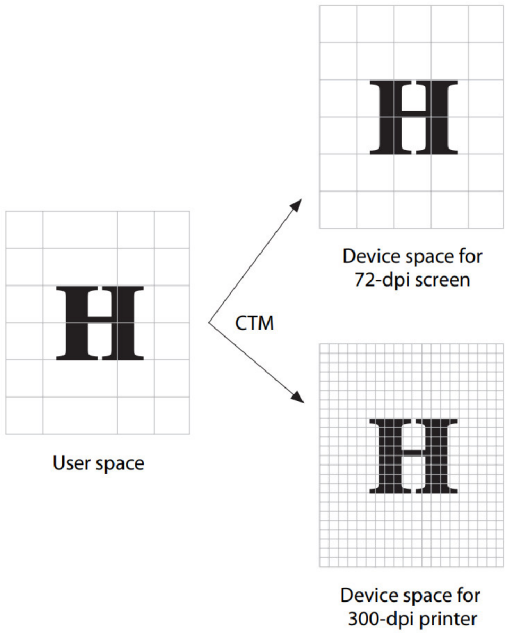
Obrázek 2.3: Zobrazení objektu pomocí device space. [1]

2.2.2 Souřadnicový systém user space

Hlavním úkolem souřadnicového systému *user space* je vyhnout se závislostem na zařízeních. [1] Tento souřadnicový systém je používán v PDF. Definuje nezávislý souřadnicový systém na zařízení, který zaručí, že objekt bude zobrazen stejně, bez ohledu na výstupní zařízení, na kterém je vykreslován. Souřadnicový systém *user space* je inicializován pro každou stránku dokumentu. Kladné hodnoty osy x se zvyšují směrem doprava a kladné hodnoty osy y svisle ve směru nahoru od konce stránky. Délka jednotky na ose x a y je nastavena položkou *UserUnit* ve slovníku stránky. Pokud tato hodnota není k dispozici, použije se výchozí hodnota 1/72. Souřadnicový systém *user space* je v podstatě nekonečná rovina a pouze malá část této nekonečné roviny se zobrazí na výstupním zařízení. Této oblasti se říká *CropBox* a je definována ve slovníku stránky. Transformaci z uživatelského prostoru do prostoru zařízení definuje *current transformation matrix* (CTM). Aplikace PDF může upravit *CTM* pro nativní rozlišení konkrétního výstupního zařízení a zajistit tak nezávislost zařízení na popisu stránky PDF. Na obrázku 2.4 je ukázka vykreslení objektu nezávisle na vlastnostech výstupního zařízení. To, co se může zdát jako absolutní souřadnice v obsahu stream, není absolutní vzhledem k aktuální stránce, protože jsou vyjádřeny v souřadnicovém systému, který se může posouvat a zmenšovat nebo rozšiřovat.

2.3 Formát SVG

Scalable Vector Graphic (SVG) je vektorový formát, který slouží pro popis dvourozměrné grafiky. [16] [8] Kromě základního kreslení podporuje i animace, a dokonce i interakce (např. hypertextové odkazy). SVG je založený na Extensible Modeling Language (XML). Informace o obrázku se ukládají jako prostý text a přinášejí výhody otevřenosti XML, přenositelnosti a interoperability. Tento formát je navržený a vyvinutý jako standard společností



Obrázek 2.4: Zobrazení objektu pomocí user space. [1]

World Wide Web Consortium (W3C). SVG vychází z předchůdců jako Vectro Markup language a Precision Graphics Markup Language (PGML). Výhodou SVG je, že jej lze vytvářet a upravovat pomocí libovolného textového editoru, protože SVG je založený na XML.

Formát SVG má několik prvků v podobě tvaru. Mezi nejzákladnější patří cesta skládající se z přímek a křivek, kruh, obdélník, text, obrázek atd. Grafické objekty lze seskupovat, stylovat, transformovat a skládat do dříve vykreslených objektů. Lze také přidat animaci, filtry a přechody.

Výhodou je, že velikost souborů SVG je podstatně menší a lze je komprimovat. Je možné jej rovněž tisknout ve vysoké a konzistentní kvalitě v mnoha různých rozlišeních. Obrázek nebo fotografii v SVG lze zvětšovat, což znamená, že je můžeme přiblížit bez ztráty kvality.

Aktuální verzí SVG, která se dnes používá, je verze SVG 1.1. V dnešní době podporuje SVG většina používaných prohlížečů. Se zobrazením SVG má problémy pouze Internet Explorer 8 a jeho starší verze, které soubory SVG nepodporují. Prohlížeče se liší pouze v rychlosti zpracování SVG a v podpoře zobrazení složitějších efektů. Pro nepodporované prohlížeče je tu možnost definovat alternativní textový obsah. Dalším alternativním způsobem může být vložení SVG přímo do HTML souboru do tagu `<svg>`, jelikož HTML5 je rovněž založeno na XML. SVG je formát obrázku a lze ho tedy zahrnout do stránek HTML jako i jiné obrázky. Existují dva přístupy. Obrázek může být vložen do HTML pomocí tagu <img>, nebo jej lze vložit jako vlastnost stylu jiného prvku.

2.3.1 Struktura dokumentu SVG

Ukázku struktury SVG dokumentu lze vidět na obrázku 2.5. SVG dokument začíná standardní instrukcí pro zpracování XML a deklarací `DOCTYPE`, což je deklarace typu dokumentu. Kořenový prvek `<svg>` definuje šířku a výšku konečné grafiky v pixelech. Rovněž definuje obor názvů SVG prostřednictvím atributu `xmlns`. Prvek `<title>` je k dispozici prohlížečům pro použití v záhlaví a prvek <desc> obsahuje celý popis obrázku.

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">

<svg width="140" height="170"
  xmlns="http://www.w3.org/2000/svg">
<title>Cat</title>
<desc>Stick Figure of a Cat</desc>
<!-- the drawing will go here -->
</svg>
```

Obrázek 2.5: Ukázková struktura SVG dokumentu. [16]

Než budou popsány jednotlivé prvky jazyka, je vhodné objasnit několik základních výrazů.

Svět SVG je nekonečné plátno (Canvas), na které se vykresluje SVG kód. Tato plocha je však ve skutečnosti omezená výřezem (Viewport). Je to oblast plátna, kterou dokument může využít. Jedná se o plochu, na které uživatel vidí vykreslené elementy SVG. Velikost tohoto výřezu je určena samotným SVG dokumentem a nachází se v prvku `<svg>`. Pokud jsou některé elementy SVG vykresleny mimo tento výřez, nebudou ve výsledném dokumentu vidět. Velikost rozměrů může být v SVG definována několika způsoby:

- em - jednotka relativní k velikosti písma (2em znamená dvakrát větší než velikost písma),
- ex - jednotka relativní k výšce písma (2ex znamená dvakrát větší než výška písma),
- px - velikost v pixelech,
- pt - points (1/72 palce),
- pc - picas (1/6 palce),
- cm - centimetry,
- mm - milimetry,
- in - inches (palce),
- % - procenta.

SVG používá souřadný systém, kdy v levém horním rohu jsou nastaveny hodnoty souřadnic `x` a `y` na 0. Tento bod se zapisuje jako (0,0) a nazývá se počátek. Hodnoty osy `x` se směrem doprava zvyšují a hodnoty osy `y` se zvyšují směrem dolů.

Jedním z cílů XML je poskytnout způsob strukturování dat a oddělení této struktury od vizuální prezentace. SVG stejně jako HTML soubory podporuje stylování pomocí CSS (pouze do verze CSS2). SVG umožňuje stylovat prvky čtyřmi způsoby:

- vložením definice stylu k příslušnému SVG prvku,
- pomocí interní šablony stylů - každému prvku se nemusí nastavovat styl. Vytvoří se seznam stylů, které se použijí na všechny výskyty dokumentu.

- pomocí externí šablony stylů - pokud se má sada stylů používat na více SVG dokumentech. Někdy je potřeba změnit styl globálně pro všechny dokumenty. Proto je lepší některé vlastnosti vložit do společné externí šablony pro styly.
- atributy prezentace - CSS vlastnost lze použít jako atributy prvků SVG.

2.3.2 Základní prvky jazyka SVG

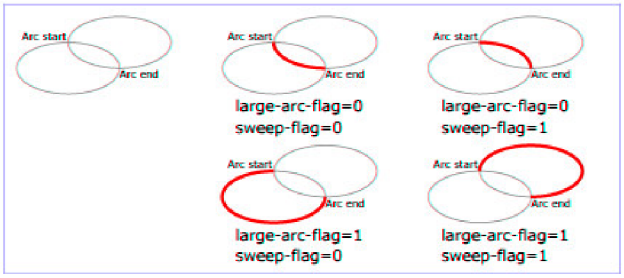
V této kapitole budou podrobněji popsány základní grafické elementy jazyka SVG, které budou důležité pro pozdější vykreslování výsledného SVG souboru.

Cesta (path)

Všechny základní tvary jsou ve skutečnosti vytvořeny pomocí obecnějšího prvku path. Používání přímo základních prvků (např. obdélník) místo cest zvyšuje čitelnost kódu. Díky prvku path můžeme nakreslit objekt libovolného tvaru. Tento element obsahuje hlavní atribut *d* a *path data*, která nám popisují výsledný tvar cesty. Atribut tvoří libovolná sekvence příkazů definující tvar. Příkazy, které můžeme použít v atributu *d*, jsou:

- **M x y (moveto)** - každá cesta musí začínat příkazem `moveto`. Jedná se o velké písmeno M následované souřadnicemi x a y. Jde o počáteční bod každé cesty. Tento bod v podstatě představuje bod položení pera na papír.
- **L x y (lineto)** - za příkazem `moveto` následuje jeden či více příkazů `lineto`. Jedná se o velké písmeno L následované souřadnicemi x a y. Tento příkaz vykreslí úsečku z posledního bodu do bodu definovaného souřadnicemi x a y. Tento příkaz má ještě dvě varianty, a to konkrétně `Horizontal lineto` (H) a `Vertikal lineto` (V). Jedná se o vykreslení horizontální, respektive vertikální úsečky. Tyto příkazy mají pouze jeden parametr, a to parametr x nebo y.
- **Z x y (closepath)** - tento příkaz slouží k uzavření cesty. Tento příkaz nakreslí přímku z aktuálního bodu zpět do počátečního bodu. Jedná se o velké písmeno Z následované souřadnicemi x a y.
- **A rx ry x-axis-rotation large-arc-flag sweep-flag x y (elliptical arc)** - tento příkaz nakreslí eliptický oblouk z aktuálního bodu do bodu *(x,y)*. Velikost a orientace elipsy je definovaná dvěma poloměry (rx, ry) a rotací osy x, což naznačuje, jak se elipsa jako celek otáčí vzhledem k aktuálnímu souřadnicovému systému. Střed *(cx, cy)* elipsy se vypočítá automaticky, aby vyhovoval omezením uloženým ostatními parametry. Přepínače `big-arc-flag` a `sweep-flag` určují, jaká část elipsy se vykreslí. Použití těchto dvou přepínačů je znázorněno na obrázku 2.6.
- **Kubické a kvadratické Beziérovy křivky** - další alternativa, jak vykreslit křivky. Pomocí těchto křivek se dají vykreslit jakékoli tvary. Pro tyto křivky slouží příkazy C,S,Q a T. V této práci jsou v rámci SVG používány především eliptické křivky, proto jsou Beziérovy křivky v této části pouze zmíněny a nebude jim zde věnována další pozornost.

Všechny výše zmíněné příkazy mají dvě varianty, a to s velkým a malým písmenem. Velké písmeno znamená, že se používají absolutní souřadnice, kdežto malé písmeno znamená, že se používají souřadnice relativní. Kvůli přehlednosti výsledného SVG dokumentu budou v této práci používány pouze absolutní souřadnice.



Obrázek 2.6: Ukázka použití přepínačů `big-arc-flag` a `sweep-flag` u eliptického oblouku. [3]

Obdélníky (rectangles)

Prvek obdélník je v této práci jeden z nejdůležitějších prvků s ohledem na fakt, že nástroj CSSBox je založen na obdélnících.

```
\texttt{<rect x="10" y="10" width="30" height="50" style="fill:
    \#0000ff;"/>}
```

Tento SVG element definuje obdélník, který bude vyplněn modrou barvou a bude mít šířku 30px a výšku 50px a jeho levý horní bod bude umístěn na souřadnicích (10,10). Atributy x a y definují počáteční bod vykreslování obdélníku. Atribut *width* definuje jeho šířku a atribut *height* definuje jeho výšku. Obdélník může obsahovat atribut *style*, který definuje vizuální styl obdélníku. Díky tomuto atributu může nastavovat například barvu výplně (*fill*), šířku a barvu okraje (*stroke*) a průhlednost elementu (*opacity*).

Gradienty (gradients)

Jedná se o barevné přechody, které se dají použít místo vyplnění objektu plnou barvou. Přechody stejně jako v CSS mohou být lineární, kde k přechodu odstínů dochází na základě přímky, nebo radiální, kde k přechodu dochází za pomoci kružnic směrem ven ze středového bodu. Gradienty jsou definované pomocí párového SVG elementu `<defs>` a používají se pouze přes odkazy, k čemuž se využívá atribut *id* konkrétního gradientu.

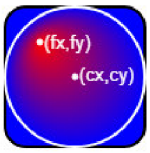
Výchozím chováním lineárního gradientu je přechod z levé strany objektu na pravou. Pokud chceme přechod barev přes svislou osu objektu nebo pod nějakým úhlem, musí se nastavit počáteční a koncový bod čáry. Definici lineárního gradientu a změnu jeho směru zprava doleva můžeme vidět na příkladu níže. Body se vyjadřují většinou pomocí procent, nebo jako desetinná čísla v rozsahu od 0 do 1. Důvodem je, že při definici gradientu nevíme, na které všechny elementy bude gradient použit.

```
<defs>
    <linearGradient id="three_stops">
        <stop offset="0%" style="stop-color: #ffcc00;"/>
        <stop offset="33.3%" style="stop-color: #cc6699"/>
        <stop offset="100%" style="stop-color: #66cc99;"/>
    </linearGradient>

    <linearGradient id="right_to_left"
        xlink:href="#three_stops"
```

```
        x1="100%" y1="0%" x2="0%" y2="0%"/>
</defs>
```

Parametry lineárního gradientu jsou dva body v souřadném systému, které určují směr gradientu. Radiální gradient je definován rovněž dvěma body. První z nich definuje kruh, kolem kterého přechod končí. Vyžaduje středový bod, který je určen atributy *cx* a *cy* a poloměr *r*. Druhý bod se nazývá ohnisko a je definován atributy *fx* a *fy*. Ohnisko definuje, kde je střed přechodu. To je snadněji pochopitelné z obrázku 2.7



Obrázek 2.7: Nastavení radiálního gradientu.

2.4 Formát PDF

Portable Document Format (PDF) je nativní formát od společnosti Adobe. [1] [2] Tento formát spadá do rodiny protokolů Adobe Acrobat. Tyto produkty mají za cíl umožnit uživatelům snadnou a spolehlivou výměnu a prohlížení elektronických dokumentů bez ohledu na prostředí, ve kterém byly vytvořeny. Bez ohledu na operační systém, zařízení nebo rozlišení výstupního zařízení bude všude vypadat PDF dokument identicky. Stránky mohou obsahovat libovolnou kombinaci textu, grafiky a obrázků. Vzhled stránek je popsán pomocí tzv. *content stream*, který obsahuje posloupnosti grafických objektů, jenž mají být vykresleny na stránku. Tento vzhled je plně specifikován a veškeré rozvržení a formátování již bylo provedeno aplikací generující proud obsahu.

Kromě popisu statického vzhledu stránek mohou být obsahem PDF dokumentu také interaktivní prvky. Ty existují pouze v elektronické podobě. Jedná se například o hypertextové odkazy, zvuky a videa. Může také obsahovat akci na základě triggeru, uživatelského rozhraní nebo může obsahovat formuláře pro zadávání informací, které je možné následně exportovat.

Dokument PDF je možné zašifrovat za pomoci hesla. Běžně se používají dva typy hesel, *user password* a *owner password*. Pomocí hesla *owner-password* získá uživatel plný přístup k souboru a může měnit i heslo. Pomocí hesla *user-password* může uživatel vykonávat pouze některé činnosti povolené autorem práce.

2.4.1 Struktura PDF dokumentu

PDF dokument je ve tvaru textového souboru a skládá se z operátorů a operandů jazyka PDF. Obsah PDF se dá zobrazit pomocí aplikace *Adobe Acrobat Reader* nebo pomocí webového prohlížeče. Aplikace `Adobe Acrobat Reader` umožňuje navíc upravování, komentování a podepisování PDF dokumentu.

Soubor PDF se skládá ze čtyř prvků:

- Jednořádkové záhlaví identifikující verzi specifikace PDF (např. *%PDF-1.7*).
- Tělo obsahující objekty, které tvoří dokument. Jedná se například o stránky nebo typ písma.

- Tabulka křížových odkazů obsahující informace o nepřímých objektech v souboru.
- Patička udávající umístění tabulky křížových odkazů a určitých zvláštních objektů v těle souboru.

Počáteční strukturu lze upravit pozdějšími aktualizacemi, které na konec souboru připojí další prvky.

Dokument PDF lze považovat za hierarchii objektů. Kořen hierarchie je slovník katalogu dokumentů. Většina objektů v hierarchii jsou slovníky. Například každá stránka dokumentu je reprezentována objektem stránky - slovníkem, který obsahuje odkazy na obsah stránek a další atributy. Jednotlivé objekty stránky jsou svázány dohromady ve struktuře zvané strom stránek, který je zase určen nepřímým odkazem v katalogu dokumentů. Vztahy rodičů a potomků jsou v hierarchii definovány položkami slovníku, jejichž hodnoty jsou nepřímými odkazy na jiné slovníky.

2.4.2 Základní prvky grafiky jazyka PDF

Grafické operátory použité v *PDF content stream* popisují vzhled stránky, která se bude následně reprodukovat na rastrovém výstupním zařízení. Tyto operátory lze rozdělit do šesti skupin:

- **Graphics state operators** - manipulují s datovou strukturou zvanou *graphics state*. Ta zahrnuje aktuální transformační matici mapující souřadnice uživatelského prostoru, aktuální barvu, aktuální ořezovou cestu a další parametry.
- **Path contructions operators** - určují cesty, pomocí kterých se definují další tvary vektorové grafiky.
- **Path-painting operators** - operátory, jenž vyplňují cesty barvou. Obarví oblast v blízkosti cesty, nebo ji použije jako hranici ořezu.
- **Other painting operators** - vybarvují grafické objekty, jako například obrázky, stínování apod.
- **Text operators** - zobrazují pouze textové prvky. Znaky jsou reprezentovány *glyphy*. Glyph je grafický tvar, na který se dají použít všechny grafické manipulace.
- **Marked-content operators** - přidružují logické informace vyšší úrovně k objektům v *content stream*.

Operandy a operátory se zapisují postupně pomocí postfixové notace. Jedná se spíše o statický popis posloupnosti grafických objektů. `PDF` poskytuje pět typů grafických objektů:

- **Path object** - je to libovolný tvar složený z přímek, obdélníků a kubických Bézierových křivek. Objekt cesty je ukončen jedním nebo více operátory, které určují šířku cesty, zda je cesta vyplněná, nebo zda je použita hranice pro ořez.
- **Text object** - textový objekt se skládá z jednoho nebo více řetězců znaků, identifikujících sekvenci glyfů, které mají být vykresleny. Mohou obsahovat stejné informace jako má objekt path, například výplň či hranice ořezu.

- **External object** - je grafický objekt, jehož obsah je definovaný mimo *content stream*. Existují tři typy externích objektů: Image XObject, Form XObject a PostScriptXobjekt. XObjekt jako image představuje rastrový obrázek. XObject jako formulář je samostatný popis libovolné sekvence a PostScript XObject obsahuje fragment kódu v jazyce PostScript.
- **Inline image** - vyjadřuje data malého obrázku přímo v rámci *content stream*.
- **Shading object** - popisuje geometrický tvar, jehož barva je libovolná funkce polohy uvnitř útvaru. Tento objekt se v této práci využívá především pro vykreslování lineárního a radiálního gradientu.

Tyto operátory jsou využity v implementaci v kapitole 7, kde je popsáno jejich použití pro generování jednotlivých prvků webových stránek z nástroje CSSBox.

Kapitola 3

Knihovny pro generování vektorové grafiky na platformě Java

Tato kapitola bude věnována popisu a srovnání knihoven na platformě *Java* pro generování *SVG* nebo *PDF*.

3.1 SVG knihovny na platformě Java

Pro práci s SVG v jazyce Java je třeba zvolit vhodnou knihovnu. Diplomová práce *Transformace dokumentů HTML na vektorovou grafiku SVG* [19], na kterou tato práce navazuje, používá pro generování SVG knihovnu Apache Batik. Přestože bude s největší pravděpodobností vhodné využít tuto knihovnu, je žádoucí prostudovat, zda by pro tento účel nebylo vhodnější použít některou z jiných knihoven pro generování SVG. Popis a porovnání jednotlivých knihoven bude popsán níže.

3.1.1 Apache Batik

Apache Batik je *Java* knihovna, která umožňuje vytváření a manipulaci s SVG grafikou. [15] Knihovna usnadňuje aplikacím nebo appletům, založeným na Javě, práci s obsahem SVG. Batik knihovna je rozdělena do několika modulů, které lze použít samostatně nebo společně ke generování obsahu SVG, prohlížení obsahu SVG nebo převodu na rastrový či jiný vektorový formát. K vykreslování SVG výstupu využívá knihovna Batik DOM strom. Nejprve je nutné vytvořit objekt typu SVG dokument, ke kterému se postupně přidávají další objekty. Objekt typu dokument se pak předá knihovně Batik, která ho převede na dokument SVG. Výhoda této knihovny je především v její modularitě a možnosti pozdějšího rozšíření. Jeden z důvodů zvolení této knihovny je, že reprezentuje celý dokument pomocí DOM stromu. Je to velmi podobný princip, který používá projekt CSSBox, pro nějž bude tato knihovna určena.

3.1.2 JFreeSVG

JFreeSVG je rychlá jednoduchá knihovna vektorové grafiky sloužící pro generování grafického výstupu ve formátu SVG přímo z kódu Java. Tato knihovna je oblíbená zejména díky své kompaktní velikosti. Kromě toho, že má malou velikost, disponuje také velkou rychlostí. Ve srovnání s knihovnou Apache Batik obsahuje pouze malou množinu funkcí souvisejících s SVG. Pokud jde o velikost generovaného souboru, je větší právě u této knihovny JFRee-

SVG. Naproti tomu knihovna Batik dosahuje menší velikosti cílového souboru. JFreeSVG je však cca 5x rychlejší než Batik. Jako vhodnější se tak stále jeví knihovna Batik vzhledem na výslednou velikost souboru a použití DOM stromu pro generování výsledného souboru. Rychlost v tomto případě nehraje až tak zásadní roli.

3.2 PDF knihovny na platformě Java

Pro generování PDF v jazyce *Java* bude vhodnější použít nějakou již existující knihovnu. V práci *Generování PDF dokumentů z webových stránek* [18] a v práci *Transformace webových stránek do vektorové grafiky* [7], na které bude tato práce navazovat, je použita pro generování PDF výstupu knihovna *PDFBox*. Předpokladem je, že pro tuto práci bude rovněž použita knihovna *PDFBox*. V této kapitole bude knihovna popsána blíže a bude porovnána s ostatními knihovnami, aby bylo ověřeno, zda je její použití pro tuto práci vhodné.

3.2.1 Apache PDFBox

Apache PDFBox je open source nástroj pro práci s PDF dokumenty na platformě Java. [11] Umožňuje vytváření nových PDF dokumentů, manipulaci s existujícími dokumenty a má možnost extrahovat obsah z dokumentů. Apache PFDBox rovněž obsahuje několik obslužných programů příkazového řádku. Tato knihovna zahrnuje velké množství tříd a funkcí. Zde budou vyjmenovány některé důležité třídy, které budou nejvíce použity v rámci této práce.

- *PDDocument* - je in-memory reprezentace dokumentu PDF.
- *PDPage* - její instance reprezentuje stránku dokumentu.
- *PDPageContentStream* - instance této třídy reprezentuje obsah, do kterého jsou vkládány elementy dokumentu PDF.
- *PDFont* - třída, reprezentující font pro použitý text. Nese v sobě rovněž informace o písmu, například jeho řez apod.

Další třídy, které je vhodné zmínit, jsou třídy z balíku *graphics* obsahující funkce pro práci s grafikou objektu. Všechny tyto třídy jsou třídy z balíku *org.apache.pdfbox.pdmodel* tvořící kostru knihovny PDFBox.

3.2.2 iText

Knihovna iText umožňuje vytváření a manipulaci s dokumenty PDF, a to jak v prostředí *Java*, tak i v prostředí *.NET*. Knihovna iText poskytuje podporu pro nejpokročilejší funkce, například pro podpisy založené na *PKI*, 40 bitové nebo 128 bitové šifrování, korekci barev, formuláře PDF a další. Tato knihovna nabízí řadu nástrojů, které ulehčují práci s PDF dokumentem, a navíc má velmi dobrou dokumentaci, díky níž je práce s touto knihovnou snazší. Oproti knihovně *PDFBox* obsahuje nástroje pro vytvoření tabulky, kterou není potřeba tvořit manuálně. Knihovnu iText lze však bezplatně použít pouze pod licencí *Affero General Public License (AGPL)*. Licence AGPL přichází s několika omezeními. Vyžaduje zveřejnění úplného zdrojového kódu vlastních aplikací, může se používat pouze v prostředí, které je taktéž pod licencí *AGPL*. Každý zásah do knihovny iText musí být zveřejněn včetně všech změn a musí být uvedeno, že aplikace byla vytvořena s pomocí této knihovny.

Toto jsou všechna omezení, která musí být splněna, aby bylo možné tuto knihovnu použít bezplatně. Z tohoto důvodu je pro tento projekt vhodnější knihovna *PDFBox*, která sice nenabízí takové funkce jako iText, i tak je ale dostatečná pro určené použití.

3.2.3 gnujpdf

Je to Java balíček s lícencí LGPL. Poskytuje jednoduché API pro vytváření PDF souborů a tisk pomocí podtříd java.awt.Graphics a java.awt.PrintJob. PDF třídy zapisují do OutputStream v pdf formátu namísto typických grafických objektů, ale volání metod zůstávají stejná. Tato knihovna používá licenci GNU Lesser General Public License (GNU LGPL), která obvykle znamená, že program, který tuto knihovnu využívá, musí mít stejnou licenci, což je pro práci s touto knihovnou nepříjemné omezení.

3.2.4 PDF CLown

PDF Clown je open-source knihovna, která, stejně jako knihovny PDFBox, iText a gnujpdf, dokáže manipulaci a vytváření PDF dokumentů. To probíhá na základě abstrakčních vrstev, které dodržují specifikaci 1.7. Tato knihovna však není průběžně aktualizována. Poslední aktualizace této knihovny proběhla v listopadu 2015.

Na základě průzkumu existujících vhodných knihoven pro jazyk Java se jeví jako nejoptimálnější knihovna *PDFBox*, především díky její licenci na použití a také s ohledem na fakt, že projekt, na který tato práce navazuje, byl vytvořen za pomocí této knihovny.

Kapitola 4

Knihovna CSSBox

Tato práce se bude zabývat rozšířením pro nástroj CSSBox. Z tohoto důvodu bude v této kapitole popsán nástroj CSSBox a jeho vlastnosti.

Dále zde budou popsány již existující implementace pro vykreslování SVG a PDF z nástroje CCSBox a důvody, proč současný stav implementace nevyhovuje a co nefunguje dle očekávání a bude tak muset být opraveno.

4.1 Jádro knihovny CSSBox

CSSBox je (X)HTML/CSS zobrazovací nástroj napsaný čistě v jazyce Java. [4] Jeho primárním účelem je poskytnout úplné a dále zpracovatelné informace o obsahu a rozložení stránky. Je možné ho také použít k procházení vykreslených dokumentů v aplikaci Java Swing. Vstupem zobrazovacího nástroje je strom DOM dokumentu. Zobrazovací nástroj dokáže automaticky načíst šablony stylů, na které odkazuje dokument. Výstupem zobrazovacího nástroje je objektově orientovaný model popisující rozložení stránky. Tento model je možné přímo zobrazit, ale je také vhodný pro další zpracování pomocí algoritmů analýzy rozložení, jako jsou například algoritmy segmentace stránek nebo extrakce informací.

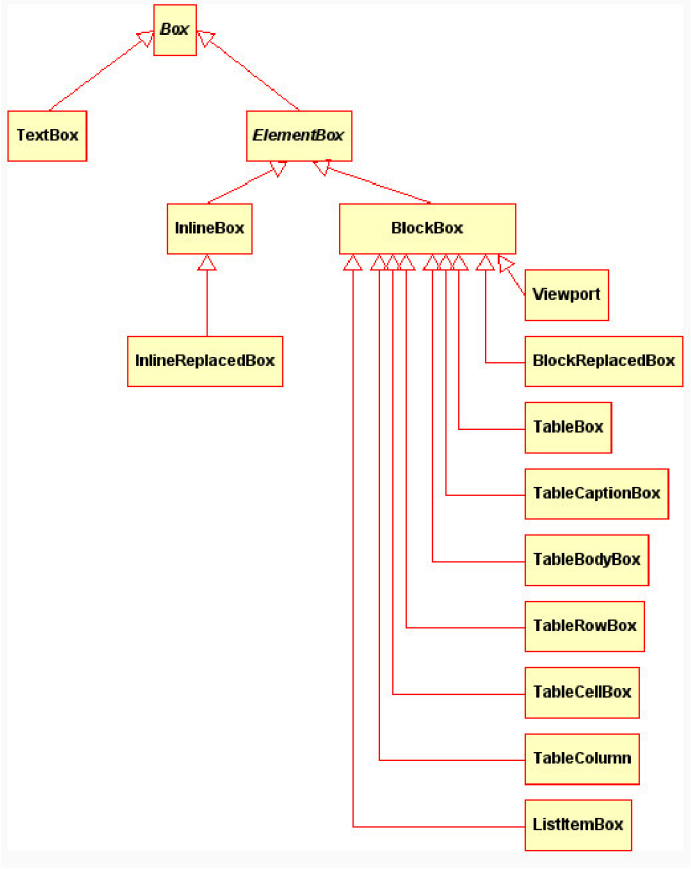
Základní knihovnu CSSBox lze rovněž využít k získání bitmapového nebo vektorového obrazu zobrazovaného dokumentu. Pomocí balíčku SwingBox lze CSSBox také využít jako interaktivní webový prohlížeč.

CSSBox obecně očekává implementaci DOM stromu na jeho vstupu, který je reprezentován jeho kořenovým uzlem. Způsob získání DOM stromu není pro nástroj CSSBox důležitý, ale obvykle se tak děje analýzou souboru HTML nebo XML. Tato analýza je založená na analyzátorech *NekoHTML* a *Xerces2*.

Výsledné rozložení dokumentu je reprezentováno jako strom *boxů*. Každý *box* vytvoří na výsledné stránce obdélníkovou oblast, která odpovídá konkrétnímu vykreslovanému HTML elementu. Může existovat více boxů pro jeden prvek. Například víceřádkový odstavec <*p*> je rozdělen na několik boxů reprezentujících jednotlivé řádky. Rámeček může být buď složen z rámečků, které jsou jeho potomky, nebo může odpovídat určité části obsahu dokumentu, kterou může být například textový řetězec nebo obrázek. Každý *box* je reprezentován objektem, který rozšiřuje abstraktní třídu `Box`. V nástroji CSSBox aktuálně existuje několik typů boxů, které odpovídají vlastnosti CSS display pro daný element.

Na obrázku 4.1 je znázorněna hierarchie boxů, se kterými nástroj CSSBox pracuje. Šipky znázorňují dědičnost. Kořenový uzel stromu boxů je vždy reprezentován objektem *Viewport*, který představuje výřez prohlížeče. Ten má vždy jednoho potomka, který se nazývá `root`.

Box *root* odpovídá kořenovému prvku kódu HTML, který je předán objektu *BrowserCanvas*. Obvykle se jedná o prvek `<body>`.



Obrázek 4.1: Hierarchie typů boxů v nástroji CSSBox. [4]

Vykreslování výstupu

Z celého nástroje CSSBox je pro tuto práci klíčové pochopit, jak funguje vykreslování jednotlivých elementů. V aktuální verzi CSSBoxu generuje rastrový výstup třída *GraphicsRenderer*. Tato třída implementuje rozhraní *BoxRenderer*. Rozhraní deklaruje šest metod, které jsou postupně volány při generování výstupu. Vykreslování elementu začíná voláním metody *startElementContents()*. Před voláním této metody se nevykresluje žádný obsah elementu, ale pomocí metody *renderElementBackground()*, která se volá ještě před metodou *startElementContents()*, může být již vykreslené pozadí nebo rámeček elementu. Metody *renderReplacedContent()* a *renderTextContent* slouží ke generování obsahu elementu. Po vykreslení elementu a všech jeho potomků se zavolá metoda *finishElementContents()*. Poté, co jsou vykresleny všechny elementy, se zavolá metoda *close()*, která ukončí výstup.

4.2 JStyleParser

Java knihovna *JStyleParser* slouží pro analýzu kaskádových stylů a k přiřazení těchto stylů k HTML nebo XML elementům, ke kterým patří. Slouží jako parser CSS stylových souborů

a má vlastní aplikační rozhraní. Stejně jako CSSBox je tento nástroj napsán čistě v jazyce Java. Dokáže analyzovat jednotlivé *CSS* soubory a efektivně provádět výpočet stylů prvků *DOM*. Tato knihovna je součástí projektu CSSBox. Pracuje podle specifikace W3C CSS2.1 a podporuje i velkou část specifikace CSS3.

4.3 Stávající podpora SVG výstupu

Výstup SVG je implementován v rámci knihovny *CSSBoxSVG*. Ta slouží pro generování vektorového výstupu ve formátu SVG. Tento projekt vznikl v rámci diplomové práce, jejímž autorem je Ing. Martin Šafář [19]. Jádrem tohoto projektu je třída `SVGDOMRenderer`. Jedná se o vylepšenou verzi třídy `SVGRenderer`, která je obsažena v nástroji CSSBox. V této kapitole se budeme zabývat především třídou `SVGDOMRenderer`. Tato třída implementuje rozhraní `BoxRenderer`. Třída `SVGDOMRenderer` zpracovává po jednotlivých elementech vstupní HTML dokument a z těchto elementů pak postupně vytváří výsledný SVG dokument. Během analýzy tato třída zpracovává elementy DOM dokumentu. Tyto elementy převede na typ DOM element a postupně, jak jsou elementy zpracovávány, je ukládá do stromové struktury. Poté, co jsou zpracovány všechny elementy ze vstupního DOM dokumentu, proběhne převod do tvaru reprezentujícího SVG soubor. Tato třída provádí generování do SVG elegantnějším způsobem než třída `SVGRenderer`. Ta na rozdíl od třídy `SVGDOMRenderer` generuje příkazy SVG přímo v textové podobě z DOM dokumentu. Třída `SVGDOMRenderer` tudíž poskytuje lepší škálovatelnost a lepší strukturu výsledného SVG dokumentu. Třída `SVGDOMRenderer` také podporuje více CSS3 vlastností, jako například *border-radius*, gradienty, transformace, stínování a průhlednost.

4.4 Stávající podpora PDF výstupu

CSSBoxPdf slouží jako knihovna nástroje CSSBox pro generování vektorového výstupu. Tentokrát se jedná o generování do formátu PDF. Tento projekt vznikl v rámci bakalářské práce Ing. Zbyňka Červinky [18]. Na tuto práci později navázal Bc. Hoang Duong Nguyen v rámci své bakalářské práce [7], který původní projekt rozšířil o některé vlastnosti, jako například border-radius, gradient, stínování apod. Jádrem tohoto projektu je třída `PDFRenderer`. Postupně tak, jak jsou analyzovány a zpracovávány elementy ze vstupního DOM dokumentu, tato třída pro každý takový element vyskytující se v DOM dokumentu vytvoří instanci třídy `Node` a ekvivalentní uzel stromové struktury TREE. Instance třídy `Node` je vložena do seznamu, který reprezentuje pořadí elementů v jakém jsou zpracovávány. Toto pořadí se využije při generování v pozdější fázi. Instance uzlu stromu je vložena do datové struktury TREE. Tato datová struktura slouží k tomu, aby uchovala všechny vlastnosti daného elementu, které jsou potřebné při dalším zpracování. Před samotným vykreslováním výstupu vytvoří *PDFRenderer* tabulky *avoidTable* a *breakTable* a musí se vyřešit problém se stránkováním, protože existují vlastnosti, které vynucují nebo zakazují zalomení stránky na příslušném místě. K tomuto účelu slouží právě tabulky *avoidTable* a *breakTable*. Na základě těchto tabulek jsou vykreslované elementy posunuty tak, aby bylo zajištěno správné stránkování. Následně probíhá vykreslování všech elementů, které jsou uloženy ve struktuře LIST s vlastnostmi uložené ve struktuře TREE. Veškeré vykreslování elementů do PDF podoby pomocí knihovny *PDFBox* má na starosti třída `writeAllElementsToPDF()`. Před vykreslením elementů je potřeba provést ještě převod souřadnic. To je z důvodu, že *CSSBox* používá souřadnicový systém *device*, ale PDF používá souřadnicový systém *user space*. Sou-

řadnicové systémy a jejich rozdíly již byly zmíněny v kapitole 2.2. Projekt PDFRenderer také podporuje řadu CSS3 vlastností, jako například *border-radius*, gradienty, transformace, stínování a průhlednost.

Kapitola 5

Kaskádové styly a standard CSS3

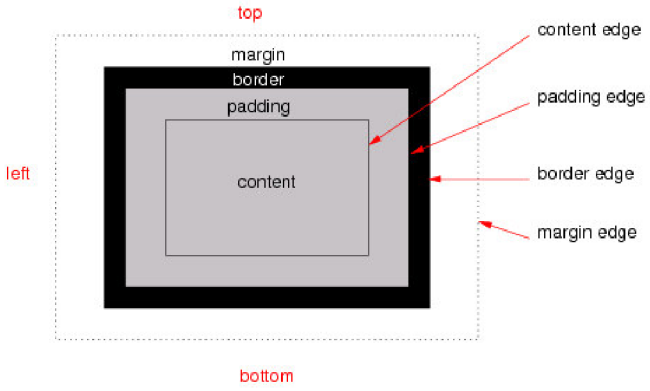
CSS je jazyk pro popis prezentace webových stránek. [12] [13] Určuje vizuální vzhled jednotlivých elementů včetně barvy, rozložení a dalších vlastností. Umožňuje také přizpůsobit prezentaci různým typům zařízení. CSS je nezávislý na HTML a lze jej použít s jakýmkoliv značkovacím jazykem založeným na XML. Oddělení CSS od HTML zlepšuje přizpůsobení zobrazení různým prostředím, udržování webů a sdílení šablon stylů mezi stránkami, což umožňuje mimo jiné oddělení obsahu od jeho prezentace. CSS je možné vložit přímo do HTML dokumentu jako atribut elementu do hlavičky mezi značky *<style>* *</style>* nebo do externího souboru CSS. Pokud jsou CSS vlastnosti uvedeny v externím souboru, je nutné uvést v hlavičce HTML dokumentu, který se má stylovat, odkaz pomocí značky *<link>*. Každý styl CSS má dvě základní části: selektor a blok deklarace. Deklarační blok nese vlastnosti formátování - barva textu, velikost písma, pozice atd. Selektor určuje, co se bude formátovat. Může vybrat obecně nějakou množinu položek, či může vybrat jen jednu konkrétní položku nebo kolekci podobných položek. Aktuální verze CSS je CSS3. Ta obsahuje oproti předchozím verzím více vlastností. Mezi ně patří například transformace, možnost využití gradientů pro pozadí a zaoblené rohy rámečků. Syntaxe CSS jazyka zde popsaná nebude, protože tato práce se tímto tématem nezabývá. Pro účely této práce budou podstatné hlavně některé vlastnosti CSS, které jsou v této práci implementovány ve formě PDF a SVG. Vybrané vlastnosti budou popsány níže.

5.1 CSS Box Model

Každé pole má oblast obsahu a volitelné obklopující oblasti **padding, border** a **margin**. [9] Velikost každé oblasti je určena odpovídajícími vlastnostmi. Velikost těchto oblastí může být nulová, nebo v případě okrajů i záporná. Na obrázku 5.1 je graficky znázorněno, jak spolu tyto oblasti souvisejí.

Pro všechny obklopující oblasti lze ovládat vlastnosti každé strany nezávisle. Obvod každé ze čtyř oblastí se nazývá hrana. Obklopujícími oblastmi jsou:

- **content edge/inner edge (okraj obsahu)** - okraj obsahu obklopuje obdélník daný šířkou a výškou pole, které často závisí na obsahu prvku, případně na velikosti bloku. Všechny čtyři strany společně definují pole obsahu.
- **padding edge** - pokud má padding na dané straně nulovou šířku, okraj paddingu se shoduje s okrajem obsahu pro danou stranu. Tyto čtyři stěny definují padding boxu, který obsahuje oblasti obsahu i paddingu.



Obrázek 5.1: Model CSS boxu. [10]

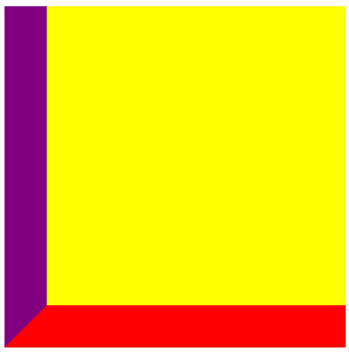
- **border edge (hrana okraje)** - obklopuje hranici boxu. Pokud má ohraničení na dané straně nulovou šířku, tak se tento okraj shoduje s hranou paddingu na této straně. Čtyři strany společně definují rámeček, který ohraničuje obsah rámečku, padding a hranice okraje.
- **margin edge/outer edge (vnější okraj)** - pokud má tento okraj nulovou šířku, shoduje se s okrajem border. Čtyři strany tohoto okraje společně definují okrajový rámeček pole, který ohraničuje obsah, padding, hranu okraje a vnější okraj.

Pozadí obsahu, paddingu a rámečku je určeno jeho vlastnostmi pozadí. Okrajová oblast ohraničení může být obarvena vlastností *border*. Oblast *margin* je vždy průhledná.

5.2 Rámečky a zaoblené rohy

V CSS lze pro každou stranu zvlášť definovat barvu, styl a šířku. [10] V CCS3 lze navíc ještě definovat zaoblení rohů rámečku.

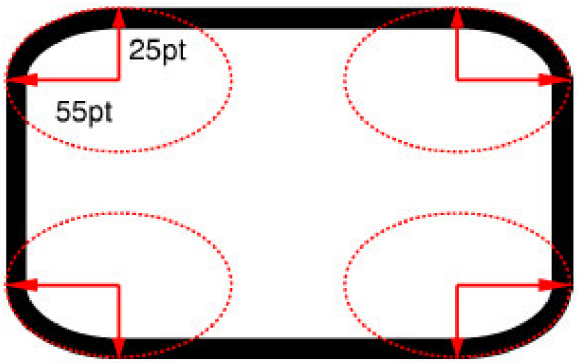
Pokud není nastaveno žádné zaoblení rámečku, je roh mezi stranami vykreslen pomocí trojúhelníku, jak je možné vidět na obrázku 5.2.



Obrázek 5.2: Vykreslení rohu bez nastaveného rádiusu (příklad je z prohlížeče Chrome).

Zaoblení rohu se nastavuje pomocí vlastnosti *border-radius*. Tuto vlastnost lze nastavit každému jednotlivému rohu. Každé zaoblení rohu je definováno elipsou. Hodnoty vlastnosti *border-radius* definují poloměry čtvrtinové elipsy, která definuje tvar rohu vnější hra-

nice. První hodnota je horizontální poloměr, druhá je pak vertikální poloměr. Podle těchto hodnot jsou nastaveny velikosti poloos elipsy. Pokud je druhá hodnota vynechaná, bere se pro velikost druhé poloosy první hodnota. Pokud je délka zaoblení nulová, je roh ostrý, nikoliv zaoblený. Dá se tedy definovat rozdílné zaoblení pro roh ve směru x i ve směru y. Vnitřní elipsa je tvořena stejnou elipsou, pouze velikosti jejich poloos jsou sníženy o šířku rámečku příslušné strany. Vykreslování zaoblených rohů rámečku je graficky znázorněno na obrázku 5.3.



Obrázek 5.3: Ukázka vykreslení vlastnosti *border-raius*. [10]

V kódu CSS vypadá kód pro toto zaoblení následovně:

```
border: 8pt solid black;
border-radius: 55pt / 25pt;
```

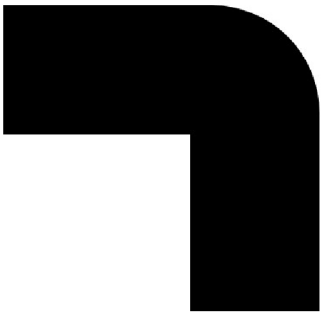
V tomto případě bude okraj široký 8pt, černý a nepřerušovaný. Pokud jsou hodnoty uvedeny před a za lomítkem, pak hodnoty před lomítkem nastavují horizontální poloměr a hodnoty za lomítkem nastavují vertikální poloměr. Pokud není lomítko, pak hodnoty nastaví oba poloměry stejně. Délka poloosy elipsy ve směru x bude nastavena na hodnotu 55pt a délka elipsy ve směru y bude nastavena na hodnotu 25pt. Vnitřní elipsa pak bude mít velikost poloos 47pt a 17pt.

Vlastnost *border-radius* platí pro všechny rohy, ale dá se definovat i pro každý roh samostatně pomocí vlastností *border-top-left-radius*, *border-top-rigt-radius*, *border-bottom-rigt-radius* a *border-bottom-left-radius*. Nebo lze zadat více poloměrů do vlastnosti border radius. Například `border-radius: 25px, 50px`. To nastaví zaoblení pro levý horní roh 25px a pro pravý horní roh 50px. Takto lze zadat všechny čtyři rohy. Pokud hodnota pro některý roh chybí, je převzata z předchozí. V tomto případě by měl pravý spodní roh zaoblení 25px a levý spodní roh 50px.

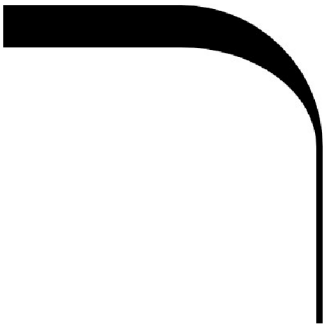
Poloměr vnitřní hranice rohu se počítá jako poloměr vnějšího okraje mínus šířka okraje. V případě, že vyjde jako výsledek záporná hodnota, je poloměr vnitřního okraje 0. Poté bude vnitřní okraj ostrý. Tento případ nastane, pokud bude šířka hrany větší než poloměr v daném rohu. Ukázku vykreslení pro tento případ můžeme vidět na obrázku 5.4.

Pokud mají dva sousední okraje různou šířku, roh bude vykreslen plynule a bude se postupně ztenčovat od širšího okraje ke slabšímu. Tento případ je znázorněn na obrázku 5.5.

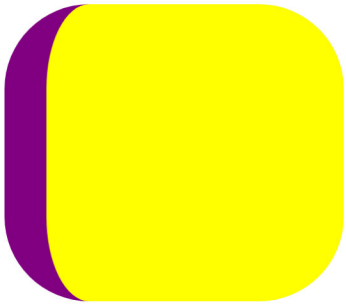
Speciální případ této vlastnosti můžeme vidět na obrázku 5.6. Ten nastává ve chvíli, kdy je tloušťka jednoho rohu nulová. V takovém případě se rámeček postupně ztenčuje až do bodu, kde končí zaoblení rámečku pro následující hranu.



Obrázek 5.4: Vykreslení rohu pro případ, kdy je šířka hrany větší než hodnota zaoblení pro daný roh (příklad je z prohlížeče Chrome).



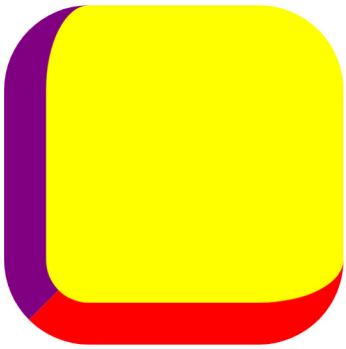
Obrázek 5.5: Vykreslení rohu pro případ, kdy je šířka dvou sousedních hran různá (příklad je z prohlížeče Chrome).



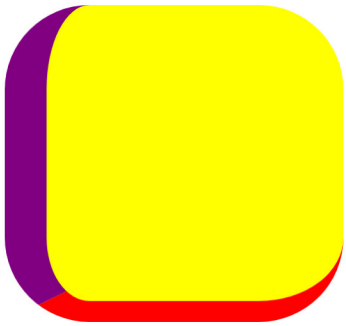
Obrázek 5.6: Vykreslení rohu pro případ, kdy je šířka jedné hrany nulová (příklad je z prohlížeče Chrome).

V případě, že mají dvě sousední strany stejnou šířku, ale rozdílnou barvu, musí dojít v zaoblení rohu k přechodu barvy. V tomto případě, kdy jsou šířky sousedních hran shodné, k tomu dojde na místě, kde diagonála vykreslovaného elementu protíná spoj obou rámečků stran viz. obrázek 5.7.

V případě, že sousední hrany mají rozdílnou barvu a k tomu mají ještě rozdílnou šířku, je třeba vykreslování rohu realizovat tak, že se hrany postupně ztenčují nebo rozšiřují, aby se uprostřed rohu setkaly ve stejné šířce. Přechod barev se pak nenachází na diagonále, ale na přímce, jejíž směrnice je určena podle poměru šířek jednotlivých stran.



Obrázek 5.7: Vykreslení rohu pro případ, kdy je šířka dvou sousedních hran stejná, ale mají rozdílnou barvu (příklad je z prohlížeče Chrome).



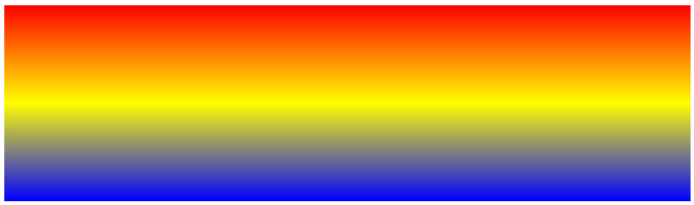
Obrázek 5.8: Vykreslení rohu pro případ, kdy je šířka dvou sousedních hran různá a navíc mají rozdílnou barvu (příklad je z prohlížeče Chrome).

5.3 Gradient

CSS definuje dva základní typy gradientů, a to konkrétně lineární a radiální gradient. [13] Oba dva se ještě mohou dělit na opakující se a neopakující se. Nejčastěji se gradienty v CSS využívají na pozadí, i když to není jejich jediné využití. Jedná se o plynulý vizuální přechod z jedné barvy na druhou. Například přechod z bílé na černou jde od bílé přes stále tmavší odstíny šedé, až nakonec dospěje k černé barvě. Pokud se nedefinuje velikost gradientu, je jako výchozí nastavena hodnota auto. To znamená, že gradient pokryje celou plochu pozadí. Gradient je v podstatě bitový obrázek stejně jako PNG, SVG apod.

Lineární gradient

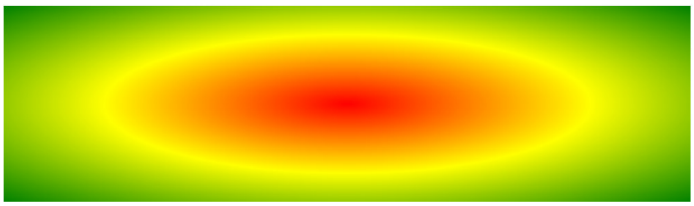
Lineární gradient je přechod probíhající podél lineárního vektoru, který se nazývá *gradient line*. [13] Pozice počáteční a koncové barvy je dána pomocí dvou bodů *starting point* a *ending point*. Pokud jsou tyto body umístěny mimo plochu, kam má být gradient vložen, je vykreslen přechod pouze po barvu, která se ještě vejde do vybrané oblasti. Volitelně mohou být zadané hodnoty *positioning color stops*, díky nimž můžeme zadat pozice, které dávají schopnost narušit rovnoměrný přechod mezi barvami. Směr gradientu lze nastavit doleva, doprava, nahoru, dolů, diagonálně nebo pomocí úhlu. Příklad lineárního gradientu lze vidět na obrázku 5.9.



Obrázek 5.9: Ukázka lineárního gradientu (příklad je z prohlížeče Chrome).

Radiální gradient

Radiální gradient reprezentuje přechod mezi dvěma nebo více barvami začínající v nějakém bodě a postupně se z tohoto bodu rozšiřuje směrem ven. [13] Může mít nastaven tvar kruhu či elipsy. Tento tvar je reprezentovaný pomocí *ending shape*. Bodu, odkud gradient začíná, se říká *center point*. Pokud není zadána jeho pozice, je tento bod ve středu daného elementu, kde má být gradient vykreslen. Pokud gradient nemá určený parametr nastavující jeho tvar, je vykreslen jako kruh v případě, že se jedná o element tvaru čtverce. V ostatních případech má tvar elipsy. Tvar gradientu však nesouvisí s velikostí. Velikost lze nastavit procentuálně, ale pouze pro tvar elipsy. Gradienty lze zmenšit také následujícími klíčovými slovy: *closest-side*, *farthest-side*, *closest-corner* nebo *farthes corner*. Klíčová slova nelze kombinovat s velikostí nebo procenty. *Color stops* fungují u radiálního gradientu stejným způsobem jako u lineárního gradientu. Ukázku radiálního gradientu je možné vidět na obrázku 5.10.



Obrázek 5.10: Ukázka radiálního gradientu (příklad je z prohlížeče Chrome).

5.4 2D transformace

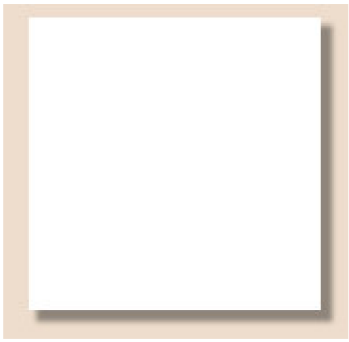
Pomocí transformací CSS je možné objekty otočit, posouvat, měnit jim rozměry apod. [13] Při těchto operacích dochází k modifikaci souřadného systému objektů. Existují dvoudimenzionální (2D) a trojdimenzionální (3D) transformace. 2D transformace pracují s horizontální osou souřadného systému (x) a vertikální osou souřadného systému (y). 3D transformace ještě navíc přidávají osu (z) pro hloubku. CSS3 umožňuje aplikovat na HTML elementy následující efekty:

- **translate** - posune element po jedné nebo více osách z původního místa. Pomocí `translateX()` a `translateY()` lze element posunout pouze po jedné ose. Pokud je vyžadováno posunutí elementu po ose x i po ose y, použije se `translate()`. V tomto případě se zadávají dvě hodnoty, kde první představuje posunutí po ose x a druhá posunutí po ose y. Kladné hodnoty posouvají element doprava nebo dolů a záporné hodnoty jej posouvají doleva nebo nahoru. Posunutí lze vyjádřit např. v pixelech nebo v procentech, kdy se procenta uvádí vzhledem k rozměrům daného elementu.

- **scale** - jedná se o transformaci měřítka, kdy se element zvětšuje či zmenšuje. Hodnoty, podle kterých se daný element mění, jsou bezjednotková reálná čísla a jsou vždy kladná. Je možné měnit odděleně šířku nebo výšku prvku s použitím `scaleX()` a `scaleY()`, nebo můžeme měnit měřítko obou os najednou pomocí `scale()`. Zde je první parametr pro osu x a druhý pro osu y. Pokud se však zadá pouze jeden parametr, použije se na měřítka obou dvou os.
- **rotation** - funkce rotace způsobí, že se prvek bude otáčet kolem své osy podle hodnoty parametru *angle*. Pokud je hodnota kladná, bude se prvek otáčet ve směru hodinových ručiček. I zde existují varianty `rotateX()` a `rotateY()` pro jednotlivé osy a varianta `rotate()` pro rotaci kolem obou os zároveň.
- **skew** - tato funkce provádí zešikmení elementů. Element lze naklonit podél jedné nebo obou os x a y. V obou případech se zadá úhel, o kolik je daný prvek zkosený.
- **matrix** - pomocí této funkce lze realizovat všechny výše zmíněné operace, ale navíc i jejich kombinace.

5.5 Stínování

Stínování se v CSS nastavuje pomocí *box-shadow*. Tato funkce má čtyři parametry. První hodnota nastavuje horizontální offset a druhá hodnota nastavuje vertikální offset. Kladná čísla pohybují stínem dolů a doprava a záporná pohybují stínem nahoru a doleva. Je-li zadán třetí parametr, definuje vzdálenost rozostření, která určuje, kolik prostoru stínu bude rozmazáno. Čtvrtý parametr definuje roztažnou vlastnost stínu, která mění celkovou velikost stínu. Kladné hodnoty způsobí roztažení stínu ještě před rozmazáním a záporné mají za následek zmenšení stínu. Ukázku stínování lze vidět na obrázku 5.11.



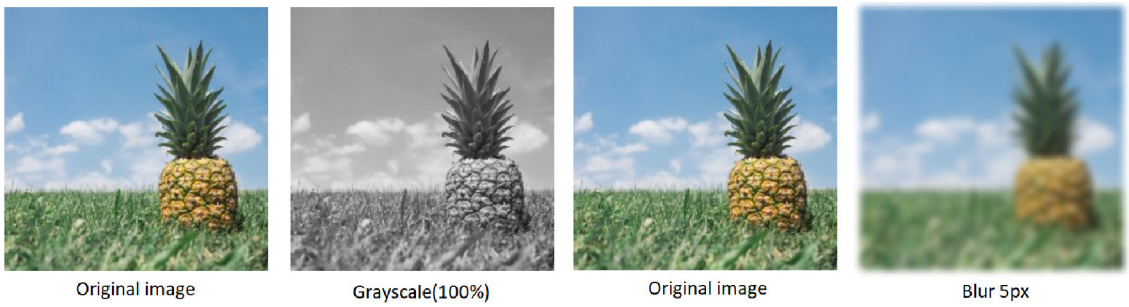
Obrázek 5.11: Ukázka CSS stínování (příklad je z prohlížeče Chrome).

5.6 Filter

CSS3 *filter* přináší grafické efekty, jako například rozmazání, průhlednost, invertování barev a další. Hodnoty této vlastnosti jsou definované funkcemi typu filtr. Vybrané základní filtry budou popsány níže:

- **blur(<length>)** - rozostří obsah prvku pomocí Gaussova rozostření. Standardní odchylka je dána hodnotou *length*.

- **opacity([<number> | <percentage>])** - aplikuje filtr průhlednosti. Hodnota 1 nebo 100% nechá prvek beze změny.
- **drop-shadow(<length>2,3 <color>?)** - vytvoří vržený stín, který odpovídá tvaru alfa kanálu prvku s rozmazáním a použitím libovolné barvy. Pokud není barva zadána, použije se barva daného elementu.



Obrázek 5.12: Základní filtr funkce pro CSS.[1]

5.7 CSS Page-break

Stránkovaná média se od kontinuálních médií liší tím, že je obsah dokumentu rozdělen na jednu nebo více samostatných stran. [17] Tento odstavec popisuje pět CSS vlastností, na základě kterých se při vykreslování určí, kde se stránka může nebo má ukončit a kde se naopak stránkování vyhnout, pokud je to možné.

- **page-break-before:**
  - **always** - před prvkem se stránka při tisku zalomí vždy
  - **avoid** - před prvkem se stránka při tisku nesmí zalomit (pokud je to možné)
- **page-break-after:**
  - **always** - za prvkem se stránka při tisku zalomí vždy
  - **avoid** - za prvkem se stránka při tisku nesmí zalomit (pokud je to možné)
- **page-break-inside:**
  - **avoid**- uvnitř prvku nesmí dojít k zalomení stránky při tisku (pokud je to možné)

[1] Pro demonstraci vlastnosti filter byl použit obrázek ze stránky https://www.w3schools.com/howto/howto_css_image_effects.asp

Kapitola 6

Návrh nové společné struktury pro vykreslování SVG a PDF vektorové grafiky

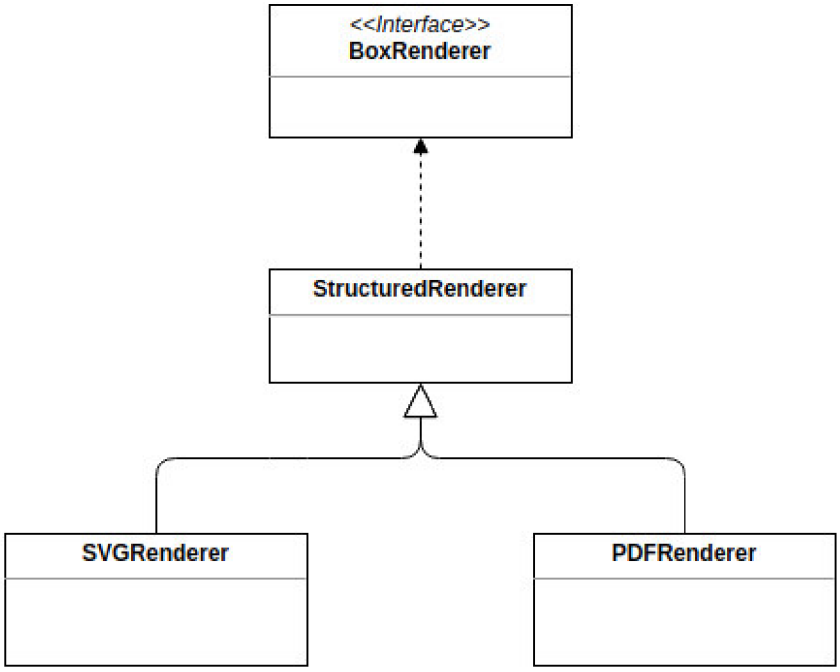
V této kapitole bude popsán návrh nové struktury, jejímž cílem bude co nejvíce spojit oba přístupy pro renderování vektorové grafiky PDF a SVG. Rovněž zde budou vysvětleny důvody, proč je aktuální struktura nevyhovující, a které části jednotlivých prací byly implementovány chybně.

Již existující bakalářské a diplomové práce, které řeší problém vykreslování HTML výstupu z renderovacího nástroje CSSBox, mají rozdílný přístup k tomuto problému. Prozkoumáním těchto řešení se však ukázalo, že velká část implementace by mohla být společná. Největší problém původních prací je v jejich nevyhovující struktuře. U obou projektů je většina metod pro vykreslování implementována v rámci jednoho mnohařádkového souboru. Tato struktura má za následek, že existující implementace je poměrně nepřehledná a nelze použít jednotlivé části společně pro oba dva přístupy. Další nevýhodou je, že oba dva přístupy mají odlišný princip zpracování celého vykreslování. Ve společném řešení pro generování SVG a PDF by se tak objevovalo mnoho duplicitních metod řešících stejný problém.

6.1 Nová struktura

Obě třídy `SVGRenderer` i `PDFRenderer` implementují rozhraní `BoxRenderer`. Základem nové struktury je vytvoření nové třídy `StructuredRenderer`, která bude rovněž implementovat rozhraní `BoxRenderer`. Tato třída bude spojovat společné postupy a metody pro vykreslování PDF i SVG. Tuto třídu budou rozšiřovat dvě specifické třídy `SVGRenderer` a `PDFRenderer`. Obě třídy budou obsahovat konkrétní části pro vykreslování PDF či SVG, které již nešly sjednotit do společného postupu do třídy `Structured Renderer`. Návrh této části lze vidět na obrázku 6.1.

Základním předpokladem pro realizaci této struktury je, aby se v metodách pro zpracování elementu, které třída `StructuredRenderer` implementuje, prováděly podobné postupy. Konkrétně se jedná o metody `startElementContents()`, `finishElementContents()`, `renderElementBackground()`, `renderReplacedContent()`, `renderTextContent()` a `renderMarker()`. V předešlé verzi tomu tak není. Každé řešení má zcela odlišný princip zpracování a vykreslování elementů, a proto nelze tyto dva přístupy spojit. Verze SVG v těchto metodách implementuje přímo zpracování a vykreslování pro dané elementy, jak je

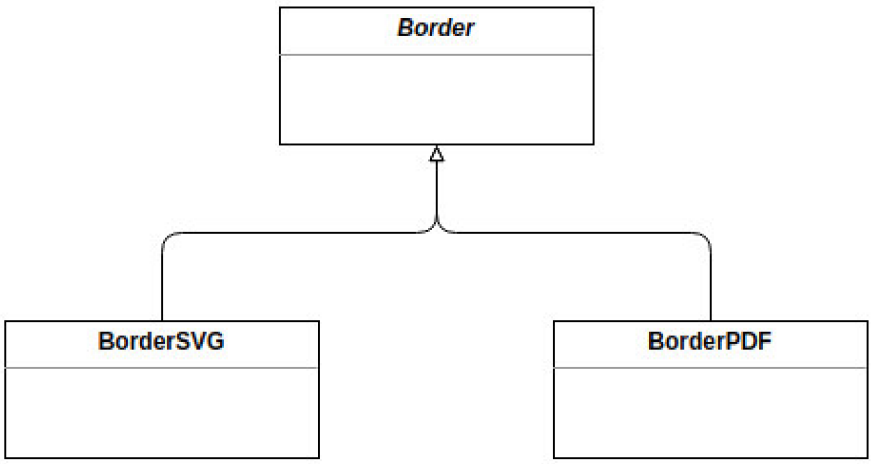


Obrázek 6.1: Návrh společné struktury.

předpokládáno. Verze PDF však nikoliv. Ta z důvodu problému se stránkováním v těchto metodách pouze ukládá dané elementy, které jsou až po uložení všech elementů těmito metodami zpracovány a vykreslovány. Bude tedy nutné přepracovat proces stránkování PDF. Je žádoucí, aby se stránkování vyřešilo již při zpracování prvního elementu. Jedná se konkrétně o element, který je instancí třídy *Viewport*. Zde se projde celý DOM strom elementů a na jeho základě se vytvoří tabulky BreakTable a AvoidTable pro podporu stránkování a spočítá se stránkování vzhledem k vlastnostem zpracovávaných elementů. Ve chvíli, kdy budou zpracovávány ostatní elementy, již bude stránkování dáno, a proto nebude nic bránit tomu, spojit tyto dva principy do jednoho. V původním přístupu PDF nebylo intuitivní, jak se jednotlivé elementy zpracovávají, protože názvy tříd, které byly implementovány z rozhraní `BoxRenderer`, neodpovídaly tomu, co v nich bylo implementováno. V tomto řešení bylo nutné vytvořit znovu postup zpracovávání elementu, obdobně jak tomu je při volání metod třídy `BoxRenderer`. Při analýze existujících řešení se ukázalo, že velká část implementace by mohla být pro tyto projekty společná. Řešení pro SVG má obecně lepší strukturu zpracování jednotlivých elementů a je mnohem přehlednější, intuitivnější i lépe strukturované. Proto bylo rozhodnuto, že se bude vycházet z řešení SVG, které bude modifikované pro potřeby spojení obou projektů a verze PDF bude přepracována do principu zpracování projektu pro vykreslování SVG.

Oba dva projekty obsahují třídy, které reprezentují například rámeček, zaoblené rohy, gradienty nebo transformace. Tyto třídy a postupy získávání hodnot v těchto třídách se od sebe v jednotlivých řešeních výrazně liší. Tyto třídy implementují to samé a není důvod, aby se například reprezentace zaobleného rohu a jeho parametry lišily pro verzi SVG i PDF. Jediná odlišnost vznikne až při vykreslování samotného objektu ve formě PDF nebo SVG. Přesto vnitřní reprezentace těchto objektů pro zpracování může být společná. Proto byla

navržena nová struktura pro reprezentaci a zpracování těchto objektů. Princip bude ukázán na třídě pro rámeček. Novou strukturu pro tento prvek lze vidět na obrázku 6.2.



Obrázek 6.2: Návrh společné struktury pro reprezentaci a vykreslování rámečků.

Nová struktura zahrnuje vytvoření abstraktní třídy `Border`. Ta obsahuje obecnou vnitřní reprezentaci rámečku elementu. Jsou zde reprezentovány všechny body rámečku a každý roh rámečku. Tato třída obsahuje také metody společné pro PDF i SVG. Těmito metodami jsou metody pro výpočet ostatních bodů potřebných při vykreslování rámečku. Vypočítané body jsou použity například, pokud hranice obsahuje zaoblené rohy. Body začátků a konců pro vykreslení jednotlivých rovných hran jsou tak posunuty. Tuto abstraktní třídu `Border` rozšiřují třídy `BorderSVG` a `BorderPDF`. Tyto třídy obsahují přepočet mezi souřadnicemi systému *user space* a *device space*. Dále obsahují již konkrétní způsob vykreslování rámečku a také konkrétní způsob vykreslení zaoblených rohů, protože ten se vzhledem k různému využití technologií v obou případech liší.

Celý diagram znázorňující novou strukturu projektu pro generování vektorové grafiky v SVG i v PDF se nachází v příloze D.

6.2 Problémy aktuálního řešení

Obě varianty nejsou v původním stavu, ze kterého se vychází, zcela funkční. Kromě nevhodné struktury obsahují rovněž řadu chybně implementovaných nebo vůbec neimplementovaných funkcí.

6.2.1 Problémy knihovny pro vykreslování výstupu SVG

Verze pro vykreslování SVG je obecně propracovanější. Přesto se v její implementaci objevuje několik problémů.

Ukázalo se, že při použití vlastnosti *border-radius* pro rámeček daného elementu nedojde ve všech případech k ořezání pozadí. To má za následek, že rámeček je sice v rozích zaoblen, pozadí daného elementu ale hranice rámečku v rozích přesahuje a má stále tvar klasického obdélníku. Další dva problémy byly rovněž nalezeny při vykreslování rámečku elementu. Pokud není rámečku nastavena barva, měla by být použita výchozí barva, a to

konkrétně barva černá. V tomto případě obsahuje implementace chybu, a pokud není nastavena barva rámečku, nevykreslí rámeček elementu vůbec. Další problém nastává ve chvíli, kdy je nastavená velikost rádiusu daného rohu shodná se šířkou rámečku. V tomto případě dochází v implementaci k neošetřenému dělení nulou v metodě `ellipseLineIntersect()` a výsledný rámeček je vykreslen chybně, jak je ukázáno na obrázku 6.3. Rámeček v takovém případě místo k dalšímu bodu pro vykreslení rádiusu zamíří k bodu se souřadnicemi (0, 0).

Obrázek 6.3: Problém při vykreslování rámečku ve verzi SVG.

6.2.2 Problémy knihovny pro vykreslování výstupu PDF

Ve verzi PDF je problémů mnohem více. Bude zde uvedeno pouze několik z nich.

Při zkoumání řešení problému stránkování se ukázalo, že je chybně počítán offset při nastavení hodnoty avoid pro atribut elementu *page-break-inside*. Hodnota offset se počítá pouze z výšky daného elementu. Musí zde být ale započítána vzdálenost až ke konci dané stránky. Prvek, který má nastavenou vlastnost avoid inside, by se měl v případě, že konec stránky bude dělit tento element, posunout na novou stránku. Tato chyba má za následek, že místo toho, aby se celý prvek začal vykreslovat na následující stránce, se část prvku začne vykreslovat ještě na aktuální stránce a zbytek na následující stránce. Prvek je tedy rozdělen koncem stránky i přesto, že má nastavenou vlastnost, která toto rozdělení uvnitř prvku zakazuje.

Další chyba v implementaci se týká rovněž stránkování. V případě, že se nachází nějaký záznam v tabulce avoid table, jsou následně ignorovány záznamy v tabulce break table, které slouží k tomu, aby před nebo za nějakým elementem došlo k nucenému ukončení aktuální stránky.

Další z hlavních problémů byl nalezen při vykreslování rámečku. Konkrétně při vykreslování zaoblených rohů. SVG verze používá pro vykreslování zaoblených rohů eliptické křivky, pomocí kterých je vykreslován zaoblený roh i v CSS3. To umožňuje nastavit jiný rádius pro hodnotu x i hodnotu y. Rádius rohu tak není na všech místech konstantní. SVG také podporuje různé šířky sousedních hran a přechody barev uvnitř zaobleného rohu. Tento postup je správný. Verze PDF však specifikaci CSS3 nedodržuje. Umí vykreslit pouze jednobarevný zaoblený roh. To je z důvodu, že se v tomto případě vykresluje zaoblený roh jako celek. Kvůli tomu PDF nepodporuje ani přechod barev uvnitř zaobleného rohu. Taktéž PDF nepodporuje různý rádius pro hodnotu x a hodnotu y. V PDF neexistují eliptické křivky, proto bude muset být navržen způsob, jak pomocí Beziérových křivek vykreslovat zaoblené rohy v PDF tak, aby splnila všechna specifika CSS3.

Kapitola 7

Implementace

V této kapitole bude popsán způsob realizace společné struktury pro jednotlivé prvky. Také zde bude popsáno jakým způsobem probíhala implementace společné struktury a oprava všech chyb, které bylo nutné provést u částí, jež byly využity z předešlých prací.

7.1 Návrh a implementace generování pozadí

Jednou z metod, která se při renderování stránky pomocí *CSSBoxu* volá jako první, je metoda `renderElementBackground()`. Tato metoda má za úkol generování pozadí a rámečků pro elementy. V rámci nové struktury projektu byl vytvořen nový postup vykreslování pozadí jak u SVG, tak u PDF. Cílem nové struktury bylo co nejvíce sjednotit princip, jakým probíhá generování pozadí pro oba vektorové formáty.

Metoda `renderElementBackground()`, nacházející se ve třídě `StructuredRenderer`, spojuje společné postupy pro oba formáty při vykreslování pozadí. Zde nejdříve proběhne získání souřadnic elementu, pro nějž se bude pozadí vykreslovat. Dále se pomocí metody `findBackgroundSource()`, která je rovněž implementována ve třídě `StructuredRenderer`, získá zdroj pro pozadí elementu. Na základě tohoto zdroje je poté rozhodnuto, zda bude pozadí vykresleno. Pokud ano, může být pozadí realizováno jako pozadí jednobarevné, pozadí realizováno obrázkem nebo pozadí obsahující gradient. V těchto případech jsou volány konkrétní metody, které realizují vykreslení příslušného typu pozadí. Tyto metody a jejich principy budou podrobněji popsány níže.

V rámci metody `renderElementBackground()` jsou následně ve třídě `StructuredRenderer` vykresleny rámečky obrázku. O to se stará metoda `renderBorder()`, implementovaná ve třídách `SVGRenderer` a `PDFRenderer`. Princip vykreslování rámečků a zaoblených rohů v obou formátech bude podrobněji vysvětlen v kapitole 7.2.

Po dokončení společného postupu pro oba formáty ve třídě `StructuredRenderer` pokračuje implementace této metody ve třídách `SVGRenderer` a `PDFRenderer`, které ze třídy `StructuredRenderer` dědí.

V případě SVG, pokud je použito pozadí, dochází ke kontrole, zda je na prvek použita nějaká transformace nebo vlastnost *filter*. V takovém případě je na prvek aplikována příslušná úprava. Také je zde nutné přidat ořezovou cestu pro případ, kdy element obsahuje zároveň zaoblené rohy. Ořezová cesta je přidána z toho důvodu, aby pozadí nepřesahovalo hranice elementu. Nakonec dojde k přidání pozadí jako potomka předchozímu elementu. Metody pro použití transformací a filtrů budou popsány níže v kapitolách 7.9 a 7.10.

V případě PDF je nutné před provedením metody *renderElementBackground()* ve třídě *StructuredRenderer* vyřešit stránkování, které bude popsáno podrobně v kapitole 7.7. Poté je volána metoda z abstraktní třídy pro provedení společné části. Po dokončení společné části se v této metodě pro PDF řeší případné transformace elementu. Ty budou rovněž popsány stejně jako v případě SVG v kapitole 7.9.

7.1.1 Generování jednobarevného pozadí

Generování jednobarevného pozadí probíhá v rámci metody *renderColorBg()*, která je volána v rámci metody *renderElementBackground()* z třídy *StructuredRenderer*. Tento postup nebylo možné více spojit a je odlišný pro verzi SVG i PDF. Implementace této metody se tak nachází ve třídách *SVGRenderer* a *PDFRenderer*.

Ve verzi SVG dojde pouze k vykreslení obdélníku o dané velikosti s určenou barvou. Případné ořezání se řeší již v metodě *renderElementBackground()* pomocí přidání ořezové cesty, která se přidá jako potomek elementu reprezentující pozadí.

Ve verzi PDF je situace o něco komplikovanější. Nejdříve je nutné převést souřadnice *device-space* souřadnicového systému na souřadnicový systém *user-space*. Tento postup musí být proveden u všech elementů, které budou v PDF vykreslovány. Převod probíhá podle následujícího vzorečku:

```
y = (pdf.getPageHeight() - (elem.getAbsoluteContentY()) + i *
    pdf.getPageHeight() - elem.getContentHeight() - plusHeight -
    plusOffset - elem.getPadding().bottom - elem.getBorder().bottom);
```

Samotné generování barevného pozadí se liší v závislosti na tom, zda obsahuje element zakulacené rohy a je tedy nutné pozadí oříznout. K tomuto účelu slouží metoda *bordersRadiusUsed*, která na základě vlastností elementu vrátí hodnotu true, pokud má nějaký roh elementu nastavenou vlastnost `border-radius`.

V případě, že zaoblené rohy element neobsahuje, je vykreslen pomocí knihovny PDFBox pouze obdélník na dané pozici o určité velikosti, kterému je nastavena požadovaná barva. V případě zaoblených rohů je nutné vytvořit již přímo element, jehož vnější hranice bude kopírovat rámeček elementu včetně zaoblených rohů. Je zavolána metoda *insertClippedBackground()*, která získá ořezovou cestu elementu s ohledem na zaoblené rohy. Pro vznik této cesty jsou použity algoritmy, které budou popsány níže v kapitole 7.2 popisující vykreslování zaoblených rohů. Tato cesta reprezentuje pozadí elementu. Dále stačí pouze nastavit této cestě barvu a aplikovat výplň na daný element.

7.1.2 Generování pozadí pomocí obrázku

Pro generování pozadí je nejprve potřeba získat obrázek, který má být pro pozadí použit. To probíhá v rámci metody *renderElementBackground* ve třídě *StructuredRenderer*. Následně je s tímto obrázkem volána metoda *renderImageBg()*, která se stará o vykreslení obrázku v příslušném vektorovém jazyce.

V SVG je pomocí metody *renderImageBg()* vytvořen obrázek pomocí knihovny Apache Batik, který je následně nastaven jako potomek elementu reprezentujícího pozadí pro vykreslovaný prvek.

V PDF jsou nejdříve převedeny souřadnice obrázku do souřadnicového systému *user-space*. Následně se řeší generování obrázku dvěma způsoby stejně jako v případě jednobarevného pozadí v PDF. Pokud element nemá zaoblené rohy, je pomocí knihovny PDFBox pouze vložen obrázek na určitou pozici do výstupního dokumentu. V opačném případě je

nutné řešit ořezovou cestu. To má za úkol *insertClippedImage*. Ta vytvoří pomocí metody *createClippedPath()* ořezovou cestu stejně jak tomu bylo v předešlém případě. Při renderování obrázku pomocí knihovny PDFBox se pak pomocí příkazu `appendRawCommands("W ")` definuje, že se má obrázek oříznout podle ořezové cesty.

Posledním typem generování pozadí je pozadí ve formě gradientu. Tato problematika je poněkud komplikovanější. Proto bude tvorbě a vykreslení gradientů věnována kapitola 7.8.

7.2 Návrh a implementace generování rámečků a zaoblených rohů

V obou původních řešeních se reprezentace rámečků a zaoblených rohů výrazně lišila. Proto byly vytvořeny společné objekty pro reprezentaci rámečku a zaoblených rohů, jak již bylo popsáno v kapitole 5.2. Taktéž byly vytvořeny společné postupy pro všechny výpočty nutné ke generování zaobleného rohu. Samotný způsob konečného vykreslení zaobleného rohu se pak v obou vektorových jazycích liší.

7.2.1 Reprezentace rámečku

Celý rámeček daného elementu reprezentuje abstraktní třída *Border*. Tato třída obsahuje původní reprezentaci rámečku pomocí třídy *Rectangle*. Dále obsahuje všechny krajní body pro vykreslení rovných čar rámečku vzhledem k míře zaoblení rohů. Tyto body společně s postupem výpočtu budou vysvětleny níže. Třída *Rectangle* rovněž obsahuje barvu pro každou stranu rámečku a pro každý roh jednu instanci třídy *CornerRadius* reprezentující body potřebné k vykreslení zaoblených rohů.

Standardní možnosti vykreslení rámečku se zaoblenými rohy, které se nacházejí v SVG nebo PDF, nejsou vhodné, protože neodpovídají standardu CSS. V CSS3 je možné definovat zaoblení pro každý jednotlivý roh. To není pomocí nástrojů pro vykreslování zaoblených rámečků v SVG a PDF možné. Také musí umožňovat přechod barvy uprostřed zaobleného rohu v případě, že sousedící strany mají odlišnou barvu a možnost nastavení jiného zaoblení rohu pro osu x a y pomocí hlavní a vedlejší poloosy elipsy. Řešením je generovat každou hranu rámečku samostatně a následně vykreslit jednotlivě zaoblené rohy po částech pomocí Beziérových nebo eliptických křivek.

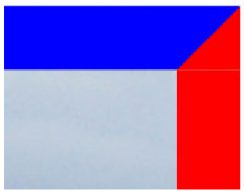
7.2.2 Vykreslení rámečku

Nejdříve se spočítají krajní body pro rovné čáry rámečku. To má na starost metoda *calculateBorderPoints()*, která se nachází v abstraktní třídě *Border*. Tato metoda vypočítá krajní body pro vykreslení přímek na základě rozměru elementu, velikosti zaoblených rohů a šířky jednotlivých stran. Pomocí obdélníku se vykreslí prvek, který reprezentuje jednu stranu rámečku a je vyplněn příslušnou barvou. Tento postup se opakuje následně pro všechny strany rámečku. Princip postupu je stejný pro verzi SVG i PDF.

7.2.3 Vykreslení nezaoblených rohů

Protože každá strana rámečku může mít jinou barvu, nelze vykreslit rohy společně se stranami rámečku. V rohu rámečku musí dojít k přechodu barvy na jeho diagonále, která míří do vnějšího rohu rámečku. Toto je realizováno v SVG i PDF obdobným způsobem. Nejdříve

se na základě krajních bodů sousedních stran vypočítají příslušné body pro vykreslení trojúhelníku, který představuje polovinu rohu. Tyto tři body jsou následně spojeny rovnými čarami a oblast, kterou vytvoří, je vyplněna příslušnou barvou. Druhá polovina rohu je vykreslena obdobným způsobem, pouze jsou zaměněny příslušné x-ové a y-nové souřadnice bodů. Příklad vykreslení nezaobleného rohu, kde dvě sousední strany mají stejnou barvu, je vidět na obrázku 7.1. Tento postup proběhne pro všechny čtyři rohy vykreslovaného elementu.



Obrázek 7.1: Ukázka generování nezaobleného rohu v PDF.

7.2.4 Vykreslení zaoblených rohů

Zaoblený roh reprezentuje třída *CornerRaius*. V CSS3 může být každý roh rámečku definován následovně:

```
border-top-right-radius: r s
```

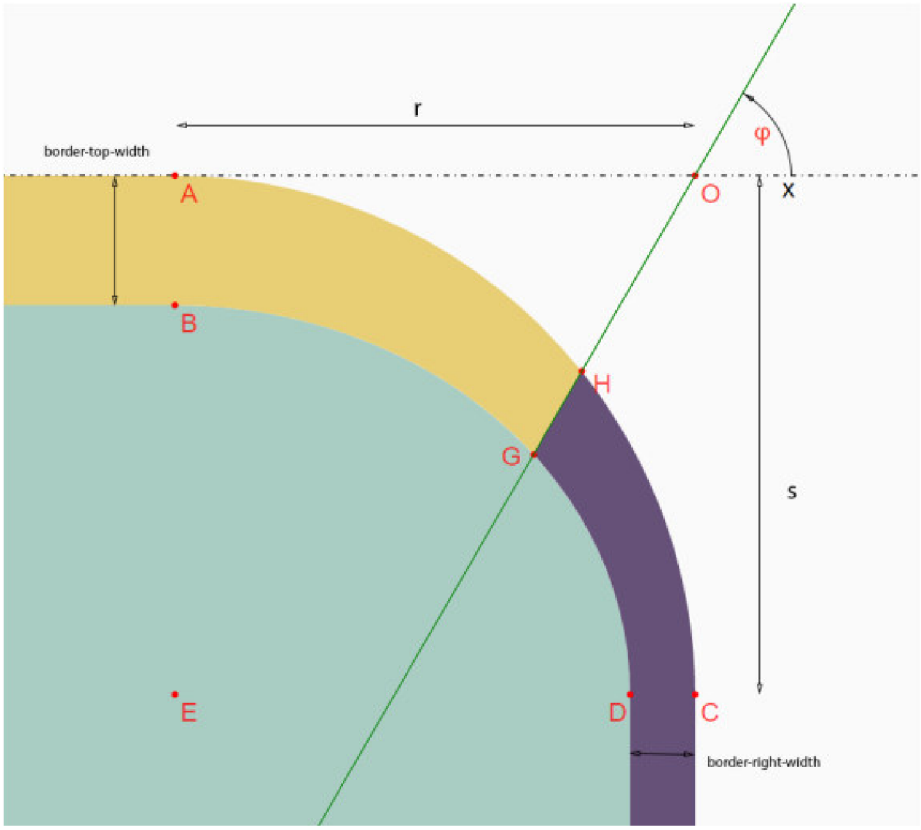
Hodnoty r a s definují délku hlavní a vedlejší poloosy, které určují míru zaoblení vnějšího okraje rámečku. Pokud je jedna hodnota nulová, zaoblený roh vykreslen není a místo něho je vykreslen roh ostrý. Pokud je zadána pouze jedna hodnota, je druhá hodnota totožná s hodnotou první. Vnitřní okraj rámečku je rovněž definován jako elipsa. Délky obou poloos jsou však zmenšeny o šířku rámečku. V případě rohu na obrázku 7.2 je od hlavní poloosy r odečtena šířka rámečku v horizontálním směru a od vedlejší poloosy je odečtena šířka rámečku ve vertikálním směru. Pokud je rozdíl délky poloosy a šířky rámečku, jak v jednom, tak v druhém směru roven 0, je vnitřní roh ostrý a nedojde u něj k zaoblení. Na obrázku 7.2 jsou zobrazeny body, které bude nutné získat, aby bylo možné vykreslit zaoblený roh.

K vykreslení zaobleného rohu je nutné získat body *A, B, C, D, E, G, H a O*. [19] Bod O reprezentuje roh elementu, pro který je rámeček vykreslován. Bod E reprezentuje střed elipsy a získá se odečtením nebo přičtením hodnot r a s od souřadnice O v závislosti na tom, který roh je vykreslován. Body A a C reprezentují začátek a konec vnějšího zaobleného rohu a získají se rovněž odečtením či přičtením hodnoty r nebo s od souřadnice O. Body B a D reprezentují začátek a konec vnitřního zaobleného rohu a pro jejich získání se musí odečíst nebo přičíst hodnota r nebo s, a navíc se musí odečíst nebo přičíst šířka rámečku v závislosti na vykreslovaném rohu.

Tyto body jsou dostačující pro vykreslení rohu v jedné barvě. CSS3 však umožňuje vytvářet uvnitř zaobleného rohu barevné přechody v případě, pokud mají dvě sousední strany rozdílnou barvu. Ve specifikaci CSS3 je řečeno, že se přechod barev musí nacházet v oblasti vymezené zaoblením rohu a že střed barevného přechodu se nachází na přímce, jejíž směrnice je určena poměrem šířek rámečků. Přesná poloha přímky je ponechána na interpretru.

K vykreslení tohoto barevného přechodu budou sloužit body G a H. Tyto body se získají pomocí průsečíku přímky s vnější a vnitřní elipsou. Směrnicový tvar přímky je následující:

$$y = kx + q \tag{7.1}$$



Obrázek 7.2: Body potřebné k vykreslení zaobleného rohu. [19]

Přímka může procházet bodem E (střed elipsy) nebo bodem O (krajní roh elementu). Směrnice přímky se následně získá jako tangens úhlu mezi přímkou a osou y.

$$\varphi = \frac{border - top - width}{border - top - width + border - right - width} \cdot \frac{\pi}{2} \tag{7.2}$$

$$k = \tan \varphi \tag{7.3}$$

Koeficient q se získá dosazením směrnice a souřadnic bodu O do rovnice. Tangens není definován pro $\frac{\pi}{2}$. Tento případ je ošetřen a nastane pouze v případě, že je šířka rámečku na jedné straně nulová.

Abychom získali body G a H, je potřeba nalézt průsečík elipsy s touto přímkou. To se řeší jako soustava dvou rovnic o dvou neznámých.

$$\begin{gathered} y = kx + q \\ \frac{(x - x_0)^2}{a^2} + \frac{(y - y_0)^2}{b^2} = 1 \end{gathered} \tag{7.4}$$

7.2.5 Generování zaobleného rohu

Kvůli problémům s přechodem barev uvnitř zaobleného rohu je generování rohu rozděleno na dvě části. V první části se generuje první polovina rohu. Konkrétně se jedná o cestu, která prochází body A, H, G a B viz. obrázek 7.2 a skládá se ze dvou rovných čar a dvou křivek.

Druhá polovina se pak vykreslí jako cesta, která prochází body C, H, G a D a rovněž se skládá ze dvou rovných čar a dvou křivek. Způsob vykreslení zaoblených křivek je pro verzi SVG a PDF rozdílný, jak bude následně vysvětleno.

Ke generování zaobleného rohu v SVG lze použít eliptické křivky. Je třeba znát pouze délku hlavní a vedlejší poloosy elipsy, podle níž je pak zaoblený roh vykreslen. Princip je totožný a jsou tedy použity stejné souřadnice, které jsou zadány v CSS3.

Problém nastává u verze PDF, a to z důvodu, že PDF eliptické křivky nepodporuje. Celý problém vykreslení zaobleného rohu se bude řešit pomocí Beziérových křivek. Bylo tudíž nutné vymyslet princip převodu eliptických křivek na Beziérovy křivky tak, aby bylo možné určit kontrolní bod pro Beziérovu křivku. Existují určité algoritmy převodu eliptických křivek pro Beziérovy, případně kubické Beziérovy křivky. Ty se dají použít, ale pouze v případě, že chceme zjistit kontrolní bod pro roh, který se skládá ze čtvrtinové elipsy či kruhu. Kvůli přechodu barvy uprostřed rohu musí být roh vykreslen ve dvou fázích. V jedné fázi je vždy vykreslena pouze 1/8 elipsy nebo kruhu. Bylo tedy nutné vymyslet výpočet pro tento převod, který bude obecně fungovat pro všechny možné konfigurace eliptické křivky. Zde bude popsán princip získání kontrolního bodu z parametrů pro eliptické křivky.

Princip vykreslení bude znázorněn na pravém horním vnějším rohu, konkrétně na jeho první polovině. Výpočet kontrolního bodu pro ostatní rohy je principiálně podobný. Liší se pouze v závislosti na otočení rohu.

Jelikož se zaoblený roh skládá ze dvou částí, nebude vykreslen celý roh současně pomocí jedné kubické Beziérovy křivky, ale oblouk se bude skládat ze dvou Beziérových křivek, kde každá bude mít pouze jeden kontrolní bod. Ten se získá tak, že se spočítá bod, který je přesně uprostřed mezi bodem A a bodem H, a to jak na ose x, tak na ose y. V obrázku 7.3 je tento bod znázorněn písmenem X. Následně spočítáme rovnici přímky, která prochází středem elipsy (bod E) a nově vzniklým bodem X. Nejdříve se na základě těchto bodů spočítá vektor, jehož jedna souřadnice je vynásobena hodnotou -1.

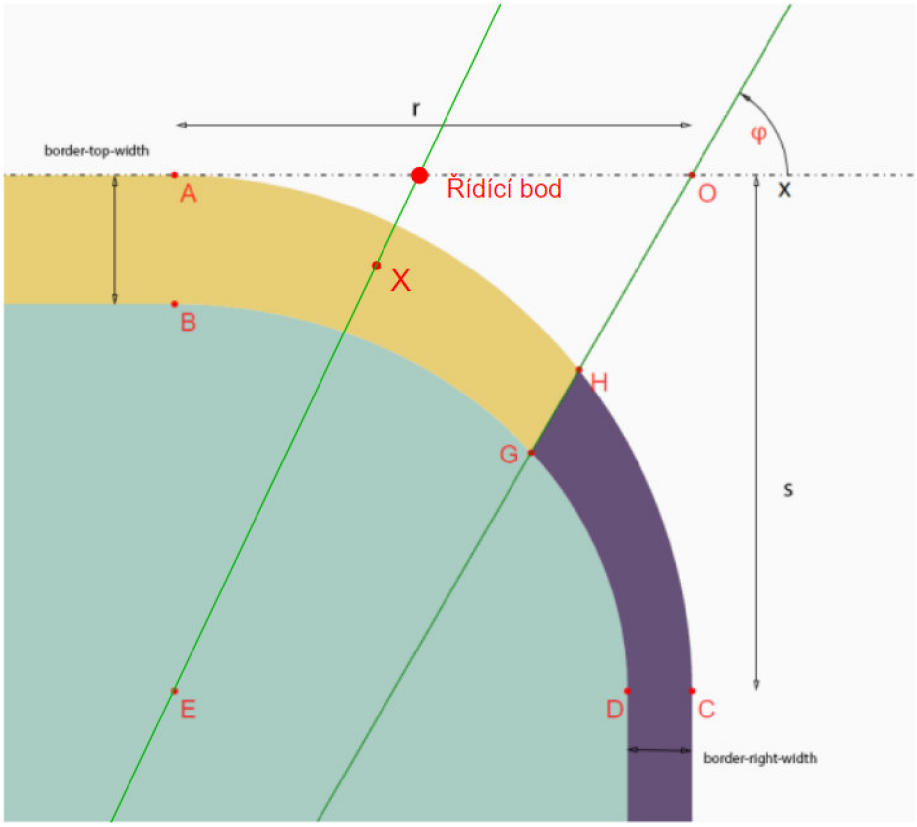
$$\begin{gathered} bodE[X_E, Y_E] \\ bodX[X_X, Y_X] \\ (X_E - X_X, -(Y_E - Y_X)) \end{gathered} \tag{7.5}$$

Z tohoto vektoru se vyjádří následně obecná rovnice přímky a na základě dosazení jednoho z bodů se dopočítá hodnota c.

$$ax + by + c = 0 \tag{7.6}$$

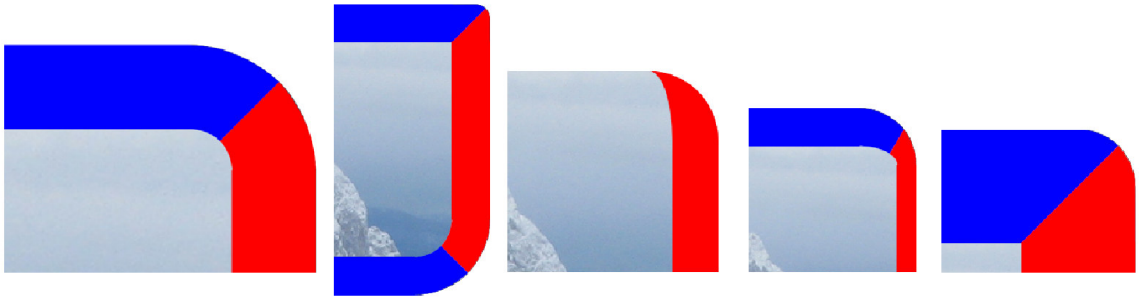
Po získání rovnice přímky se spočítá průsečík této přímky s vnějším okrajem elementu. Konkrétně pro první část pravého horního rohu se toto realizuje tak, že se do rovnice přímky dosadí za y hodnota bodu A a z této rovnice se získá hodnota x. Řídící bod vznikne ze získané hodnoty *x-ové* souřadnice bodu X a *y-nové* souřadnice bodu A. Tento princip je použit pro vykreslení ostatních rohů, pouze s rozdílem, že je roh otočen pod jiným úhlem.

Speciální případ nastává, pokud má jedna strana rámečku nulovou šířku. V tom případě zajistí vykreslení celého rohu pouze jedna polovina zaobleného rohu. Tento případ je v kódu ošetřen tak, že pokud je šířka rámečku, se kterým daná polovina rohu sousedí, nulová, tak se tato polovina rohu vůbec nevykreslí. A to z důvodu, aby nedocházelo k nekorektním výpočtům, které by způsobily chyby ve výsledném PDF kódu. Celý zaoblený roh je v tomto případě generován pomocí metody, která za běžné situace vykresluje polovinu tohoto zaobleného rohu.



Obrázek 7.3: Výpočet řídícího bodu pro Beziérovu křivku pro vykreslení zaobleného rohu.

Na obrázku 7.4 můžeme vidět některé speciální případy vykreslení zaobleného rohu v PDF.



Obrázek 7.4: Ukázka některých případů zaoblených rohů vykreslených pomocí PDF.

7.3 Generování textu

Dalším klíčovým prvkem pro generování webových stránek ve vektorové grafice je generování textu. Toto má za úkol metoda `renderTextContent()`, která se nachází ve třídách `PDFRenderer` a `SVGRenderer`. Způsob vykreslování textu u SVG a PDF se ve velké míře liší. Z tohoto důvodu nebylo možné příliš sjednotit postup generování textu. Dalším důvodem, proč nebylo možné sjednotit postup pro generování textu SVG a PDF, byl fakt, že knihovna

CSSBox nesmí obsahovat žádné externí knihovny, jako například knihovnu PDFBox. Přímo v CSSBoxu nebylo tudíž možné vytvořit třídu, která by daný text reprezentovala. Proto byla tato třída vytvořena pouze v rámci knihovny a generování textu budou mít na starost třídy `SVGRenderer` a `PDFRenderer`, které třídu `StructuredRenderer`, nacházející se v CSSBoxu, rozšiřují.

Byla vytvořena abstraktní třída `TextClass`, která bude reprezentovat daný text a která ponese společné informace o textu pro SVG i PDF. SVG a PDF mají však jinou reprezentaci fontu. SVG využívá pro reprezentaci fontu *String*, nesoucí název fontu. PDF používá třídu `PDFont` z knihovny `PDFBox`. Z tohoto důvodu byly přidány ještě třídy `TextClassSVG` a `TextClassPDF`, které abstraktní třídu `TextClass` rozšiřují a obsahují navíc reprezentaci použitého fontu pro jednotlivé vektorové formáty. Konstruktory těchto tříd zajistí získání všech informací potřebných pro generování textu z třídy `TextBox`, která je předána metodě `renderTextContent()` jako parametr.

7.4 Generování textu SVG

U SVG je generování textu poměrně snadné. Samotné generování spočívá v tom, že se vytvoří element, kterému jsou následně nastaveny atributy, jako pozice, výška, šířka, všechny styly textu a text, který se má vykreslit. Element je pak přidán jako potomek aktuálně zpracovávanému elementu. Jediná komplikace při vykreslení textu pomocí SVG nastává, pokud jsou explicitně nastaveny mezery mezi slovy v textu. V takovém případě nelze text zpracovávat běžným způsobem jako celek, ale musí být rozdělen na jednotlivá slova. Funkce vykreslení textu je tak volána na jednotlivá slova, mezi která jsou vkládány požadované mezery.

7.5 Generování textu PDF

Oproti SVG je vykreslení textu v PDF složitější. Největším problémem je šířka stránky. Strana v PDF je ve většině případů užší než webová stránka. Zde se nabízejí dvě možnosti. První je zmenšit obsah stránky tak, aby vizuálně odpovídal skutečnému rozložení stránky pouze v menším měřítku. Tato varianta však přináší nevýhodu v nečitelnosti původního textu v případě velmi široké webové stránky. Druhá varianta pak nabízí nechat text v původní velikosti, nebo zmenšit text pouze do té míry, aby byl čitelný. Text, který se na stránku už nevejde, je vykreslen níže a celý zbylý obsah stránky je o tuto část posunut. Toto přináší hlavní nevýhodu v tom ohledu, že vykreslená stránka pak nemusí vzhledově odpovídat původní webové stránce, především rozložením. V této práci byly naimplementovány obě dvě varianty s ohledem na jejich porovnání. Nakonec byla zvolena první varianta a bylo upřednostněno především stejné rozložení stránky, jako měla původní stránka. Problém s nečitelností textu při velmi širokých stránkách byl ponechán na autorovi dané webové stránky.

Souřadnice a velikost všech elementů jsou přepočítávány v určitém poměru vzhledem k vstupní a výstupní stránce. Velikost fontu však tímto získaným koeficientem být přepočítávána nemůže, protože poměrné zmenšení fontu daným koeficientem neodpovídá poměrnému posunutí souřadnic textu. To vytvářelo nechtěný efekt v textu. Pokud se text skládal v HTML kódu z více částí, které však na sebe na originální stránce navazovaly, v PDF se tvořily mezi těmito texty nechtěné mezery. Tento problém byl vyřešen tak, že se počítá

jiný koeficient pro text, který zmenší použitý font tak, aby odpovídal poměrnému posunutí textu a nevznikaly tak nechtěné mezery.

Explicitně se musí následně řešit podtržení textu a hypertextové odkazy. Podtržení textu se řeší pomocí obdélníku o minimální výšce, který je vykreslen na souřadnice pod textem.

Hypertextový odkaz je realizován taktéž pomocí obdélníku. Před textem, který hypertextový odkaz obsahuje, je vykreslen průhledný obdélník, kterému je pomocí knihovny PDFBox nastaven link na požadovanou stránku.

7.6 Generování seznamů

Pro generování seznamů je nutné kromě textu vykreslit i odrážky před textem. Ty jsou vykresleny v rámci metody `renderMarker()`. Odrážky mohou být geometrického tvaru, jímž nejčastěji bývá čtverec nebo kruh. Dalšími typy odrážek jsou odrážky, které jsou tvořeny znaky. Tento typ reprezentují například číslované seznamy. Jiným typem odrážek mohou být odrážky, které jsou reprezentovány jako obrázek. Princip spojení struktury pro vykreslení odrážek a oprava všech chyb při generování do obou dvou vektorových formátů zde budou nyní popsány.

Do třídy `StructuredRenderer` byl implementován společný princip pro vykreslení odrážek pro jazyk SVG i PDF. Zde se zjistí, zda mají být vykresleny odrážky jako obrázek, nebo jen za pomocí znaků a tvarů. V případě, že má být odrážka vykreslena pomocí obrázku, zavolá se metoda `writeMarkerImage()`, která se nachází v abstraktní třídě `StructuredRenderer` a je společná pro oba dva vektorové formáty. Tato metoda nejprve zkontroluje, zda je odrážka tvořená obrázkem. Pokud ano, provádí se dále zpracování, pokud ne, tak metoda vrátí false a pokračuje se ve vykreslení odrážky běžným způsobem. Pokud je odrážka tvořena obrázkem, získá se požadovaný obrázek a dopočítá se pozice obrázku na výstupní stránce. Následně se nad těmito parametry zavolá metoda `CreateImageBullet`. Tato metoda provede potřebné kroky k vygenerování obrázku reprezentující odrážku. Tyto kroky se liší pro verzi SVG i PDF, proto je tato metoda odlišná pro oba dva vektorové formáty a její implementace se tak nachází ve třídách `SVGRenderer` a `PDFRenderer`.

Pokud není odrážka realizovaná za pomoci obrázku, má na starost její vykreslení metoda `writeBullet()`. Tato metoda je implementována ve třídě `StructuredRenderer` a obsahuje společný postup pro verzi SVG i PDF. V této metodě se spočítá správná pozice pro danou odrážku. Následně se volá příslušná metoda podle toho, jakého tvaru nebo typu odrážka je. Může se jednat buďto o odrážku ve tvaru kruhu, čtverce, plného kruhu nebo o odrážku realizovanou textovými znaky. Metody realizující vykreslení příslušného typu jsou pak implementovány ve třídách `PDFRenderer` a `SVGRenderer`, protože obsahují konkrétní postupy pro daný jazyk, které nebylo možné sjednotit do společného postupu.

7.7 Stránkování PDF

V PDF se na rozdíl od SVG musí řešit stránkování. Celý obsah musí být rozdělen na stránky, přičemž musí být brán ohled na vlastnosti CSS, které vynucují nebo zakazují přechod stránky v daném místě, pokud je to možné. Tyto vlastnosti byly vysvětleny výše v kapitole 5.7.

Tato práce navazuje na existující akademické práce [18], kde se stránkování v PDF již nějakým způsobem řešilo. V těchto pracích však stránkování v PDF bylo nefunkční, pokud některý prvek na stránce obsahoval vlastnost vynucující nebo zakazující vytvoření

nové stránky. Z tohoto důvodu byl analyzován celý způsob stránkování, který byl následně upraven a opraven. Některé principy byly převzaty z práce *Generování PDF dokumentů z webových stránek* [18], jiné byly upraveny nebo znovu vytvořeny tak, aby vyhovovaly aktuální struktuře a způsobu generování PDF dokumentu. Hlavním problémem původního stránkování v aktuální struktuře byla nefunkčnost některých vlastností pro stránkování a chybný výpočet souřadnic jednotlivých prvků, který vedl k neočekávanému výstupu.

Většina částí ke zpracování stránkování musela být upravena. Pro tuto část práce bylo stěžejní pochopit princip, jakým zpracování stránkování funguje. I když byly některé principy převzaty z předchozí práce, bude zde z tohoto důvodu vhodné popsat celý princip, jakým způsobem stránkování po opravě a úpravě probíhá nyní.

Prvním zásadním rozdílem je, že se stránkování tvoří již před zpracováním prvního elementu, který je instancí třídy *Viewport*. Při zpracování dalších elementů je již stránkování připraveno. Není tak nutné v prvním kroku procházet všechny prvky, které mají být zpracovány, a až následně provádět jejich generování pomocí druhého průchodu. Dalším velkým rozdílem je, že v původním řešení bylo nutné vytvářet dvě datové struktury TREE a LIST. Pro stránkování je nyní potřeba pouze jedna struktura TREE, která je získána z DOM stromu elementů a obsahuje všechny informace potřebné ke stránkování.

7.7.1 Datová struktura TREE

Položka datové struktury TREE je realizována jako třída *Node*. Třída *Node* obsahuje následující atributy:

- **Node** nodeParent - reference na rodičovský uzel.
- **Vector<Node> nodeChildren** - obsahuje reference na potomky uzlu. V případě listového uzlu obsahuje hodnotu *null*.
- **ElementBox elem** - pokud instance reprezentuje blok typu element, bude obsahovat referenci na instanci třídy *ElementBox*.
- **Textbox text** - pokud instance reprezentuje text, bude obsahovat referenci na instanci třídy *TextBox*.
- **ReplacedBox box** - pokud instance reprezentuje obrázek tvořený tagem *<img>*, bude obsahovat referenci na instanci třídy *ReplacedBox*.
- **ListItemBox item** - pokud instance reprezentuje odrážku seznamu, bude obsahovat referenci na instanci třídy *ListItemBox*.
- **float plusHeight, plusOffset** - reprezentuje roztažení a posunutí daného prvku.
- **int parentIDOfNoninsertedNode** - ID rodiče uzlu, který není možné momentálně zařadit do struktury TREE a bude zařazen do této struktury později.

7.7.2 Tabulky breakTable a avoidTable pro vytvoření stránkování

V této kapitole bude popsán princip, podle kterého jsou vytvořeny tabulky *breakTable* a *avoidTable* a způsob, jakým probíhá jejich následné zpracování. Tvorba těchto tabulek a realizace následného stránkování je implementována ve třídě *BreakAvoidTables*.

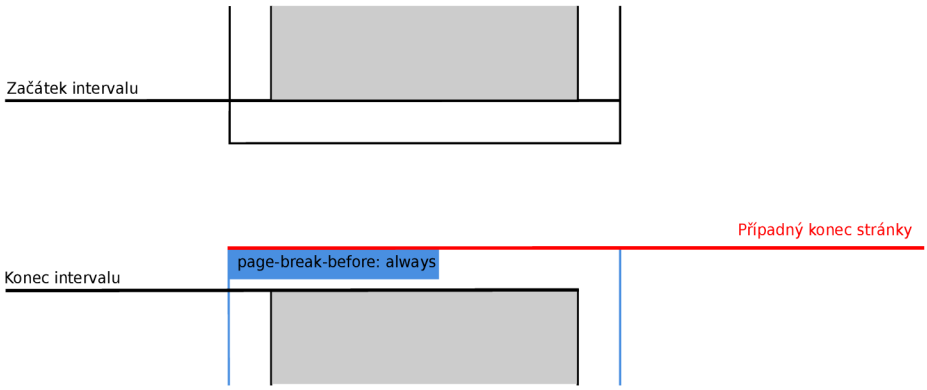
Obě dvě tabulky jsou vytvořeny na základě datové struktury TREE. Na základě vlastností CSS, které byly popsány v kapitole 5.7, jsou vloženy záznamy do tabulky *breakTable* nebo *avoidTable*. Každý záznam obsahuje následující tři položky:

- Začátek intervalu.
- Konec intervalu.
- Místo pro vytvoření nové stránky.

V následujících odstavcích bude popsán princip, jakým způsobem jsou vytvořeny záznamy v obou tabulkách na základě CSS vlastností vynucujících nebo zakazujících vytvoření nové stránky v daném místě. Element, který má danou CSS vlastnost, je na obrázku 7.5 ohraničen modrým rámečkem. Místo konce stránky je v každém z případů označeno červenou čárou s popiskem.

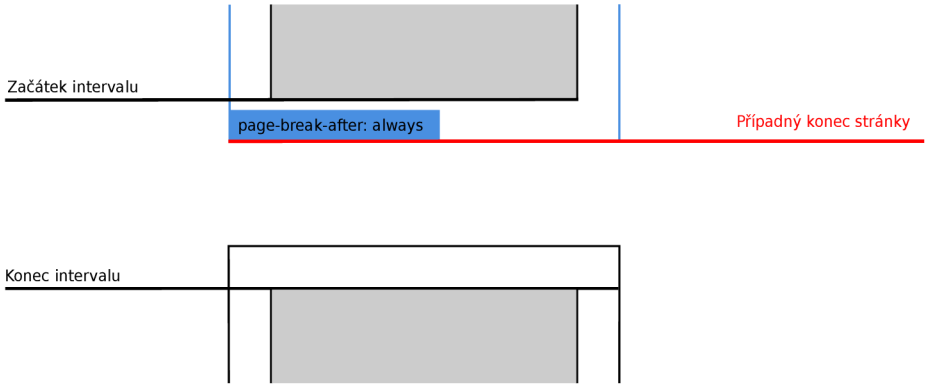
Vlastnost ***page-break-before: always*** je znázorněna na obrázku 7.5. Nový záznam bude v tomto případě vložen do tabulky *breakTable*. Začátek intervalu je dán spodní hranicí posledního synovského elementu daného elementu, který se nachází nad elementem, který má nastavenou tuto vlastnost CSS. Konec intervalu je určen vrchní hranicí prvního synovského elementu elementu s touto vlastností CSS. Místo, kde dojde k případnému zalomení stránky, je určeno vrchní hranicí elementu s touto vlastností CSS.

Při prohledávání všech synovských elementů jsou upřednostňovány elementy, které nejsou typu *ElementBox*. Toto platí i pro všechny následující vlastnosti zakazující nebo vynucující stránkování. V tomto konkrétním případě, pokud element neobsahuje žádné synovské elementy, začátek intervalu bude dán spodní hranicí elementu, který předchází elementu s touto vlastností CSS. Konec intervalu pak bude v případě nutnosti tvořit vrchní hranice elementu s touto vlastností CSS.



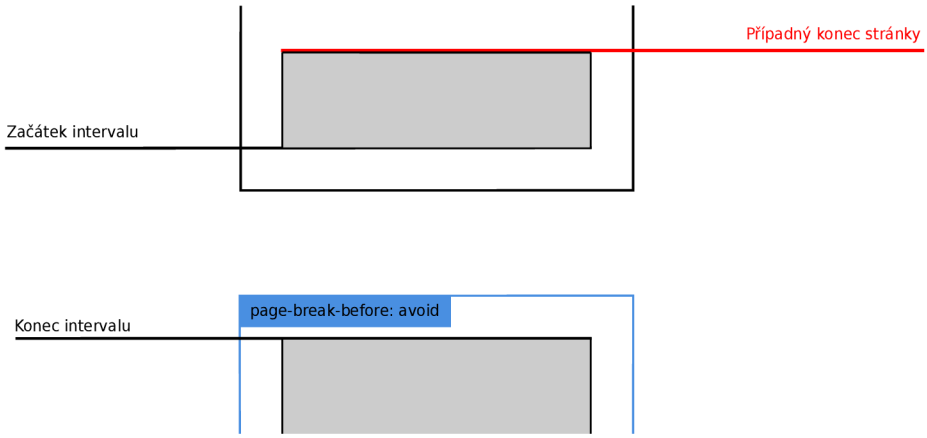
Obrázek 7.5: Interval a místo zalomení stránky pro vlastnost *page-break-before: always*.

Vlastnost ***page-break-after: always*** je znázorněna na obrázku 7.6. Nový záznam bude v tomto případě rovněž vložen do tabulky *breakTable*. Začátek intervalu je dán spodní hranicí posledního synovského elementu elementu s touto vlastností CSS. Konec intervalu je dán vrchní hranicí prvního synovského elementu, který se nachází pod elementem s touto vlastností CSS. Místo, kde dojde k případnému zalomení stránky, je určeno spodní hranicí elementu, který má nastavenou tuto vlastnost CSS. Pokud by element neobsahoval synovské elementy, bude začátek intervalu určen spodní hranicí elementu s vlastností CSS nebo konec bude určen vrchní hranicí elementu, který následuje za elementem s touto vlastností CSS.



Obrázek 7.6: Interval a místo zalomení stránky pro vlastnost *page-break-after: always*.

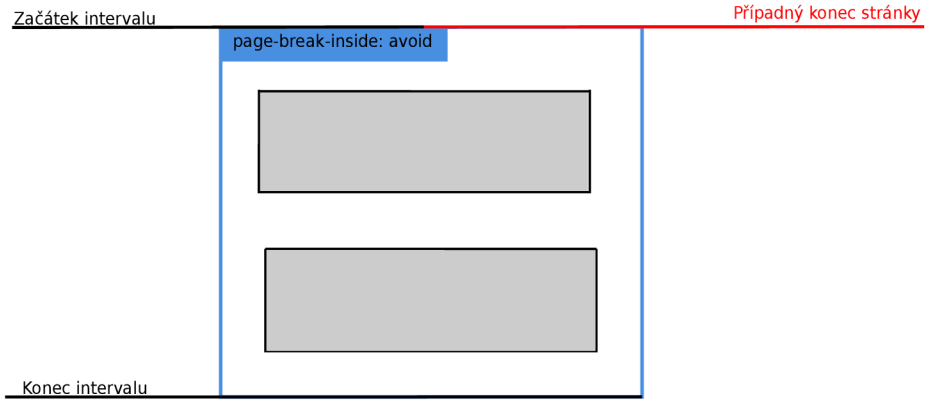
Vlastnost ***page-break-before: avoid*** je znázorněna na obrázku 7.7. Tentokrát bude nový záznam vložen do tabulky *avoidTable*. Začátek intervalu je dán spodní hranicí posledního synovského elementu daného elementu, který bezprostředně předchází elementu, který má nastavenou tuto vlastnost CSS. Konec intervalu je určen vrchní hranicí prvního synovského elementu elementu s touto vlastností CSS. Místo, kde dojde k případnému zalomení stránky, je určeno vrchní hranicí elementu, který se nachází bezprostředně nad elementem s touto vlastností CSS. Pokud by některý element neobsahoval synovské elementy, bude začátek a konec intervalu určen stejně, jako v případě vlastnosti *page-break-before: always*.



Obrázek 7.7: Interval a místo zalomení stránky pro vlastnost *page-break-before: avoid*.

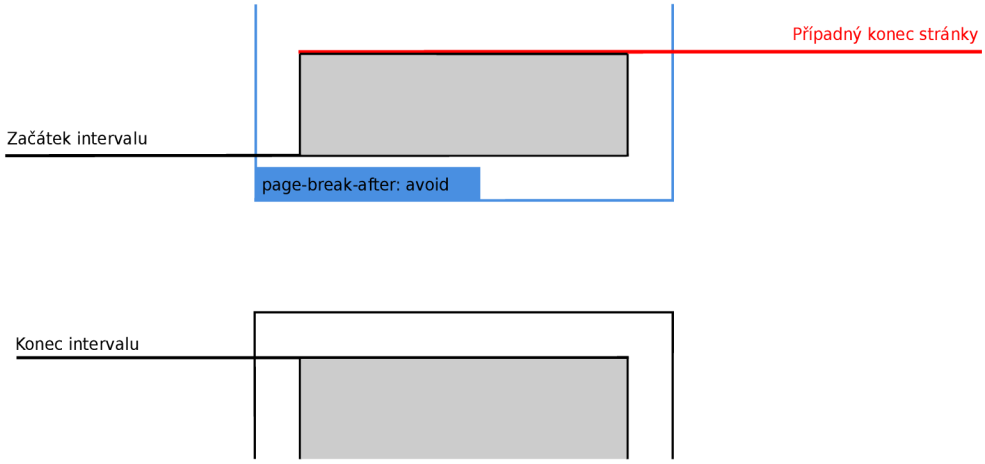
Vlastnost ***page-break-inside: avoid*** je znázorněna na obrázku 7.8. Nový záznam bude vložen do tabulky *avoidTable*. Začátek intervalu je dán vrchní hranicí elementu s touto vlastností CSS a konec intervalu je dán spodní hranicí elementu s touto vlastností CSS. Místo, kde dojde k případnému zalomení stránky, je určeno nad vrchní hranicí tohoto elementu s danou vlastností CSS.

Vlastnost ***page-break-after: avoid*** je znázorněna na obrázku 7.9. Záznam bude vložen do tabulky *avoidTable*. Začátek intervalu je dán spodní hranicí posledního synovského elementu toho elementu, který má nastavenou tuto vlastnost CSS. Konec intervalu je pak dán vrchní hranicí prvního synovského elementu toho elementu, který následuje za elementem s touto vlastností CSS. Pokud by element neobsahoval žádné synovské elementy, bude



Obrázek 7.8: Interval a místo zalomení stránky pro vlastnost *page-break-inside: avoid.*

začátek intervalu určen spodní hranicí elementu s touto vlastností CSS a konec bude určen vrchní hranicí elementu, který se nachází pod elementem s touto vlastností CSS.



Obrázek 7.9: Interval a místo zalomení stránky pro vlastnost *page-break-after: avoid.*

Pro nalezení elementu nad nebo pod daným elementem slouží funkce *getElementAbove()* a *getElementBelow()*. Pro nalezení vrchní hranice prvního synovského elementu nebo spodní hranice posledního synovského elementu slouží funkce *getFirstTop()* a *getLastBottom()*. Princip těchto funkcí se nijak výrazně nezměnil, byly pouze opraveny chyby při výpočtu souřadnic elementů.

Po sestavení tabulek *breakTable* a *avoidTable* je nutné záznamy v obou tabulkách upravit. Zejména jde o sloučení intervalů, které se překrývají, nebo vymazání intervalů v tabulce *avoidTable*, jejichž interval je větší než 80 % stránky. Toto se dělá z důvodu, aby nebyl narušen proces stránkování. Tyto zmíněné funkce mají na starost metody *mergeAvoids()* a *deleteAvoidsBiggerThan()*. Tyto metody nebyly nijak modifikovány a byly převzaty z minulé práce. [18] Proto zde jejich princip nebude podrobněji popsán.

7.7.3 Zalomení stránky

Zalomení stránky, s ohledem na záznamy v tabulkách *breakTable* a *avoidTable*, má na starost metoda `makePaging()`. Tato metoda byla předělána z důvodu nefungujícího stránkování s využitím obou výše zmíněných tabulek.

Metoda `makePaging()` probíhá do té doby, dokud je nějaký záznam v tabulce *breakTable*, nebo dokud ještě nebylo dosaženo konce dokumentu. Zde je stránkování rozděleno na několik typů a na jejich základě je volána metoda `makeBreakAt()` s příslušným parametrem, který určuje místo zlomu stránky. O metodě `makeBreakAt()` bude pojednáno níže.

- Pokud se v tabulce *breakTable* nevyskytuje žádný záznam, nebo pokud se případný konec stránky vyskytuje nad prvním záznamem v tabulce *breakTable*, pokračuje se ve stránkování na základě velikosti stránky výsledného dokumentu. Předtím je ale nutné projít záznamy v tabulce *avoidTable*, zda se případný konec stránky nevyskytuje v nějakém intervalu, kde je zalomení stránky zakázáno. Pokud ano, je zavolána metoda `makeBreakAt()` se souřadnicí zalomení stránky získaného z tabulky *avoidTable*. Pokud ne, je metoda `makeBreakAt()` zavolána se souřadnicí určující aktuální konec stránky.

- Druhý případ nastává, pokud je běžný konec stránky v nějakém intervalu v tabulce *breakTable*. Pokud je u daného záznamu v tabulce *breakTable* případný konec stránky nad běžným koncem stránky, je zavolána metoda `makeBreakAt()` se souřadnicí zalomení stránky z tabulky *breakTable*. V opačném případě je metoda `makeBreakAt()` zavolána se souřadnicí určující běžný konec stránky.

- Poslední případ je pro záznamy z tabulky *breakTable*, pro které neplatí dva předchozí případy. V tomto případě je volána funkce `makeBreakAt()` se souřadnicí z tabulky *breakTable* patřící konkrétnímu záznamu.

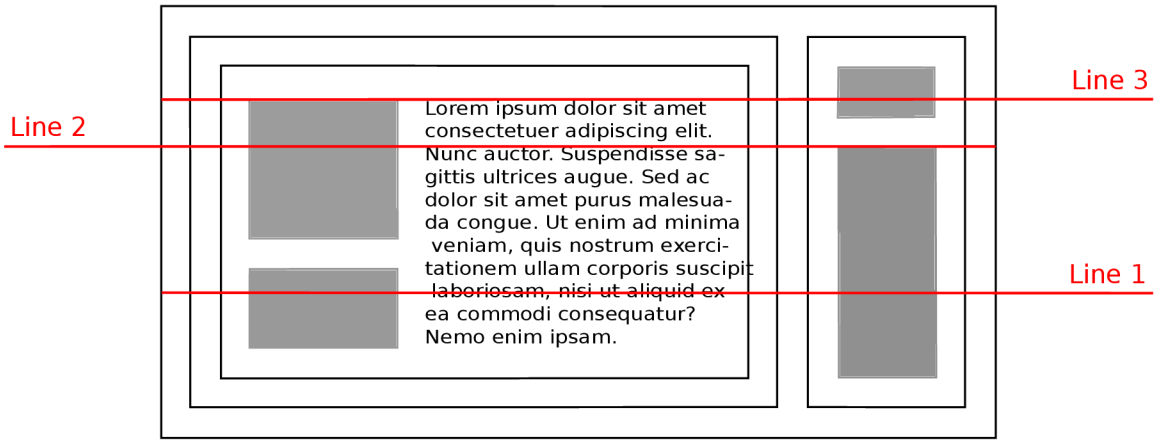
Zde byla ještě doimplementována kontrola pro vynucené stránkování. Kontroluje, zda již daná podmínka není splněna. Například pokud by měl prvek vlastnost *page-break-before: always* a již při normálním stránkování by se nacházel na začátku nové stránky, je zbytečné vkládat před něj zalomení stránky a vytvářet tak jednu volnou stránku nad tímto elementem. Obdobně to platí i pro vlastnost *page-break-after: always*. Tato kontrola je realizována v metodě `elemIsOnStrartOfPage()`.

7.7.4 Realizace zalomení stránky

Metoda *breakTable()* realizuje zalomení stránky v místě, které určuje souřadnice přijímaná jako parametr metody. V této metodě byl upraven především princip získávání a počítání souřadnic jednotlivých elementů. Také zde probíhá kontrola protnutí konce stránky na prvky patřící do skupiny *replacement content*. Princip počítání *line2* a *line3*, jak bude vysvětleno níže, byl ponechán v původním stavu. Z důvodu, že pochopení tohoto principu je klíčové pro mnou provedené opravy, bude zde celý princip počítání hranic popsán.

Zalomení stránky je realizováno pomocí posunutí či roztažení elementů. Tato funkce musí navíc řešit korektnost zalomení stránky. Může nastat případ, kdy případné zalomení stránky prochází obrázkem či textem. V této funkci jsou pro řešení stránkování sestaveny tři hranice:

- **line1** - tato hranice je funkci *breakTable* předána jako parametr. Zalomení na této hranici je však možné pouze v případě, že tato hranice neprotíná žádný obrázek ani



Obrázek 7.10: Ukázka realizace zalomení stránky a výpočtu jednotlivých hranic pro zalomení stránky.

text. V takovém případě dojde k nalezení všech prvků typu *ElementBox*, které protíná hranice *line1* a tyto elementy roztáhnou svou výšku. Všechny ostatní elementy jsou posunuty na další stránku.

- **line2** - ve většině případů však hranice *line1* protíná kromě prvků typu *ElementBox* i jiné typy prvků. Proto je nutné určit hranici *line2*, případně *line3*, podle kterých se bude stránka zalamovat v těchto případech. Hranice *line2* vznikne tak, že se projdou všechny prvky, které nejsou typu *ElementBox* a z těchto prvků se vybere takový, který má nejmenší vzdálenost od vrcholu stránky. Na jeho horní hranici je pak umístěna hrana *line2*. Podle hranice *line2* dojde k zalomení stránky.

- **line3** - hranice *line3* řeší speciální případ, který může nastat, pokud hranice *line2* prochází prvkem, který není typu *ElementBox*. Může se jednat o obrázek nebo text a tento prvek není možné roztáhnout, proto je nutné ho zobrazit již na další stránce. Na základě hranice *line3* je tedy určena vzdálenost, o kterou budou prvky posunuty nebo roztaženy.

Nyní máme určeny všechny tři hranice potřebné k zalomení stránky a dojde tudíž k zalomení stránky podle hranice *line2* následujícím způsobem:

- elementy, které jsou celé nad hranicí *line2*, zůstanou beze změny,

- elementy, které jsou instancí třídy *ElementBox* a jsou protnuté hranicí *line2*, budou roztaženy o vzdálenost, která se vypočítá na základě velikosti stránky a hranice *line1* a *line3*,

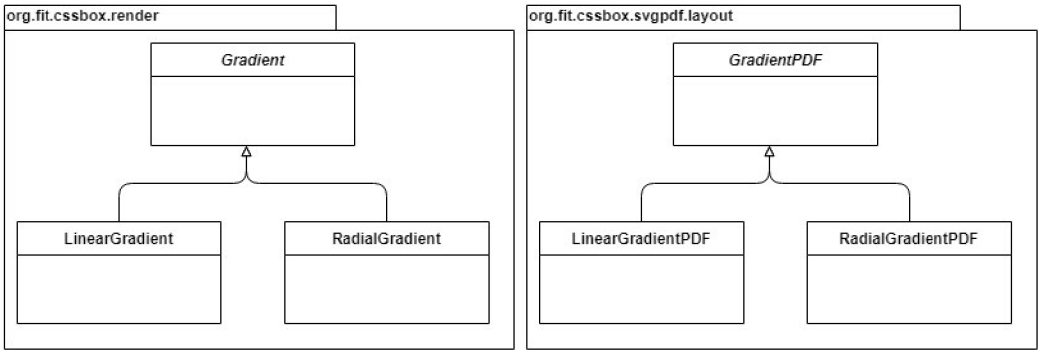
- všechny ostatní elementy budou o danou vzdálenost posunuty.

Na konci procesu zalomení stránky je nutné aktualizovat záznamy v tabulkách *avoidTable* a *breakTable*. To je nutné z důvodu, že došlo k posunu prvků, které byly na konci strany. Tímto došlo k posunu všech prvků, které se nacházely pod nimi, a tudíž musí být posunuty i intervaly pro vlastnosti prvků zakazující nebo vynucující zalomení stránky v daném místě, které byly vypočítány již dříve a nyní by nemusely být aktuální.

Výše zmíněný princip zajistí, aby text nebo obrázek v podobě *replacement content* nebyl vykreslen na předělu stránky, ale byl posunut na stranu následující. Tímto se zabrání nechtěnému efektu vykreslení vrchní části textu na konci jedné stránky a vykreslení spodní části textu na druhé stránce. Toto by při tisknutí dokumentu přineslo nečitelnost textu, protože by text ani na jedné stránce nebyl kompletní. Ostatní elementy, u kterých nevadí, že jimi prochází zalomení stránky, byly naimplementovány tak, aby se vykreslovaly na obou stránkách. S ohledem na tento fakt musely být navrženy a naimplementovány výše zmíněné prvky, jako například zaoblení rohů nebo pozadí elementu.

7.8 Vytvoření společné struktury pro vykreslování gradientu

Implementace SVG i PDF, ze které jsem vycházel, již obsahovala implementace pro lineární a radiální gradient. Obě práce však mezi sebou nesdílely žádné části. Při výpočtu a vykreslení gradientů může být přitom velká část implementace společná pro oba dva vektorové formáty. V implementaci, ze které tato práce vychází, byl gradient pro SVG funkční, pro PDF však nikoli. Provedl jsem zprovoznění obou gradientů a jejich začlenění do aktuální struktury a spojení principu vykreslení pro oba dva vektorové formáty. Detaily samotného generování gradientů v jazyce PDF a SVG zde nebudou popsány, jelikož byly převzaty z minulých prací [19] [7] a byly upraveny tak, aby měly co možná nejvíce společných částí a aby se daly zasadit do nové struktury. Proto bude tento odstavec pojednávat výhradně o spojení principů SVG a PDF pro gradienty a vytvoření nové struktury. Na obrázku 7.11 můžeme vidět aktuální strukturu tohoto projektu pro gradient.



Obrázek 7.11: Nová struktura pro vykreslování gradientu v PDF a SVG.

Třídy *Gradient*, *LinearGradient* a *RadialGradient* se nachází v projektu CSSBox. Abstraktní třída *Gradient* obsahuje společné atributy pro lineární i radiální gradient. Mezi tyto atributy patří především list prvků *GradientStop*, který reprezentuje všechny barvy použité pro gradient. Třída *GradientStop* pak obsahuje danou barvu, pozici dané barvy, a to jak v procentech, tak v pixelech.

Z třídy *Gradient* pak dědí třídy *LinearGradient* a *RadialGradient*. Reprezentace souřadnic lineárního a radiálního gradientu se liší, a proto je obsažena v těchto dvou třídách. V nich se nachází veškeré souřadnice a výpočty potřebné k vykreslení gradientu v PDF i v SVG.

Ve struktuře se dále nachází i třídy *LinearGradientPDF* a *RadialGradientPDF*. Tyto třídy obsahují pouze metody pro formát PDF, jak jejich název napovídá. Tyto třídy byly vytvořeny z důvodu, že k vykreslení gradientu v PDF je potřeba vytvořit objekt *PDShadingType3* pro radiální gradient, případně *PDShadingType2* pro lineární gradient. Tyto

třídy pocházejí z knihovny *PDFBox*. Podmínkou pro projekt CSSBox však je, že nesmí obsahovat žádné podobné externí knihovny. Z tohoto důvodu nejsou tyto dvě třídy v projektu CSSBox, ale pouze v rámci nově vytvořené externí knihovny a rozšiřují abstraktní třídu *GradientPDF*, která sdružuje společné pomocné metody pro generování lineárního a radiálního gradientu v PDF.

Dále byly upraveny především metody pro výpočet gradientu v PDF. Velké množství metod pro dopočítání chybějících hodnot pro gradient bylo duplicitních. Většina metod byla řešená už ve verzi SVG elegantnějším způsobem. Celý princip PDF byl následně upraven tak, aby principem odpovídal verzi SVG i s využitím metod pro výpočet potřebných bodů.

Vykreslení gradientu pro verzi SVG i PDF bylo vloženo do nové struktury. O tom, zda se bude vykreslovat gradient, se rozhoduje v abstraktní třídě `StructuredRenderer` v metodě `renderElementBackground()`. Pokud pozadí obsahuje gradient, je zavolána metoda `addGradient()`. Tato metoda se rovněž nachází v abstraktní třídě `StructuredRenderer` a je společná pro verzi SVG i PDF. V této metodě se získá instance třídy `Gradient` a spočítá se pozice obdélníku, kde se bude výsledný gradient na výstupní stránce nacházet. Z instance třídy `Gradient` je nutné následně zjistit, zda se jedná o gradient lineární či radiální. Na základě této podmínky je následně volána metoda pro vykreslení lineárního nebo radiálního gradientu v příslušném vektorovém formátu.

7.9 Vytvoření společné struktury pro transformace

Implementace, ze které jsem vycházel, již obsahovala transformace, a to jak ve verzi PDF, tak ve verzi SVG. Ve stavu, ve kterém jsem projekt převzal, však implementace nebyla funkční. Bylo nutné upravit ve verzi SVG způsob získávání parametrů pro transformace a ve verzi PDF předělat způsob aplikování transformací tak, aby jej bylo možné začlenit do aktuální struktury projektu.

SVG i PDF požadují odlišný způsob aplikace transformací, který bude vysvětlen podrobněji. Z tohoto důvodu nebylo možné sjednotit některé funkce pro získání nebo aplikování transformací. Vznikly tak dvě třídy *TransformPDF* a *TransformSVG*, které obsahují implementaci transformací pro daný vektorový formát. V obou dvou třídách byl pro lepší orientaci zaveden podobný postup pro kontrolu a nastavení jednotlivých transformací, který byl však ponechán odděleně. Případné sloučení tohoto postupu by rozsah daného kódu ještě více zvětšilo.

Verze SVG měla problém se získáváním parametrů pro jednotlivé transformace z vlastností elementu. Tato chyba mohla být způsobena změnami, které byly od doby vytvoření v projektu CSSBox provedeny. Způsob získání a čtení parametrů transformací tak nebyl kompatibilní s danou implementací transformací pro SVG. Tento problém byl opraven a transformace byly přidány do struktury tak, aby mohly být aplikovány na vykreslovaný element. Kontrola, zda element neobsahuje transformace a jejich případné použití, byla přidána do metod `renderElementBackground()` a `StartElementContent()` ve třídě `SVGRenderer`. V první metodě jsou transformace aplikovány na elementy typu *ElementBox*. Tato metoda se stará o transformace pozadí a rámečků elementu. V druhé metodě jsou transformace aplikovány na ostatní elementy, jako například texty nebo obrázky.

Kontrola a získání případných transformací jsou implementovány v metodě `insertTransformSVG()` ve třídě `TransformSVG`. Zde je nejdříve získána původní pozice daného elementu a vlastnosti, které určují konkrétní transformace případného elementu. Dále jsou získány souřadnice `ox` a `oy` určující bod, kolem kterého bude transformace probíhat. Tento bod přestavuje střed daného elementu. Pokud element neobsahuje žádnou transformaci,

vrací metoda prázdný řetězec. V případě, že element nějakou transformaci obsahuje, je následně získán list termů, které představují jednotlivé transformace. Dále se v cyklu prochází všechny termy a kontroluje se, zda se jedná o nějaký implementovaný typ transformace. Pokud ano, přidá se ke *Stringu*, který je použit následně jako návratová hodnota metody a který definuje danou transformaci. Pro získání těchto řetězců jsou ve třídě `TransformSVG` implementovány metody, které pro každý typ transformace převedou vlastnosti elementu určující transformaci na *String*, který je po provedení metody přiřazen vykreslovanému SVG elementu jako atribut určující transformaci SVG prvku ve výstupním souboru.

Princip vykreslování jednotlivých elementů pro PDF se v nové struktuře liší. Z tohoto důvodu bylo nutné začlenit princip vykreslování transformací do PDF jiným způsobem. Před samotným generováním prvku je nejprve nezbytné zavolat metodu *saveGraphicsState()*. Následně se aplikuje požadovaná transformace a vykreslení elementu. Nakonec je volána metoda *restoreGraphicState()*. Mezi těmito procesy nemůže dojít k ukončení streamu obsahu, ke kterému dochází v případě, kdy dojde k přepnutí stránky. Prvně se aplikují všechny transformace a následně se provede vykreslení daného prvku. Ten se skládá z pozadí, rámečků, zaoblených rohů apod., které jsou vykreslovány postupně. Z tohoto důvodu nesmí docházet k přepínání stránek během vykreslování jednotlivých částí. Metoda pro přepínání stránek byla upravena tak, aby proběhlo přepnutí stránky pouze v případě, pokud se stránka, která se má nastavit jako aktuální, liší od stránky, která je právě aktuální. Tímto nedojde k ukončení streamu v průběhu vykreslování elementu a je tedy možné použít transformace v aktuální struktuře projektu.

Celý proces transformace musel být rozdělen na dvě části. První část je aplikována před vykreslováním samotného elementu a má ji na starost metoda `transformIn()`. V této metodě proběhne získání požadovaných transformací a aplikování transformace pomocí knihovny PDFBox. Získání transformací má na starost metoda `insertTransformPDF()`, která má stejnou strukturu jako metoda `insertTransformSVG()` v případě SVG. Zde je nejdříve získána původní pozice daného elementu a vlastnosti, které určují konkrétní transformace případného elementu. Dále jsou získány souřadnice `ox` a `oy` určující bod, kolem kterého bude transformace probíhat. Následně je získán list termů představujících jednotlivé transformace. V cyklu se projdou všechny termy a probíhá kontrola, zda se nejedná o nějaký implementovaný druh transformace. Zde transformace není reprezentovaná *Stringem*, jako v případě verze SVG, ale je reprezentovaná třídou `AffineTransform`. Tato třída již obsahuje metody pro nastavení jednotlivých transformací. Je proto nutné získat nebo převést parametry jednotlivých transformací a zavolat příslušnou metodu. Pokud metoda `insertTransformPDF()` nevrací *null*, pokračuje se v aplikaci transformace. Zde je nutné převést body `ox` a `oy` do souřadnicového systému PDF. Následně je zavolaná funkce vytvořená pro aplikaci transformace pomocí knihovny PDFBox. V této metodě se instance třídy `AffineTransform` převede na matici, celý element je posunut dle souřadnic `ox` a `oy`, aplikuje se daná transformace pomocí matice a celý element se vrátí na původní pozici.

V PDF se nachází aplikace transformace na třech místech. V metodě *renderElementBackground()* pro transformaci prvků typu *ElementBox* a následně potom v metodách, které mají na starost vykreslování obrázků a textu.

7.10 Vytvoření společné struktury pro filtry

Jak v SVG, tak v PDF byly implementovány filtry na obrázky. Implementaci filtrů pro SVG i PDF se podařilo zcela sjednotit. Vznikla tak pouze jediná třída *Filter*, která obsahuje veškeré funkce týkající se použití filtrů na obrázky.

V SVG i v PDF probíhá vytvoření filtrů v metodách, které mají na starost generování elementů typu *replacement content*. Nejdříve se z elementu získají CSS vlastnosti, které říkají, zda je na obrázek použit nějaký filtr. Na základě těchto vlastností se vytvoří instance třídy `Filter`, která nese veškeré nastavení použitých filtrů. Pokud je na obrázek použit nějaký filter, proběhne následně porovnání s předem daným seznamem implementovaných filter vlastností, a pokud se shoduje filter vlastnost s některou implementovanou vlastností, pokračuje se v aplikaci daného filtru na obrázek. Následně proběhne vytvoření filtru pomocí metody `createFilter()`, která se nachází ve třídě `Filter`. V této třídě se projdou všechny implementované filter vlastnosti, které daný prvek obsahuje a je získáno jejich nastavení. Potom je na základě vlastností vytvořena instance třídy `Filter`, která je vrácena jako návratová hodnota funkce.

Vytvořený filtr se aplikuje na obrázek. O to se stará metoda `filterImg()` nacházející se rovněž ve třídě `Filter`. Tato funkce zkontroluje, o jaký typ filtru se jedná a následně zavolá funkci, která implementuje aplikaci filtr na původní obrázek. Aplikaci filtrů mají na starost rovněž metody, které jsou implementované ve třídě `Filter`. Invertování barev realizuje metoda `invertImg()`. Světlost a průhlednost realizuje metoda `setBrightOpacImg()` a převedení obrázků do tónů šedi realizuje metoda `grayScaleImg()`. Funkce pro invertování barvy a funkce pro tóny šedi fungují na podobném principu. Postupně se projde obrázek pixel po pixelu, přičemž se hodnoty jednotlivých pixelů mění v závislosti na použitém filtru. Úprava světlosti a průhlednosti obrázku je realizována pomocí již implementovaných metod ze třídy `RescaleOp`. Jako návratová hodnota metody `filterImg()` je nakonec vrácen již upravený obrázek. Vykreslení obrázku dále probíhá pomocí metody `insertImage()` v případě PDF, nebo pomocí metody `createImage()` v případě SVG, jak tomu bylo u pozadí tvořeného obrázkem.

Kapitola 8

Testování

V této kapitole bude popsán průběh testování celé knihovny pro generování vektorového grafického výstupu z HTML renderovacího stroje CSSBox. Nejdříve budou popsány možnosti, které byly zvažovány pro testování. Dále bude popsáno testování na jednotlivých grafických elementech a poté na celých reálných webových stránkách. Na konci této kapitoly budou zhodnoceny výsledky a funkčnost implementované knihovny pro projekt CSSBox.

Při testování bylo zvažováno mezi automatizovaným a manuálním testováním. Automatické testování by se dalo provádět tak, že by se nejdříve vygeneroval vektorový výstup, který by se pixel po pixelu porovnal s původní HTML stránkou. Toto by bylo možné pouze pro verzi SVG, ne však pro verzi PDF. Především z důvodu, že v PDF jsou všechny rozměry elementů přepočítávány koeficientem poměru šířky původní stránky a šířky stránky výstupního PDF souboru. Jako výhodnější se jevilo použít manuální testy, kde budou výstupy SVG a PDF srovnány vizuálně s renderovanou webovou stránkou zobrazenou v prohlížeči.

Pro testování existuje oficiální testovací sada[1] od konsorcia W3C. Použitelnost této testovací sady byla prozkoumána, ale ukázalo se, že tato testovací sada pro tento typ práce není vhodná. V této testovací sadě jsou především testy, které se nehodí pro testování tištěných dokumentů. Chybí zde zcela testy pro stránkování PDF s využitím vlastností *page-break* apod. V této testovací sadě se nachází například testy pro rámečky elementu, pozadí, transformace, obrázky a další. I tyto testy se ovšem nedají z větší části použít pro testování vektorových výstupů. Z velké části obsahují testy animace a skripty, které vektorový výstup nepodporuje. Například po kliknutí na rámeček elementu se změní barva jeho jedné strany a v tom případě daný test projde. Toto není ani v jednom z vektorových formátů podporováno. Bylo tedy nutné vytvořit svou vlastní testovací sadu, která by vyhovovala testování vektorových výstupů a obsahovala testy na všechny prvky a vlastnosti, které byly v rámci této práce naimplementovány.

Vytvořená testovací sada se nachází na přiloženém paměťovém médiu v příloze A. Přehled jednotlivých testů a jejich výsledky se nachází v příloze E. Každý test obsahuje adresář *In* a *Out*. Adresář *In* obsahuje zdrojovou HTML stránku, která se v každém testu renderuje. Případně může obsahovat i vygenerovaný výstup bez vlastností *page-break* pro lepší srovnání efektu dané vlastnosti. Adresář *Out* pak obsahuje vygenerovanou stránku ve formátu SVG a PDF. Přehled, popis a výsledky jednotlivých testů se nachází v tabulce v souboru *test-list.xlsx*.

[1] http://test.csswg.org/harness/

8.1 Testy na jednotlivých grafických elementech

Většina testů se věnuje testování pouze jedné konkrétní vlastnosti nebo funkce. Na dané stránce se nachází vždy jeden element nebo malé množství elementů, které jsou nutné k otestování dané vlastnosti nebo funkce. Do testování jsou zahrnuty i krajní případy, se kterými se však na reálných webových stránkách příliš často nesetkáme. Testování na jednotlivých prvcích probíhalo už během samotného vývoje. Po dokončení vývoje proběhly všechny testy znovu a jejich přehled lze vidět v souboru *test-list.xlsx*.

Testy probíhaly tak, že nejdříve došlo k vygenerování stránky v obou vektorových formátech. Následně proběhlo porovnání výstupu a zdrojové webové stránky. Původní stránka byla zobrazena v prohlížeči *Chrome*. Webový prohlížeč *Chrome* dovoluje zobrazit i soubory SVG. Tento výstup mohl být tak srovnán přímo v jednom prohlížeči. Pro zobrazení PDF výstupu byl použit *Adobe Acrobat Reader DC*. Je nutné přihlédnout k faktu, že velikost jednotlivých prvků nemusí být stejná jako na původní stránce, protože jsou jednotlivé prvky poměrově zmenšeny.

Obrázek 8.1: Porovnání vykreslení zaoblených rohů. Ukázka z prohlížeče Chrome se nachází vlevo a výstup PDF se nachází vpravo.

Obrázek 8.2: Porovnání vykreslení zaoblených rohů. Ukázka z prohlížeče Chrome se nachází vlevo a výstup SVG se nachází vpravo.

Nejdříve proběhlo testování pozadí, rámečků a zaoblených rohů. Tyto testy byly prováděny na prvcích současně, protože spolu úzce souvisí. Této problematice bylo věnováno hodně času ze začátku vývoje projektu, proto zde většina testů proběhla úspěšně. Bylo otestováno jednobarevné pozadí i pozadí pomocí obrázku, a to včetně ořezové cesty, která je potřebná při použití zaoblených rohů. Také bylo testováno velké množství krajních případů pro zaoblené rohy. Například přechody barev, případ nulové šířky jedné strany, případ rozdílné šířky sousedních stran nebo případ, kdy je rozdílná hodnota x a y pro daný rádius.

Většina těchto testů proběhla úspěšně. Rozdíl oproti porovnávanému prohlížeči *Chrome* je v přechodu barev uvnitř zakulaceného rohu, kde se nepatrně liší směr a pozice přímky, podle níž přechod barev probíhá. Obě dvě varianty však splňují specifikaci CSS3, proto tyto drobné odchylky v testech byly považovány za nepodstatné. Další odchylka vznikla při zaobleném rohu, pokud velikost *border-radius* přesáhla svou velikostí polovinu délky nej-

kratší strany elementu. V takovém případě dojde v prohlížeči *Chrome* k omezení velikosti *border-radius* na velikost, která odpovídá polovině délky nejkratší strany. Ve vygenerovaném výstupním souboru SVG a PDF k tomuto omezení nedojde a je dovoleno vykreslit i extrémnější zaoblené rohy. Tento případ byl ponechán na odpovědnosti autora dané HTML stránky, protože specifikace CSS3 chování prohlížeče v těchto extrémních případech již nespecifikuje.

V rámci dalších testů na jednotlivé grafické elementy byl otestován lineární a radiální gradient a transformace. Testy byly inspirovány z oficiálních stránek W3C, ze stránky W3schools[2] a také ze stránek MDN Web Docs od společnosti Mozilla[3]. Byla otestována různá nastavení pro dané vlastnosti. Také byly otestovány všechny implementované transformace v různých nastaveních.

Následně probíhalo testování stránkování. Toto se týkalo pouze vektorového výstupu ve formátu PDF. Nejdříve bylo otestováno klasické stránkování bez použití vlastností *page-break*. Následně byly otestovány všechny vlastnosti *page-break*, a to jak samotné, tak i jejich vzájemné kombinace, aby se předešlo problémům s jejich současnou aplikací na různé elementy v rámci jedné webové stránky. Celé stránkování bylo otestováno, jak na elementech typu *ElementBox*, tak na obrázcích i textu. Této problematice bylo věnováno v rámci vývoje značné úsilí, proto většina testů prošla bez problémů. Neprošel pouze jediný test. Pokud se obrázek nebo text nachází na předělu stránky, nesmí být rozdělen a je celý odsunutý na další stránku a o stejnou vzdálenost je posunut i veškerý obsah, který se nachází pod ním. Tento test tak nebylo možné uskutečnit kvůli chybě, která není způsobena touto knihovnou, ale nástrojem CSSBox, který ignoruje vlastnost obrázku *margin-top*. I přes nastavení této vlastnosti prvku má daný element při generování hodnotu této vlastnosti 0. Nebylo tudíž možné posunout prvek na místo, kde by mohla být tato funkčnost posunutí obrázku na následující stranu ověřena. Nicméně, pokud se nachází obrázek na předělu stránky v rámci kontextu celé stránky při použití více elementů, tak je úspěšně posunut na další stránku. Z tohoto lze usoudit, že při opravě této chyby v projektu CSSBox, by měl tento test úspěšně projít.

Další testy byly na filtry, které lze aplikovat na obrázek. Zde vše proběhlo v pořádku. Jediná odchylka byla u filtru *grayscale*, který dává obrázek do tónu šedi. Pro černobílý obrázek problém nenastane. Pokud je ale například tato hodnota nastavena na 50 %, liší se obrázek na stránce v prohlížeči Chrome a vektorovém formátu.

Posledním typem testů na jednotlivých grafických elementech bylo testování seznamů. Konkrétně šlo o testování odrážek pro seznamy. Byly testovány všechny typy odrážek, od jednoduchých geometrických útvarů, až po odrážky pomocí obrázku.

8.2 Testování na reálných webových stránkách

První stránka, na které byly testy provedeny, byla vlastní webová stránka. Ta byla vytvořena a používána už během vývoje. Tato stránka obsahuje většinu z výše zmíněných vlastností a funkcí, které zde jsou použity současně.

Další testování již probíhalo na reálných stránkách, jako například *www.super.cz*, *www.kytary.cz*, *www.jakpsatweb.cz* a další. Při testování byl porovnáván pouze grafický výstup vektorových formátů s vstupním HTML dokumentem. V tomto případě nastal největší problém s nástrojem CSSBox. Důvodem je, že CSSBox nepodporuje některé složitější CSS vlastnosti. Další problémy byly způsobeny technologickými překážkami, a to konkrétně

[2] https://www.w3schools.com/css/

[3] https://developer.mozilla.org/en-US/docs/Web/CSS

těmi, že CSSBox neumí zpracovat JavaScript, flashové animace nebo reklamy. Některé typy stránek, které tyto prvky obsahují ve velkém množství, není možné vygenerovat. Z tohoto důvodu byly pro testování vybrány stránky, které výše zmíněné prvky neobsahují a pokud ano, tak pouze v omezené míře. Některé chyby v rozvržení reálných stránek ve výstupních vektorových formátech jsou tak způsobeny těmito problémy. Až na některé drobné odchylky lze konstatovat, že vygenerovaný výstup z větší části odpovídá vstupnímu HTML dokumentu. Ukázky generování reálných webových stránek jsou k dispozici v příloze C.

8.3 Závěr testování

Konečné testování na jednotlivých elementech neodhalilo žádné zásadní chyby. Ty, které byly v procesu testování odhaleny, byly neprodleně odstraněny. Převážně se jednalo pouze o nepodstatné detaily. Kladný výsledek testů byl dosažen především tím, že testování probíhalo po celou dobu vývoje a většina chyb tak byla již odstraněna v rámci procesu implementace. Při renderování webových stránek byly vybrány především stránky, které lze pomocí nástroje CSS v rozumné míře zobrazit. Na těchto stránkách proběhlo testování generování vektorového výstupu úspěšně. Z procesu testování a generování reálných stránek lze dojít k závěru, že tento projekt splnil účel, pro který byl navržen. Pokud by došlo k rozšíření CSSBox o možnost zpracování i technologicky složitějších stránek, bylo by možné docílit ještě lepších výsledků generování vektorového výstupu.

Kapitola 9

Závěr

Cílem této práce bylo navrhnout a implementovat novou společnou strukturu pro knihovny vykreslující vektorový výstup ve formátu SVG a PDF z renderovacího stroje CSSBox. Na začátku práce byly prostudovány použité technologie jako vektorová grafika, SVG a PDF formát, CSS3, nástroj CSSBox, knihovny pro vykreslování vektorových výstupů v jazyce Java a další. Bylo nezbytné prostudovat knihovnu CSSBox. Protože je knihovna CSSBox velmi rozsáhlá a souvisí i s jinými knihovnami, byl vynaložen značný čas k pochopení detailů fungování této knihovny. Dále byly prozkoumány dvě již existující řešení vykreslení vektorového výstupu. Tato dvě řešení byla, co se týkalo struktury kódu, na poměrně nízké úrovni a navíc některé funkce byly chybně implementovány, nebo nebyly implementovány vůbec. Postup a princip vykreslování vektorové grafiky se od sebe v obou pracích výrazně lišil, a proto byl vytvořen návrh, který měl za cíl tyto dvě řešení spojit do jedné společné struktury. Cílem nového návrhu je, aby převážná část implementace a vnitřní reprezentace objektů mohla být společná pro obě dvě řešení a aby se tato dvě řešení lišila pouze ve vykreslování elementů do daného vektorového formátu. Byly tedy navrženy úpravy obou řešení tak, aby se dala spojit. Následně byla obě dvě řešení prostudována a testována, přičemž se ukázalo, že některé prvky vektorové grafiky nejsou vykreslovány v souladu se specifikací CSS3, nebo jejich implementace zcela chybí. Tyto části byly sepsány a byl navržen postup opravy do další fáze.

V další fázi práce došlo k úpravě struktury, změně implementace obou částí, aby tvořily společný základ, a k opravě všech chyb, které byly během procesu vývoje a testování odhaleny. Některé funkce musely být nově vytvořeny, jiné byly upraveny nebo opraveny a některé části minulých prací bylo možné použít téměř beze změny, pouze v rámci nové struktury. Byla vytvořena nová struktura s abstraktními třídami, které sjednocují postup nebo reprezentaci pro generování obou dvou vektorových formátů. Mezi prvními byl sjednocen postup pro generování pozadí elementů a rámečky elementů včetně zaoblených rohů. Značná pozornost byla věnována verzi PDF, která v předešlé verzi neodpovídala specifikaci CSS3 pro rámečky a zaoblené rohy. Postup a výpočet generování zaoblených rohů byl pro verzi PDF vytvořen nově. Následně bylo doplněno generování textů. Zde nebylo možné, na rozdíl od pozadí nebo rámečků, sjednotit strukturu v takové míře, jelikož postup generování textů v SVG a PDF se značně liší. Dále byla sjednocena struktura pro seznamy. Podstatná část této práce je věnována problematice stránkování PDF s ohledem na vlastnosti *page-break*. V této oblasti došlo k významným změnám a opravám z důvodu nefunkční předchozí verze. Nakonec byl sjednocen postup pro generování gradientů, transformací a filtrů na obrázky. Všechny tyto postupy byly do velké míry sjednoceny. Především se podařilo

sjednotit postup aplikace filtrů na obrázky, který je téměř totožný pro oba dva vektorové formáty. U těchto vlastností došlo ke značné úpravě principů generování ve verzi PDF.

Všechny výše zmíněné funkce nové knihovny pro generování vektorového výstupu byly následně otestovány. Ověření funkčnosti proběhlo pomocí vlastních testů. Nejprve byly otestovány zvlášť jednotlivé prvky na samostatných elementech. Následně byly otestovány kombinace všech funkcí. Na závěr testovacího procesu probíhalo testování na reálných webových stránkách. Většina testů měla úspěšný výsledek zejména z důvodu, že testování probíhalo po celou dobu fáze implementace.

Na tuto práci by bylo možné navázat rozšířením nástroje CSSBox o zpracování JavaScriptu. Tím by generování vektorového výstupu pokrylo větší množství reálných webových stránek. Dalším rozšířením by mohla být implementace více vlastností filter pro obrázky, přidání generování animací v SVG nebo možnost generovat rámečky s vlastností *border-style*, která umožňuje vykreslovat například tečkované rámečky.

Literatura

[1] Adobe. *PDF Reference sixth edition* [online]. Listopad 2006 [cit. 2020-11-3]. Dostupné z: https://www.adobe.com/content/dam/acom/en/devnet/acrobat/pdfs/pdf_reference_1-7.pdf.

[2] Adobe. *Document management — Portable document format — Part 1: PDF 1.7*. Adobe Systems Incorporated, červenec 2008.

[3] Brinza, B., Bellamy Royds, A., Willigers, E., Storey, D., Schulze, D. et al. *Scalable Vector Graphics (SVG) 2, Chapter 9: Paths*. W3C, říjen 2018. Https://www.w3.org/TR/2018/CR-SVG2-20181004/paths.html.

[4] Burget, R. *CSSBox Manual* [online]. [cit. 21.11.2020]. Dostupné z: http://cssbox.sourceforge.net/manual/.

[5] Christe, C. *What Is the Difference Between Vector and Raster Graphics?* [online]. deverdesigns.com, únor 2017 [cit. 2020-10-20]. Dostupné z: https://www.deverdesigns.com/what-is-the-difference-between-vector-and-raster-graphics/.

[6] Connection, T. P. *Raster Images vs. Vector Graphics* [online]. Prosinec 2013 [cit. 2020-10-20]. Dostupné z: https://www.printcnx.com/resources-and-support/addiational-resources/raster-images-vs-vector-graphics/.

[7] Duong, N. H. *Transformace webových stránek do vektorové grafiky* [online]. Brno, 2019. [cit. 2020-10-20]. Bakalářská práce. Vysoké učení technické v Brně. Fakulta informačních technologií. Ústav informačních systémů. Dostupné z: http://hdl.handle.net/11012/180585.

[8] Eisenberg, J. D. a Bellamy Royds, A. *SVG Essentials*. 2. vyd. O'Reilly Media, 2014. ISBN 978-1-449-37435-8.

[9] Etemad, E. *CSS Box Model Module Level 3*. Candidate Recommendation. W3C, prosinec 2020 [cit. 20.11.2020]. Https://www.w3.org/TR/2020/CR-css-box-3-20201222/.

[10] Etemad, E., Kemper, B. a Bos, B. *CSS Backgrounds and Borders Module Level 3*. Candidate Recommendation. W3C, prosinec 2020 [cit. 18.11.2020]. Https://www.w3.org/TR/2020/CR-css-backgrounds-3-20201222/.

[11] Lehmkühler, A. *Apache PDFBox® - A Java PDF Library* [online]. [cit. 3.12.2020]. Dostupné z: https://pdfbox.apache.org/.

[12] McFarland, D. S. *CSS the missing manual.* 2. vyd. O'Reilly Media, 2009. ISBN 978-0-596-80244-8.

[13] Meyer, A. E. a Weyl, E. *CSS The Definitive Guide.* 4. vyd. O'Reilly Media, 2018. ISBN 978-1-449-39319-9.

[14] Press, O. *Vector & Raster Graphics In Offset Printing* [online]. Prosinec 2013 [cit. 2020-10-20]. Dostupné z: https://olypress.com/vector-vs-raster-graphics-in-printing/.

[15] The Apache Software Foundation. *The Apache XML Graphics Project* [online]. 2016 [cit. 20.11.2020]. Dostupné z: https://xmlgraphics.apache.org/.

[16] Vreedi. *What it is SVG format and what are the advantages ?* [online]. Říjen 2020 [cit. 2020-10-26]. Dostupné z: https://vreedi.medium.com/what-it-is-svg-format-and-what-are-the-advantages-1f0712ca926a.

[17] W3C. *CSS 2 Editor's Draft* [online]. Leden 2021 [cit. 20.3.2021]. Dostupné z: https://drafts.csswg.org/css2/#the-page.

[18] Červinka, Z. *Generování PDF dokumentů z webových stránek* [online]. Brno, 2015. [cit. 2020-10-20]. Bakalářská práce. Vysoké učení technické v Brně. Fakulta informačních technologií. Ústav informačních systémů. Dostupné z: http://hdl.handle.net/11012/52578.

[19] Šafář, M. *Transformace dokumentů HTML na vektorovou grafiku SVG* [online]. Brno, 2016. [cit. 2020-10-20]. Diplomová práce. Vysoké učení technické v Brně. Fakulta informačních technologií. Ústav informačních systémů. Dostupné z: http://hdl.handle.net/11012/61879.

Příloha A

Obsah CD

- *doc* - zdrojové kódy technické zprávy
- *CSSBox* - zdrojové kódy knihovny CSSBox včetně modifikovaných souborů
- *CSSBoxSvgPdf* - zdrojové kódy knihovny pro generování vektorového grafického výstupu z knihovny CSSBox
- *README* - informace o knihovně a manuál pro zprovoznění
- *xchoch07.pdf* - technická zpráva ve formátu pdf
- *tests* - vstupy a výstupy jednotlivých testů
- *test-list.xlsx* - seznam a hodnocení jednotlivých testů
- *RealWebPagesExamples* - ukázky generování reálných webových stránek, které se nacházejí i v přiložených testech

Příloha B

Manuál pro zprovoznění projektu

Tento manuál slouží pro zprovoznění projektu v prostředí IntelliJ IDEA, ve kterém byl celý projekt vyvíjen. Nejprve je nutné získat kódy knihovny CSSBox, jStyleParser a knihovny pro generování vektorového výstupu, která byla vytvořena v rámci tohoto projektu. Upravenou verzi CSSBoxu pro generování vektorového výstupu je možné stáhnout z následujícího odkazu:

`https://nextcloud.fit.vutbr.cz/s/pLWGDcBp7MqMYDW`

Ze serveru *github.com* lze naklonovat repozitář pro JStyleParser následujícím příkazem:

```
git clone https://github.com/radkovo/jStyleParser.git
```

Knihovnu pro generování vektorového výstupu lze stáhnout z následnujícího odkazu:

`https://nextcloud.fit.vutbr.cz/s/47oFK9Y9jFTobPi`

V IntelliJ se následně zvolí možnost pro importovaní projektu. Pro tento účel se použije soubor `pom.xml` knihovny CSSBox. Pomocí tohoto souboru lze projekt CSSBox naimportovat do prostředí IntelliJ. Nyní je nutné projekt CSSBox propojit s knihovnou JStyleParser a s knihovnou pro vektorový výstup. V pravé části vývojového prostředí se nachází ikona *Maven*, kde po rozbalení této nabídky zvolíme možnost *Add Maven Projects*. Pokud by se zde ikona nenacházela, zobrazíme ji pomocí nabídky *View -> Tool Windows -> Maven*. Poté naimportujeme pomocí Mavenu knihovnu JStyleParser s využitím souboru *pom.xml*. Nyní je projekt CSSBox funkční a je potřeba k němu připojit knihovnu pro generování vektorového výstupu. Připojení této knihovny k projektu CSSBox probíhá stejným způsobem jako v případě knihovny JStyleParser. Pomocí možnosti *Add Maven Projects* se vybere soubor *pom.xml* z knihovny CSSBoxSvgPdf. Na závěr je nutné kliknout na tlačítko *Reload All Maven Projects* a na tlačítko *Generate Sources and Update Folders For All Projects*.

Nyní již lze spustit projekt pomocí třídy `ImageRendererSvg` pro verzi SVG, anebo pomocí třídy `ImageRendererPdf` pro verzi PDF. Obě dvě třídy se nachází v knihovně pro vektorový výstup, konkrétně v adresáři `demo`.

Projekt pro verzi SVG je možné spustit například s následujícími parametry:

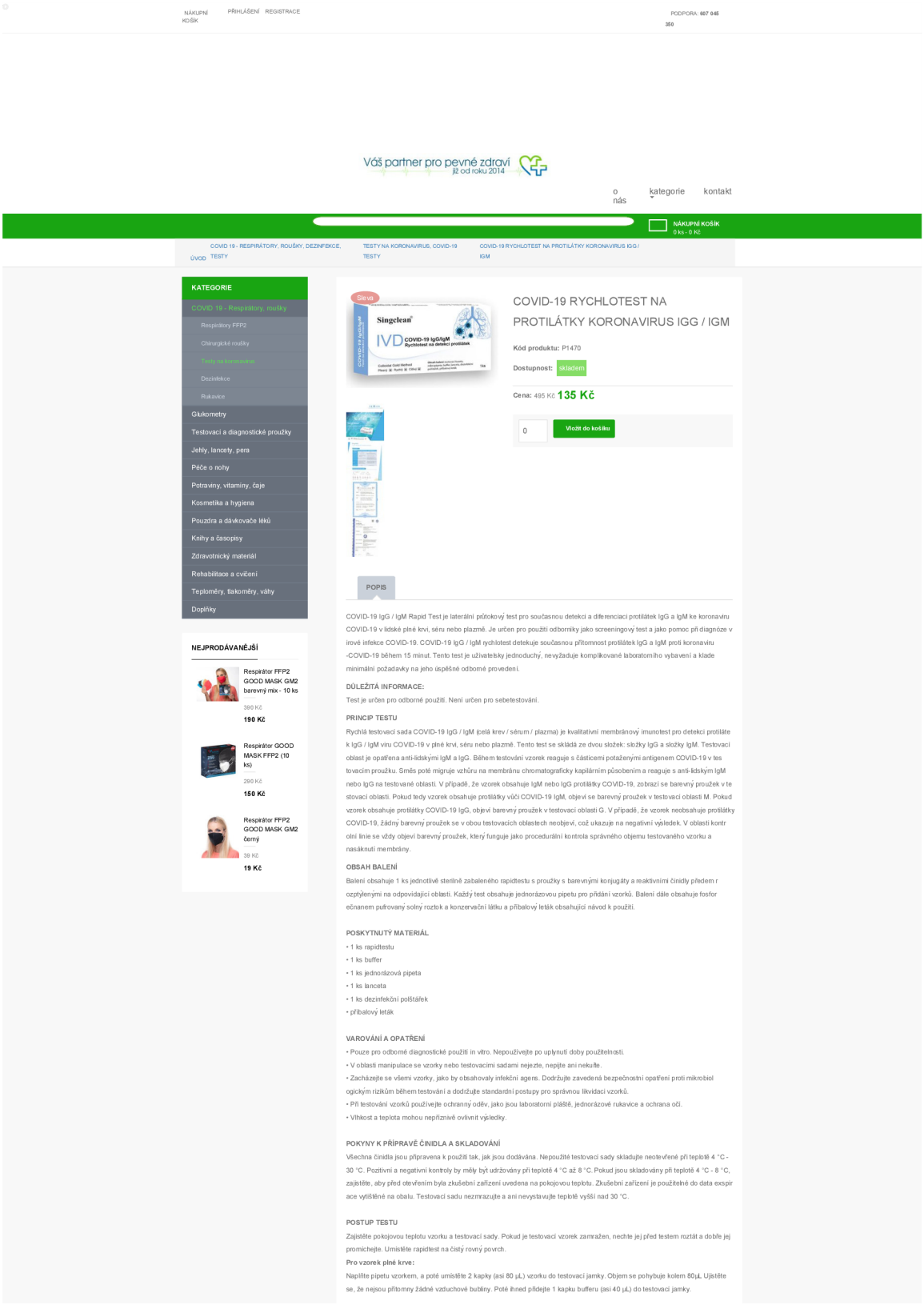
```
https://en.wikipedia.org/wiki/Brno_University_of_Technology out.svg
```

Projekt pro verzi PDF je možné spustit například s následujícími parametry:

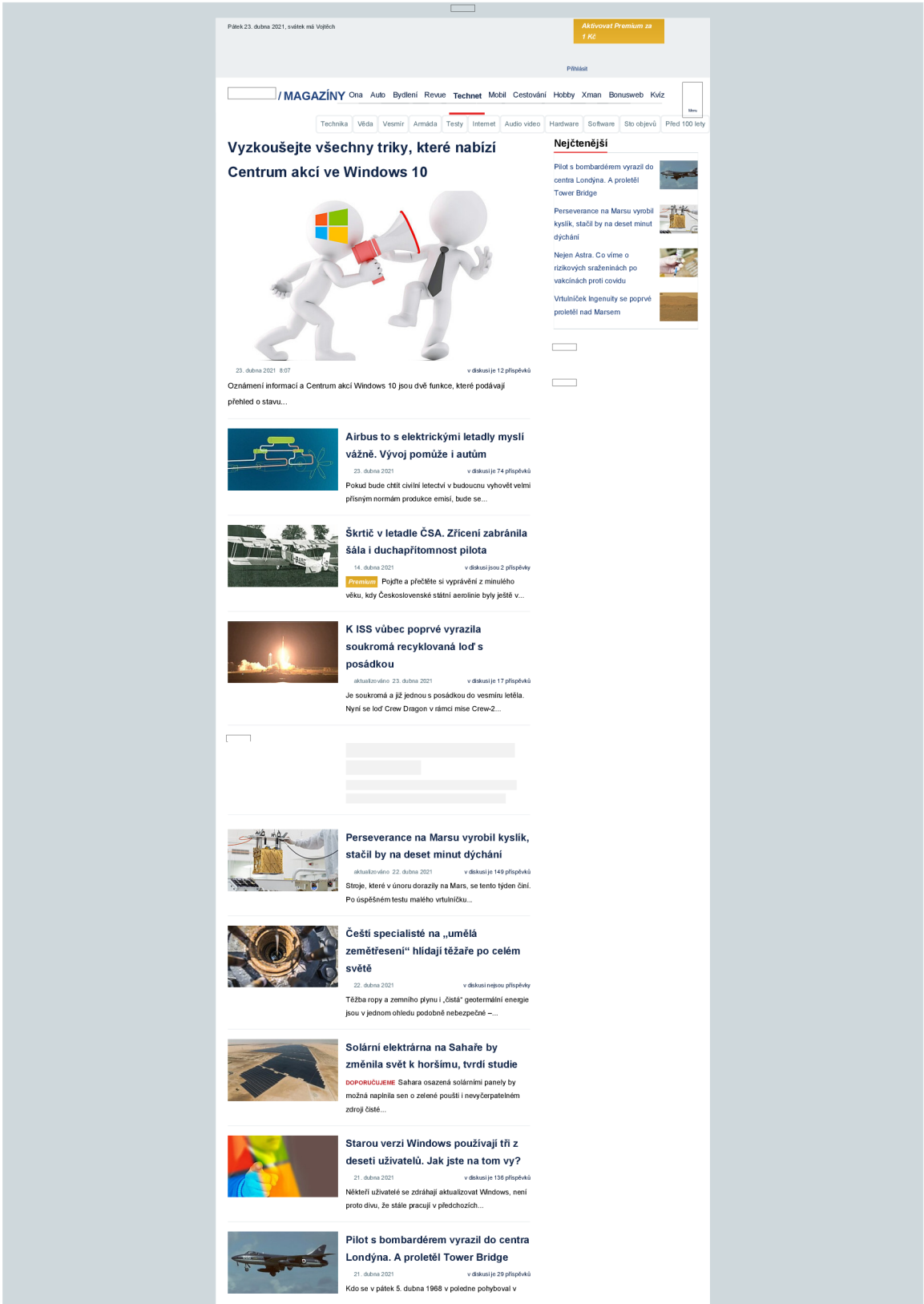
```
https://en.wikipedia.org/wiki/Brno_University_of_Technology out.pdf A4
```

Příloha C

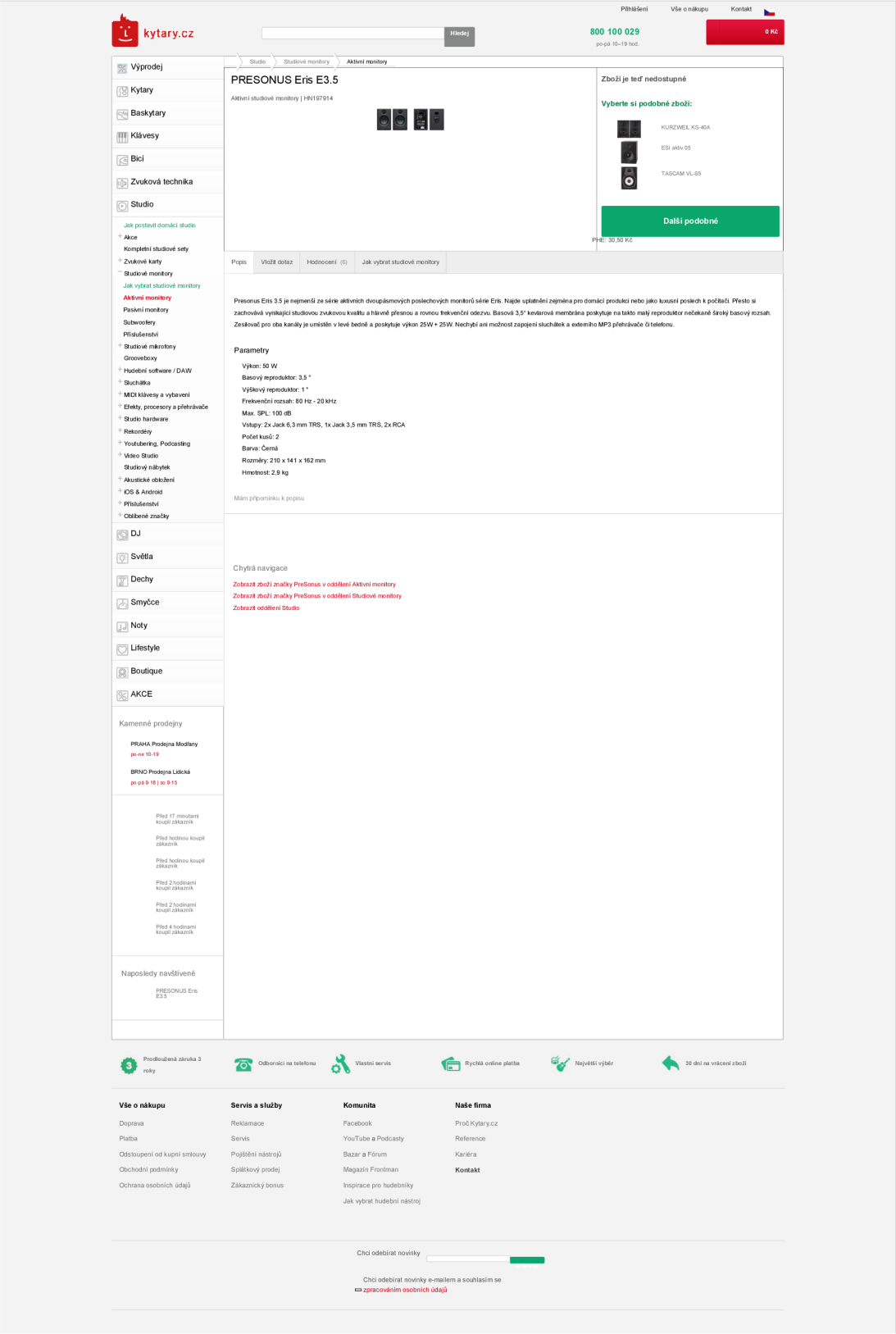
Ukázky generování reálných stránek



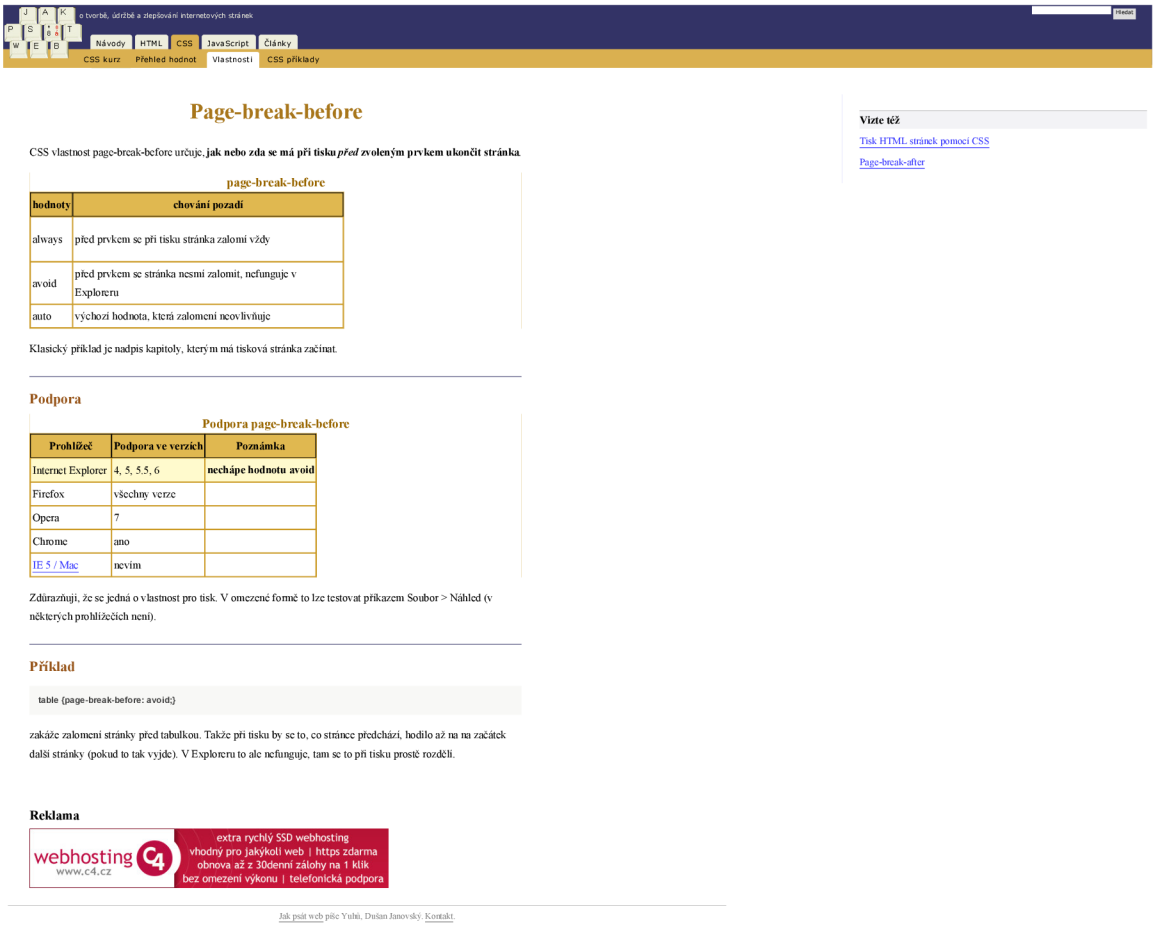
Obrázek C.1: Zdroj: https://www.dialekarna.cz/Respiratory-rousky/COVID-19-rychlotest-na-protilatky-koronavirus-IgG-IgM



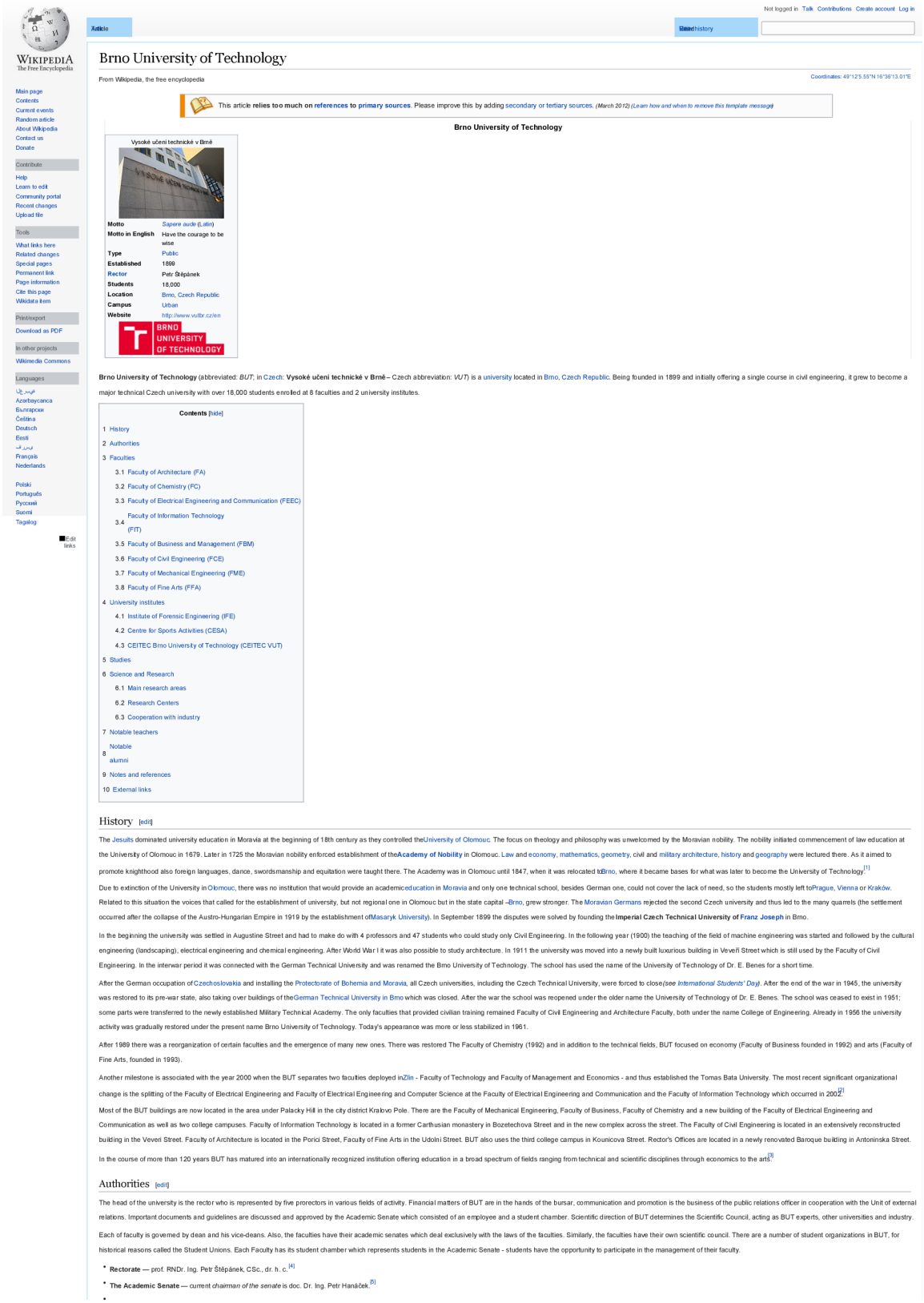
Obrázek C.2: Zdroj: https://www.idnes.cz/technet



Obrázek C.3: Zdroj: https://kytary.cz/presonus-eris-e3-5/HN197914/?gclid=CjwKCAjwlID8BRAFEiwAnUoK1SaChKlxQwUEyfpW9czM1tI3s56BzsLQed7Mkk_LKcvSbY73GyaqUBoClTMQAvD_BwE



Obrázek C.4: Zdroj: https://www.jakpsatweb.cz/css/page-break-before.html



Brno University of Technology

From Wikipedia, the free encyclopedia

This article **relies too much on references to primary sources**. Please improve this by adding secondary or tertiary sources. *(March 2012) (Learn how and when to remove this template message)*

Brno University of Technology

Vysoké učení technické v Brně

Motto	*Sapere aude* (Latin)
Motto in English	Have the courage to be wise
Type	Public
Established	1899
Rector	Petr Štěpánek
Students	18,000
Location	Brno, Czech Republic
Campus	Urban
Website	http://www.vutbr.cz/en

Brno University of Technology (abbreviated: *BUT*; in Czech: **Vysoké učení technické v Brně** – Czech abbreviation: *VUT*) is a university located in Brno, Czech Republic. Being founded in 1899 and initially offering a single course in civil engineering, it grew to become a major technical Czech university with over 18,000 students enrolled at 8 faculties and 2 university institutes.

Contents [hide]

History [edit]

The Jesuits dominated university education in Moravia at the beginning of 18th century as they controlled the University of Olomouc. The focus on theology and philosophy was unwelcomed by the Moravian nobility. The nobility initiated commencement of law education at the University of Olomouc in 1679. Later in 1725 the Moravian nobility enforced establishment of the **Academy of Nobility** in Olomouc. Law and economy, mathematics, geometry, civil and military architecture, history and geography were lectured there. As it aimed to promote knighthood also foreign languages, dance, swordsmanship and equitation were taught there. The Academy was in Olomouc until 1847, when it was relocated to Brno, where it became bases for what was later to become the University of Technology.[1]

Due to extinction of the University in Olomouc, there was no institution that would provide an academic education in Moravia and only one technical school, besides German one, could not cover the lack of need, so the students mostly left to Prague, Vienna or Kraków. Related to this situation the voices that called for the establishment of university, but not regional one in Olomouc but in the state capital – Brno, grew stronger. The Moravian Germans rejected the second Czech university and thus led to the many quarrels (the settlement occurred after the collapse of the Austro-Hungarian Empire in 1919 by the establishment of Masaryk University). In September 1899 the disputes were solved by founding the **Imperial Czech Technical University of Franz Joseph** in Brno.

In the beginning the university was settled in Augustine Street and had to make do with 4 professors and 47 students who could study only Civil Engineering. In the following year (1900) the teaching of the field of machine engineering was started and followed by the cultural engineering (landscaping), electrical engineering and chemical engineering. After World War I it was also possible to study architecture. In 1911 the university was moved into a newly built luxurious building in Veveří Street which is still used by the Faculty of Civil Engineering. In the interwar period it was connected with the German Technical University and was renamed the Brno University of Technology. The school has used the name of the University of Technology of Dr. E. Benes for a short time.

After the German occupation of Czechoslovakia and installing the Protectorate of Bohemia and Moravia, all Czech universities, including the Czech Technical University, were forced to close (see *International Students' Day*). After the end of the war in 1945, the university was restored to its pre-war state, also taking over buildings of the German Technical University in Brno which was closed. After the war the school was reopened under the older name the University of Technology of Dr. E. Benes. The school was ceased to exist in 1951; some parts were transferred to the newly established Military Technical Academy. The only faculties that provided civilian training remained Faculty of Civil Engineering and Architecture Faculty, both under the name College of Engineering. Already in 1956 the university activity was gradually restored under the present name Brno University of Technology. Today's appearance was more or less stabilized in 1961.

After 1989 there was a reorganization of certain faculties and the emergence of many new ones. There was restored The Faculty of Chemistry (1992) and in addition to the technical fields, BUT focused on economy (Faculty of Business founded in 1992) and arts (Faculty of Fine Arts, founded in 1993).

Another milestone is associated with the year 2000 when the BUT separates two faculties deployed in Zlín - Faculty of Technology and Faculty of Management and Economics - and thus established the Tomas Bata University. The most recent significant organizational change is the splitting of the Faculty of Electrical Engineering and Faculty of Electrical Engineering and Computer Science at the Faculty of Electrical Engineering and Communication and the Faculty of Information Technology which occurred in 2002.[2]

Most of the BUT buildings are now located in the area under Palacky Hill in the city district Kralovo Pole. There are the Faculty of Mechanical Engineering, Faculty of Business, Faculty of Chemistry and a new building of the Faculty of Electrical Engineering and Communication as well as two college campuses. Faculty of Information Technology is located in a former Carthusian monastery in Bozetechova Street and in the new complex across the street. The Faculty of Civil Engineering is located in an extensively reconstructed building in the Veveri Street. Faculty of Architecture is located in the Porici Street, Faculty of Fine Arts in the Udolni Street. BUT also uses the third college campus in Kounicova Street. Rector's Offices are located in a newly renovated Baroque building in Antoninska Street.

In the course of more than 120 years BUT has matured into an internationally recognized institution offering education in a broad spectrum of fields ranging from technical and scientific disciplines through economics to the arts.[3]

Authorities [edit]

The head of the university is the rector who is represented by five prorectors in various fields of activity. Financial matters of BUT are in the hands of the bursar, communication and promotion is the business of the public relations officer in cooperation with the Unit of external relations. Important documents and guidelines are discussed and approved by the Academic Senate which consisted of an employee and a student chamber. Scientific direction of BUT determines the Scientific Council, acting as BUT experts, other universities and industry.

Each of faculty is governed by dean and his vice-deans. Also, the faculties have their academic senates which deal exclusively with the laws of the faculties. Similarly, the faculties have their own scientific council. There are a number of student organizations in BUT, for historical reasons called the Student Unions. Each Faculty has its student chamber which represents students in the Academic Senate - students have the opportunity to participate in the management of their faculty.

- **Rectorate —** prof. RNDr. Ing. Petr Štěpánek, CSc., dr. h. c.[4]
- **The Academic Senate —** current *chairman of the senate* is doc. Dr. Ing. Petr Hanáček.[5]

Obrázek C.5: Zdroj: https://en.wikipedia.org/wiki/Brno_University_of_Technology

investujeme.cz

To se mi líbí | Tweet

investujeme.cz se líbí 1316 lidem. | investujeme.cz má 4214 sledujících.

Pátek 23. dubna. Svátek má Vojtěch.

INVESTICE BANKY PENZE BYDLENÍ POJIŠTĚNÍ FINANČNÍ PORADENSTVÍ BLOGY

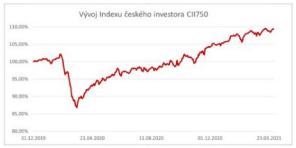
ČLÁNKY

Michal Kovářík: Pravidelná investice do každé rodiny, budu to hájit klidně celé hodiny

AKCIOVÉ TRHY | FINANČNÍ PORADENSTVÍ | INVESTICE | VÝVOJ EKONOMIKY | redakce | 23.04.2021 | 00:19

1 Komentář

Foto: Shutterstock Nedávno jsem vedl rozhovor s mladým párem kolem 30 let o pravidelném investování. Debata byla celkem jednoduchá, pár mi tvrdil, že investování nemá vůbec cenu a vesele hromadil veškeré své přebytky na bankovních účtech a stavebních spořeních. Rád bych se o toto téma podělil, případně vedl diskuzi, proč si myslím, že se to […]



Průměrný český investor dosáhl za první čtvrtletí zhodnocení 3,38 %

AKCIOVÉ TRHY | INVESTICE | Milan Vodička | 22.04.2021 | 14:12

0 Komentářů

Zdroj: Swiss Life Select a Thomson Reuters Loni se podílové fondy v České republice dokázaly poměrně rychle zotavit z jarních propadů. Nakonec dosáhly podle Indexu českého investora CII750 na průměrné zhodnocení 5,86 %. Do roku 2021 fondy vstoupily také s velmi slušnou výkonností. Za první čtvrtletí zaznamenal index průměrné zhodnocení 3,38 %.

Koronavirus zasáhl peněženky, někomu se peníze hromadí, jiní sahají do úspor

FINANČNÍ PORADENSTVÍ | INVESTICE | VÝVOJ EKONOMIKY | Jana Poncarová | 22.04.2021 | 00:40

0 Komentářů

Foto: Shutterstock Omezení výdajů, hromadění peněz na

DOPORUČUJEME

Spočítejte si

Filip Duchoň a Karel Šulc: Bankovní data zkvalitní finanční poradenství

Jak je drahá investice přes IŽP? Prozradí kalkulačka

Fincentrum Hypoindex: rekordní prosinec a rekordní celý rok 2020

KAM INVESTOVAT?

Wall Street se rýsuje závěr týdne v růstovém stylu

Frankfurtská burza uzavřela v záporných číslech

USA: Prodeje nových domů v březnu meziměsíčně vzrostly o 20,7 % při očekávaném růstu 14,2 %

Praha v závěru týdne ztrácela, ČEZ si musí na 600 Kč zatím počkat

Wall Street v úvodu obchodování umazává včerejší ztrátu

NEJNOVĚJŠÍ KOMENTÁŘE

LK: Michal Kovářík: Pravidelná investice do každé rodiny, budu to hájit klidně celé hodiny

Dobrý den, u uvedeného příkladu se jedná o investici cca 7% příjmů. Z článku není zřejmé, jaký investiční produkt je zvolený. U většiny investic do podílových fondů není připisováno zhodnocení průběžně, ale v hodnotě podílových listů. Složené úrokování není v …

HorovaL: Online platby se mohou od roku 2021 změnit

Děkuji za článek. V tomto článku je popsána aktuální situace - https://www.vimvic.cz/clanek/platby-kartou-jak-na-pojisteni-karty-a-kdo-muze-zneuzit-cislo-platebni-karty-pri-platbe-internetu .

Anna: Jak se daní

Zdroj a více informací: CII750.cz

KRÁTKÉ ZPRÁVY

Přes hrozbu sankcí ruská centrální banka nezapomíná na inflaci

0 Komentářů

Geopolitické otázky a sankce dominují výkonnosti ruských akcií, dluhopisů i měny a představují zásadní výzvu pro ruskou centrální banku a vládu, které se drží Putinovy konzervativní strategie s důrazem na stabilitu a soběstačnost.

Celá zpráva →

23.04.2021 | 17:02

Dluh zemí EU se přiblížil hranici 100 % HDP. ECB ve svých nákupech zatím polevovat nehodlá

0 Komentářů

ECB podle očekávání ani na dubnovém zasedání nezařadila nižší rychlost v nákupech dluhopisů a hodlá tak pokračovat v tisku eur i nadále. Jen v případě nouzového pandemického programu PEPP s rámcem 1,9 bilionů hodlá pokračovat až do března příštího roku. Od loňského začátku až do poloviny letošního dubna jen v rámci tohoto nástroje již nakoupila […]

Celá zpráva →

23.04.2021 | 10:38

Češi opravují domy i byty, zájem o úvěry na rekonstrukce narůstá

0 Komentářů

Píšeme na Hypoindex.cz: Kdo si nemůže kvůli vyšším cenám nemovitostí pořídit nový dům nebo byt, rekonstruuje. Když na renovace nestačí úspory, neváhají Češi požádat o půjčky. Banky a spořitelny potvrzují, že zájem o úvěry na rekonstrukce v poslední době roste, a to i kvůli práci a výuce z domova.

Celá zpráva →

23.04.2021 | 07:09

Pátý týden dovolené by firmy mohl stát desítky miliard korun

0 Komentářů

Mezi rusko-českou diplomatickou přestřelku a pandemii COVID 19 se v posledních dnech vklínilo ještě jedno téma – požadavek KSČM na povinných pět týdnů dovolené. Česko je v tuto chvíli v evropském pelotonu z pohledu délky povinné dovolené někde uprostřed, pokud tedy započítáme i státní svátky.

Celá zpráva →

22.04.2021 | 17:09

Analýzy ze Spořky: Nejlip přežívají koronakrizi důchodci

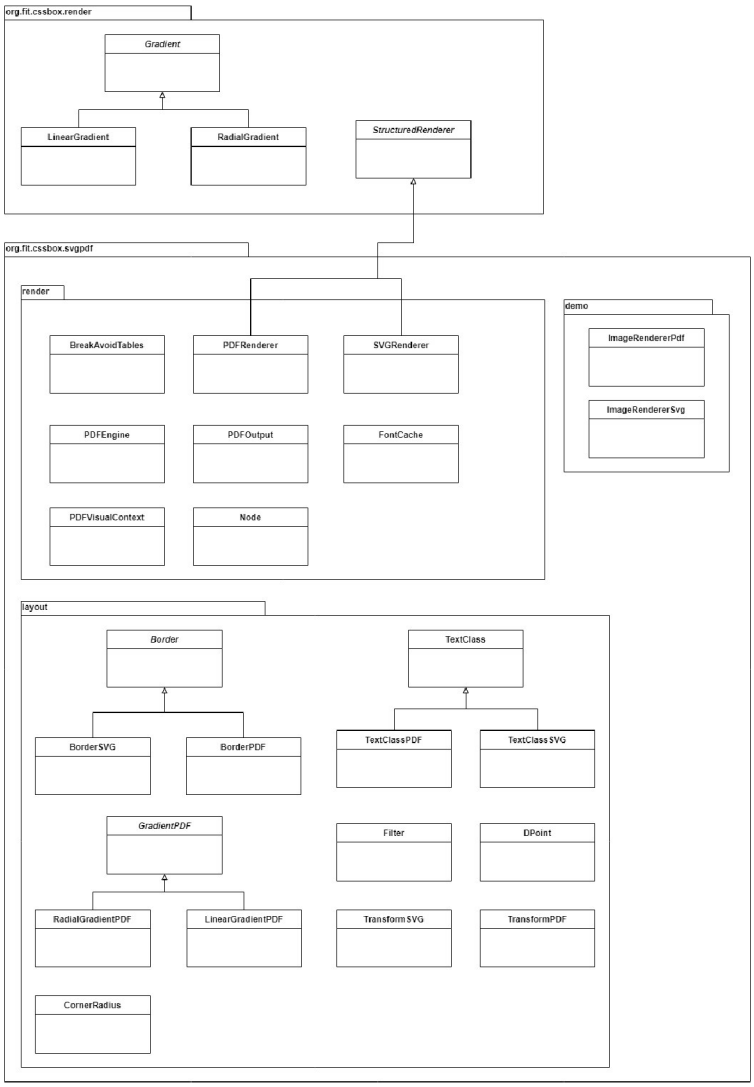
0 Komentářů

Nedávno zveřejněná data Evropské komise zpochybňují tezi české vlády, že v koronakrizi

Obrázek C.6: Zdroj: https://www.investujeme.cz/clanky/

Příloha D

Diagram tříd pro novou strukturu



Obrázek D.1: Nová struktura pro generování vektorového grafického výstupu z renderovacího stroje CSSBox.

Příloha E

Přehled testů

OK	Test prošel v pořádku a vektorový výstup odpovídá vstupní stránce
Wrong	Test neproše, vektorový výstup neodpovídá vstupní stránce
Nepodporuje	Daný vektorový formát danou vlastnost možnost neumožňuje zobrazit
OK	Test prošel s drobnými výhradami. Výstup nemusí úplně přesně odpovídat vstupní stránce
Wrong	Test neprošel, ale chyba není způsobena knihovnou pro vektorový výstup, ale ostatními moduly knihovny CSSBox

	Test	Popis textu	Stav testu PDF	Stav testu SVG	Poznámka
Testy na rámečky	1	Vykreslí obdélník s černým rámečkem bez zaoblených rohů a pozadí bude mít šedou barvu	OK	OK	
	2	Vykreslí obdelník s černým rámečkem bez zaoblených rohů, kde jako pozadí bude použit obrázek	OK	OK	
	3	Vykreslí obdélník s rámečkem bez zaoblených rohů, kde vertikální hrany budou červené a horizontální budou modré	OK	OK	
	4	Vykreslí se obdélník s černým rámečkem se zaoblenými rohy se šedým pozadím	OK	OK	
	5	Vykreslí se obdélník s černým rámečkem se zaoblenými rohy s obrázkem jako pozadí	OK	OK	
	6	Vykreslí se obdelník se zaoblenými rohy s přechodem barev mezi hranama	OK	OK	border-radius: 75px;
	7	Vykreslí se obdelník se zaoblenými rohy s přechodem barev mezi hranama	OK	OK	border-radius: 150px;
	8	Vykreslí se obdelník se zaoblenými rohy s přechodem barev mezi hranama	Wrong	Wrong	border-radius: 300px; extrémní případ, který některé prohlížeče už nevykreslí
	9	Vykreslí se obdelník se zaoblenými rohy s přechodem barev mezi hranama, kde šířky rámečku odpovídá zadanému radiusu	OK	OK	Speciální případ, který musí být v kódu osetren kvůli výpočtu a případnému dělaní nulou
	10	Vykreslí se obdelník se zaoblenými rohy s přechodem barev mezi hranama	OK	OK	border-radius: 40px; - border radius menší než šířka rámečku, vnitřní roh rámečku he ostrý
	11	Vykreslí se obdelník se zaoblenými rohy s přechodem barev mezi hranama	OK	OK	border-radius: 25px; - border radius menší než šířka rámečku, vnitřní roh rámečku he ostrý
	12	Vykreslí se obdelník se zaoblenými rohy s přechodem barev mezi hranama	OK	OK	border-radius: 5px; - border radius menší než šířka rámečku, vnitřní roh rámečku he ostrý
	13	Vykreslí se obdélník, kde vždy dva protiuhle rohy mají shodné radiusy rohů	OK	OK	border-radius: 100px 20px;
	14	Vykreslí se obdélník, kde vždy dva protiuhle rohy mají shodné radiusy rohů	OK	OK	border-radius: 200px 50px;
	15	Vykreslí se obdélník, kde každý roh má nastavený jiný border radius	OK	OK	border-radius: 50px 100px 150px 200px;
	16	Vykreslí se obdélník, kde jsou nastaveny rádiusy pro tři rohy	OK	OK	border-radius: 50px 100px 150px;
	17	Vykreslí se obdélník, kde je šířka horní strany rámečku 0	OK	OK	
	18	Vykreslí se obdélník, kde je šířka pravé strany rámečku 0	OK	OK	
	19	Vykreslí se obdélník, kde je šířka dolní strany rámečku 0	OK	OK	
	20	Vykreslí se obdélník, kde je šířka levé strany rámečku 0	OK	OK	
	21	Vykreslí se obdélník se zakulacenými rohy, kde je šířka horní strany rámečku 0	OK	OK	
	22	Vykreslí se obdélník se zakulacenými rohy, kde je šířka pravé strany rámečku 0	OK	OK	
	23	Vykreslí se obdélník se zakulacenými rohy, kde je šířka dolní strany rámečku 0	OK	OK	
	24	Vykreslí se obdélník se zakulacenými rohy, kde je šířka levé strany rámečku 0	OK	OK	
	25	Vykreslí se obdélník se zakulacenými rohy, kde je šířka levé a pravé strany rámečku 0	OK	OK	
	26	Vykreslí se obdélník se zakulacenými rohy, kde je šířka horní a dolní strany rámečku 0	OK	OK	
	27	Vykreslí se obdélník se zakulacenými rohy, kde je šířka horní a pravé strany rámečku 0	OK	OK	
	28	Vykreslí se obdélník se zakulacenými rohy, kde je šířka dolní a levé strany rámečku 0	OK	OK	
	29	Vykreslí se obdélník se zakulacenými rohy, kde tři strany mají šířku 0	OK	OK	
	30	Vykreslí obdélník se zakulacenými rohy kdy se šířka dvou sousedních stran liší 1	OK	OK	
	31	Vykreslí obdélník se zakulacenými rohy kdy se šířka dvou sousedních stran liší 2	OK	OK	
	32	Vykreslí obdélník se zakulacenými rohy kdy se šířka dvou sousedních stran liší 3	OK	OK	
	33	Vykreslí obdélník se zakulacenými rohy, kde je rozdílný horizontální a vertikální poměr pro zaoblený roh 1	OK	OK	75px 25px;
	34	Vykreslí obdélník se zakulacenými rohy, kde je rozdílný horizontální a vertikální poměr pro zaoblený roh 1	OK	OK	25px 75px;
	35	Vykreslí obdélník se zakulacenými rohy, kde je rozdílný horizontální a vertikální poměr pro zaoblený roh 1	OK	OK	50% 25%;
	36	Vykreslí obdélník se zakulacenými rohy, kde je rozdílný horizontální a vertikální poměr pro zaoblený roh 1	OK	OK	25% 50%;
	37	Vykreslí obdélník se zakulacenými rohy, kde je rozdílný horizontální a vertikální poměr pro zaoblený roh a každý roh je jiný	OK	OK	

Obrázek E.1: Přehled testů - část 1

	38	vykreslí obdélník s tloušťkou rámečku 5px a zaoblenými rohy	OK	OK	border-radius: 75px;
	39	vykreslí obdélník s tloušťkou rámečku 25px a zaoblenými rohy	OK	OK	border-radius: 75px;
	40	vykreslí obdélník s tloušťkou rámečku 50px a zaoblenými rohy	OK	OK	border-radius: 75px;
	41	vykreslí obdélník s tloušťkou rámečku 75px a zaoblenými rohy	OK	OK	border-radius: 75px;
	42	vykreslí obdélník s tloušťkou rámečku 100px a zaoblenými rohy	OK	OK	border-radius: 75px;
	43	vykreslí obdélník s tloušťkou rámečku 150px a zaoblenými rohy	OK	OK	border-radius: 75px;
	44	vykreslí obdélník s tloušťkou rámečku 200px a zaoblenými rohy	OK	OK	border-radius: 75px;
	45	vykreslí obdélník s tloušťkou rámečku 500px a zaoblenými rohy	OK	OK	border-radius: 75px;
	46	vykreslí obdélník s tloušťkou rámečku 500px a zaoblenými rohy border-radius: 500px	OK	OK	border-radius: 500px;
Lineární gradient	47	Vykreslí lineární gradient se třemi barvami směrem nahoru	OK	OK	linear-gradient(to top, red, yellow, blue);
	48	Vykreslí lineární gradient se třemi barvami směrem doprava	OK	OK	
	49	Vykreslí lineární gradient se třemi barvami směrem dolů	OK	OK	
	50	Vykreslí lineární gradient se třemi barvami směrem doleva	OK	OK	
	51	Vykreslí lineární gradient se čtyřmi barvami pod uhlem 45 stupňů	OK	OK	
	52	Vykreslí lineární gradient se čtyřmi barvami pod uhlem 135 stupňů	OK	OK	
	53	Vykreslí lineární gradient se čtyřmi barvami pod uhlem 225 stupňů	OK	OK	
	54	Vykreslí lineární gradient se čtyřmi barvami pod uhlem 315 stupňů	OK	OK	
	55	Vykreslí lineární gradient s velkým počtem barev	OK	OK	
Radiální gradient	56	Vykreslí radiální gradient se třema barvami	OK	OK	
	57	Vykreslí radiální gradient s rozdílnou velikostí obsastí pro jednotlivé barvy	OK	OK	
	58	Vykreslí radiální gradient se třemi barvami jako kruh	OK	OK	
	59	Vykreslí radiální gradient se třemi barvami jako elipsu u horního okraje	OK	OK	
	60	Vykreslí radiální gradient se třemi barvami jako elipsu u pravého okraje	OK	OK	
	61	Vykreslí radiální gradient se třemi barvami jako elipsu u dolního okraje	OK	OK	
	62	Vykreslí radiální gradient se třemi barvami jako elipsu u levého okraje	OK	OK	
	63	Vykreslí gradient se třemi barvami s vlastností closest-side	OK	OK	
	64	Vykreslí gradient se třemi barvami s vlastností farthest-side	OK	OK	
	65	Vykreslí gradient se třemi barvami s vlastností closest-corner	OK	OK	
	66	Vykreslí gradient se třemi barvami s vlastností farthest corner	OK	OK	
transformace	67	Vykreslí obdélník s rotací 45 stupňů	OK	OK	
	68	Vykreslí obdélník s rotací 90 stupňů	OK	OK	
	69	Vykreslí obdélník s rotací 135 stupňů	OK	OK	
	70	Vykreslí obdélník s rotací 180 stupňů	OK	OK	
	71	Vykreslí obdélník s rotací 235 stupňů	OK	OK	
	72	Vykreslí obdélník s rotací 270 stupňů	OK	OK	
	73	Vykreslí obdélník s rotací 315 stupňů	OK	OK	
	74	Vykreslí zkosený obdélník ve svislém směru pod úhlem -45 stupňů	OK	OK	
	75	Vykreslí zkosený obdélník ve svislém směru pod úhlem -35 stupňů	OK	OK	
	76	Vykreslí zkosený obdélník ve svislém směru pod úhlem -25 stupňů	OK	OK	
	77	Vykreslí zkosený obdélník ve svislém směru pod úhlem -15 stupňů	OK	OK	
	78	Vykreslí zkosený obdélník ve svislém směru pod úhlem -5 stupňů	OK	OK	
	79	Vykreslí zkosený obdélník ve svislém směru pod úhlem 5 stupňů	OK	OK	
	80	Vykreslí zkosený obdélník ve svislém směru pod úhlem 15 stupňů	OK	OK	
	81	Vykreslí zkosený obdélník ve svislém směru pod úhlem 25 stupňů	OK	OK	
	82	Vykreslí zkosený obdélník ve svislém směru pod úhlem 35 stupňů	OK	OK	
	83	Vykreslí zkosený obdélník ve svislém směru pod úhlem 45 stupňů	OK	OK	
	84	Vykreslí zkosený obdélník ve horizontálním směru pod úhlem -45 stupňů	OK	OK	
	85	Vykreslí zkosený obdélník ve horizontálním směru pod úhlem -35 stupňů	OK	OK	
	86	Vykreslí zkosený obdélník ve horizontálním směru pod úhlem -25 stupňů	OK	OK	
	87	Vykreslí zkosený obdélník ve horizontálním směru pod úhlem -15 stupňů	OK	OK	
	88	Vykreslí zkosený obdélník ve horizontálním směru pod úhlem -5 stupňů	OK	OK	
	89	Vykreslí zkosený obdélník ve horizontálním směru pod úhlem 5 stupňů	OK	OK	
	90	Vykreslí zkosený obdélník ve horizontálním směru pod úhlem 15 stupňů	OK	OK	
	91	Vykreslí zkosený obdélník ve horizontálním směru pod úhlem 25 stupňů	OK	OK	
	92	Vykreslí zkosený obdélník ve horizontálním směru pod úhlem 35 stupňů	OK	OK	
	93	Vykreslí zkosený obdélník ve horizontálním směru pod úhlem 45 stupňů	OK	OK	
	94	Vykreslí zkosený obdélník v horizontálním i vertikálním směru pod úhlem -25 a -25 stupňů	OK	OK	
	95	Vykreslí zkosený obdélník v horizontálním i vertikálním směru pod úhlem -15 a -15 stupnů	OK	OK	
	96	Vykreslí zkosený obdélník v horizontálním i vertikálním směru pod úhlem 15 a 15 stupnů	OK	OK	
	97	Vykreslí zkosený obdélník v horizontálním i vertikálním směru pod úhlem 25 a 25 stupňů	OK	OK	
	98	Vykreslí obdélník, který je posunut o -100px po ose x	OK	OK	
	99	Vykreslí obdélník, který je posunut o 100px po ose x	OK	OK	
	100	Vykreslí obdélník, který je posunut o 200px po ose x	OK	OK	
	101	Vykreslí obdélník, který je posunut o 300px po ose x	OK	OK	
	102	Vykreslí obdélník, který je posunut o -100px po ose y	OK	OK	
	103	Vykreslí obdélník, který je posunut o 200px po ose y	OK	OK	
	104	Vykreslí obdélník, který je posunut o 400px po ose y	OK	OK	
	105	Vykreslí obdélník, který je posunut o 600px po ose y	OK	OK	
	106	Vykreslí obdélní, který je posunut o -100 px na ose x a o -100px na ose y	OK	OK	

Obrázek E.2: Přehled testů - část 2

	107	Vykreslí obdélní, který je posunut o 100 px na ose x a o 200px na ose y	OK	OK	
	108	Vykreslí obdélní, který je posunut o 200 px na ose x a o 400px na ose y	OK	OK	
	109	Vykreslí obdélní, který je posunut o 300 px na ose x a o 600px na ose y	OK	OK	
	110	Změna měřítka pro osu x	OK	OK	transform: scaleX(0.2);
	111	Změna měřítka pro osu x	OK	OK	transform: scaleX(0.7);
	112	Změna měřítka pro osu x	OK	OK	transform: scaleX(1.5);
	113	Změna měřítka pro osu x	OK	OK	transform: scaleX(1.8);
	114	Změna měřítka pro osu y	OK	OK	transform: scaleY(0.2);
	115	Změna měřítka pro osu y	OK	OK	transform: scaleY(0.7);
	116	Změna měřítka pro osu y	OK	OK	transform: scaleY(1.5);
	117	Změna měřítka pro osu y	OK	OK	transform: scaleY(1.8);
	118	Změna měřítka pro osu x i pro osu y	OK	OK	transform: scale(0.2, 0.2);
	119	Změna měřítka pro osu x i pro osu y	OK	OK	transform: scale(1.8, 1.8);
	120	Vykreslí obrázek s rotací 45 stupňů	OK	OK	
	121	Vykreslí obrázek s rotací 135 stupňů	OK	OK	
	122	Vykreslí obrázek s rotací 235 stupňů	OK	OK	
	123	Vykreslí obrázek s rotací 315 stupňů	OK	OK	
	124	Vykreslí zkosený obrázek ve svislém směru pod úhlem -35 stupňů	OK	OK	
	125	Vykreslí zkosený obrázek ve svislém směru pod úhlem 35 stupňů	OK	OK	
	126	Vykreslí zkosený obrázek ve horizontálním směru pod úhlem -25 stupňů	OK	OK	
	127	Vykreslí zkosený obrázek ve horizontálním směru pod úhlem 25 stupňů	OK	OK	
	128	Vykreslí zkosený obrázek v horizontálním i vertikálním směru pod úhlem -15 a -15 stupnů	OK	OK	
	129	Vykreslí zkosený obrázek v horizontálním i vertikálním směru pod úhlem 25 a 25 stupňů	OK	OK	
	130	Vykreslí obrázek, který je posunut o -100px po ose x	OK	OK	
	131	Vykreslí obrázek, který je posunut o 300px po ose x	OK	OK	
	132	Vykreslí obrázek, který je posunut o -100px po ose y	OK	OK	
	133	Vykreslí obrázek, který je posunut o 400px po ose y	OK	OK	
	134	Vykreslí obrázek, který je posunut o 300 px na ose x a o 400px na ose y	OK	OK	
	135	Změna měřítka obrázku pro osu x	OK	OK	transform: scaleX(0.7);
	136	Změna měřítka obrázku pro osu x	OK	OK	transform: scaleX(1.5);
	137	Změna měřítka obrázku pro osu y	OK	OK	transform: scaleY(0.7);
	138	Změna měřítka obrázku pro osu y	OK	OK	transform: scaleY(1.5);
	139	Změna měřítka obrázku pro osu x i pro osu y	OK	OK	transform: scale(1.8, 1.8);
Stránkování	140	Zelený i modrý element bude vykreslen na první stránce	OK	Nepodporuje	
	141	Zelený i modrý element bude vykreslen na druhé stránce	OK	Nepodporuje	
	142	Zelený element bude vykreslen na první stránce a modrý element bude vykreslen na druhé stránce	OK	Nepodporuje	
	143	Zelený element bude na první stránce, modrý na druhé a červený na třetí	OK	Nepodporuje	
	144	Element bude vykreslen na úplném konci první strany	OK	Nepodporuje	
	145	Element bude vykreslen na úplném začátku druhé strany	OK	Nepodporuje	
	146	Zelený prvek má nastavenou vlastnost page-break-before-always	OK	Nepodporuje	Ve složce In se nachází výstup pdf bez použité vlastnosti page-break
	147	Hnedý prvek má nastavenou vlastnost page-break-after-always	OK	Nepodporuje	Ve složce In se nachází výstup pdf bez použité vlastnosti page-break
	148	Zelený prvek má nastavenou vlastnost page-break-after-avoid	OK	Nepodporuje	Ve složce In se nachází výstup pdf bez použité vlastnosti page-break
	149	Červený prvek má nastavenou vlastnost page-break-before-avoid	OK	Nepodporuje	Ve složce In se nachází výstup pdf bez použité vlastnosti page-break
	150	Zelený prvek má nastavenou vlastnost page-break-inside-avoid	OK	Nepodporuje	Ve složce In se nachází výstup pdf bez použité vlastnosti page-break
	151	Červený prvek má nastavenou vlastnost page-break-before-always	OK	Nepodporuje	Prvek se již nachází na začátku stránky, proto před ním přehod stránek vytvořen explicitně nebude
	152	Oranžový prvek má nastavenou vlastnost page-break-before-always - nová stránka bude až nad obrázkem	OK	Nepodporuje	Ve složce In se nachází výstup pdf bez použité vlastnosti page-break
	153	Vykreslí se jednobarevný element na předělu stránky	OK	Nepodporuje	
	154	Vykreslí se jednobarevný element s rámečkem na předělu stránky	OK	Nepodporuje	
	155	Vykreslí se jednobarevný element s rámečkem a vlastností border-raidus na předělu stránky	OK	Nepodporuje	
	156	Vykreslí se element s pozadím pomocí obrásku s rámečkem a vlastností border-radius na předělu stránky	OK	Nepodporuje	
	157	Vykreslí se obrázek na předělu stránky (bude celý posunut na další stránku)	Wrong	Nepodporuje	Chyba v CSSBoxu - nefunguje margin-top pro obrázky
Kombinace vlastností	158	Zenený prvek má vlastnost page-break-before-always a oranžový má vlastnost page-break-before-avoid	OK	Nepodporuje	Ve složce In se nachází výstup pdf bez použité vlastnosti page-break
	159	Zelený prvek má vlastnost page-break-before-always a modrý má vlastnost page-break-after-avoid	OK	Nepodporuje	Ve složce In se nachází výstup pdf bez použité vlastnosti page-break
	160	Hnědý prvek má vlastnost page-break-after-always a oranžový má vlastnost page-break-before-avoid	OK	Nepodporuje	Ve složce In se nachází výstup pdf bez použité vlastnosti page-break
	161	Hnědý prvek má vlastnost page-break-after-always a modrý má vlastnost page-break-after-avoid	OK	Nepodporuje	Ve složce In se nachází výstup pdf bez použité vlastnosti page-break
	162	Hnědý prvek má vlastnost page-break-after-always a modrý má vlastnost page-break-inside-avoid	OK	Nepodporuje	Ve složce In se nachází výstup pdf bez použité vlastnosti page-break
	163	Všechny prvky mají nastavenou vlastnost page-break-before-always	OK	Nepodporuje	Ve složce In se nachází výstup pdf bez použité vlastnosti page-break
	164	Všechny prvky mají nastavenou vlastnost page-break-after-always	OK	Nepodporuje	Ve složce In se nachází výstup pdf bez použité vlastnosti page-break
	165	Vlastnost page-break-before-always pro červený prvek, který už se nachází na začátku stránky (nevznikne volná stránka před tímto prvkem)	OK	Nepodporuje	Ve složce In se nachází výstup pdf bez použité vlastnosti page-break
	166	Vlastnost page-break-after-always pro zelený prvek, který už se nachází na konci stránky (nevznikne volná stránka za tímto prvkem)	OK	Nepodporuje	Ve složce In se nachází výstup pdf bez použité vlastnosti page-break
Filter	167	Vykreslí se obrázek s vlastností filter - opacity 0,25	OK	OK	
	168	Vykreslí se obrázek s vlastností filter - opacity 0,5	OK	OK	
	169	Vykreslí se obrázek s vlastností filter - opacity 0,75	OK	OK	
	170	Vykreslí se obrázek s vlastností filter - brightness 0,25	OK	OK	
	171	Vykreslí se obrázek s vlastností filter - brightness 0,5	OK	OK	
	172	Vykreslí se obrázek s vlastností filter - brightness 0,75	OK	OK	
	173	Vykreslí se obrázek s vlastností filter - brightness 1,25	OK	OK	
	174	Vykreslí se obrázek s vlastností filter - brightness 1,5	OK	OK	

Obrázek E.3: Přehled testů - část 3

	178	Vykreslí se obrázek s vlastností filter - grayscale 50%	Wrong	Wrong	
	179	Vykreslí se obrázek s vlastností filter - grayscale 100%	OK	OK	
Text	180	Vykreslí se text	OK	OK	
	181	Vykreslí se podtržený text	OK	OK	
	182	Vykreslí se text s hypertextovým odkazem	OK	Nepodporuje	
	183	Vykreslí se dva texty, které na sebe navazují	OK	OK	
	184	Vykreslí se dva texty pod sebou	OK	OK	
	185	Vykreslí se text na předělu stránek (text bude posunut na další stránku, aby se nenacházel na předělu stránek)	OK	OK	
	186	Vykreslí se text s explicitními mezerami mezi slovy 15px	OK	OK	
	187	Vykreslí se text s explicitními mezerami mezi slovy 30px	OK	OK	
	188	Vykreslí se text s explicitními mezerami mezi slovy 45px	OK	OK	
Seznamy	189	Vyskreslí se seznam s odrážkou jako plný kruh	OK	OK	
	190	Vykreslí se seznam s odrážkou jako prázdný kruh	OK	OK	
	191	Vykreslí se seznam s odrážkou jako čtverec	OK	OK	
	192	Vykreslí se seznam s odrážkou jako text (číslovaný seznam)	OK	OK	
	193	Vykreslí se seznam s odrážkou jako text (abeceda)	OK	OK	
	194	Vykreslí se seznam s odrážkou jako text (římské číslice)	OK	OK	
	195	Vykreslí se seznam s odrážkou jako obrázek	OK	OK	
Vlastní testovací strá	196	Kombinace většiny implementovaných vlastnost na jedné vlastí weobové stránce	OK	OK	Vlastní webová strána, na této stránce byli testovány jednotlivé prvky v průběhu vývoje
Reálné testovací strá	197	https://www.super.cz/746090-princ-harry-prekvapil-detaily-o-sve-rodine-co-bylo-archieho-prvni-slovo-a-jak-travi-cas-v-jejich-novem-domove.html	OK	OK	
	198	https://www.4home.cz/4home-prehoz-na-postel-circles-bezova/?utm_source=favi&utm_medium=cpc&utm_campaign=favi-prehoz-na-postel&utm_term=e82da38a-b449-	OK	OK	Drobné odchylky v pozicování jsou způsobeny tím, že CSSBox neumí zpracovat javascript apod.
	199	https://www.dialekarna.cz/Respiratory-rousky/COVID-19-rychlotest-na-protilatky-koronavirus-IgG-IgM	OK	OK	
	200	https://help.ubuntu.com/community/UEFI	OK	OK	
	201	https://kytary.cz/presonus-eris-e3-5/HN197914/?gclid=CjwKCAjwIlDBBRAFEiwAnUoK1SaChKlxQwUEyfpW9czM1tI3s56BzsLQed7Mkk_LKcvSbY73Gyaq	OK	OK	Drobné odchylky v pozicování jsou způsobeny tím, že CSSBox neumí zpracovat javascript apod.
	202	https://www.jakpsatweb.cz/css/page-break-before.html	OK	OK	
	203	http://www.fit.vutbr.cz/~martinek/latex/czechiso.h	OK	OK	
	204	https://www.idnes.cz/technet	OK	OK	
	205	https://en.wikipedia.org/wiki/Brno_University_of_Technology	OK	OK	
	206	https://www.investujeme.cz/clanky/	OK	OK	
Různé formáty výstu	207	Vlastní webová stránka A1	OK	Nepodporuje	
	208	Reálná webová stránka A1	OK	Nepodporuje	
	209	Vlastní webová stránka A2	OK	Nepodporuje	
	210	Reálná webová stránka A2	OK	Nepodporuje	
	211	Vlastní webová stránka A3	OK	Nepodporuje	
	212	Reálná webová stránka A3	OK	Nepodporuje	
	213	Vlastní webová stránka A4	OK	Nepodporuje	
	214	Reálná webová stránka A4	OK	Nepodporuje	
	215	Vlastní webová stránka A5	OK	Nepodporuje	
	216	Reálná webová stránka A5	OK	Nepodporuje	
	217	Vlastní webová stránka A6	OK	Nepodporuje	
	218	Reálná webová stránka A6	OK	Nepodporuje	
	219	Reálná webová stránka LETTER	OK	Nepodporuje	
	220	Vlastní webová stránka LETTER	OK	Nepodporuje	

Obrázek E.4: Přehled testů - část 4