VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

IMPLEMENTACE FLEXIBILNÍHO ROZLOŽENÍ VE ZOBRAZOVACÍM STROJI CSS

FLEXIBLE LAYOUT IMPLEMENTATION IN A CSS RENDERING ENGINE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **LUKÁŠ ONDRÁK**
AUTHOR

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

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Ondrák Lukáš**
Program: Informační technologie
Název: **Implementace flexibilního rozložení ve zobrazovacím stroji CSS**
Flexible Layout Implementation in a CSS Rendering Engine
Kategorie: Web

Zadání:

1. Seznamte se s projektem experimentálního zobrazovacího stroje CSSBox a jeho architekturou.
2. Prostudujte nové způsoby rozložení obsahu pomocí jazyka CSS jako např. grid layout a flexbox.
3. Navrhněte způsob rozšíření knihovny CSSBox o nové způsoby rozložení obsahu.
4. Po dohodě s vedoucím implementujte podporu rozložení "Flexbox" ve zobrazovacím stroji.
5. Proveďte testování vytvořeného rozšíření pomocí vhodné testovací sady.
6. Zhodnoťte dosažené výsledky.

Literatura:

- Dokumentace projektu CSSBox: http://cssbox.sourceforge.net/documentation.php
- Michálek, M.: Vzhůru do CSS3, e-kniha, 2015, https://www.vzhurudolu.cz/ebook-css3
- Andrew, R.: Get Ready for CSS Grid Layout, A Book Apart, 2016, https://abookapart.com/products/get-ready-for-css-grid-layout

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Burget Radek, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 16. října 2018

Abstrakt

Tato bakalářská práce se zabývá studiem architektury experimentálního zobrazovacího stroje CSSBox a jeho rozšířením o nový způsob rozložení webového obsahu s názvem flexibilní rozložení. Práce se podrobně věnuje popisu návrhu, implementačními detaily a také popisuje testování na referenční testovací sadě. Na závěr jsou zhodnoceny dosažené výsledky a nastíněn možný budoucí vývoj.

Abstract

This bachelor thesis is about studying architecture of experimental rendering engine CSSBox and his extension of a new way of layouting web content called flexible layout. It pays attention to designing, implementation details and testing using reference test set. In conclusion achieved results are evaluated and intimated possible future development.

Klíčová slova

flexibilní rozložení, CSSBox, Java, HTML, CSS, DOM

Keywords

flexible box layout, CSSBox, Java, HTML, CSS, DOM

Citace

ONDRÁK, Lukáš. *Implementace flexibilního rozložení ve zobrazovacím stroji CSS*. Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Radek Burget, Ph.D.

Implementace flexibilního rozložení ve zobrazovacím stroji CSS

Prohlášení

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

.....
Lukáš Ondrák
15. května 2019

Poděkování

Rád bych poděkoval vedoucímu mé bakalářské práce Ing. Radku Burgetovi, Ph.D. za odborné vedení a rady při jejím zpracování.

Obsah

Kapitola 1

Úvod

V současné době se využívají webové prohlížeče celosvětově každý den nejen na počítačích, ale i mobilech či tabletech. Proto je nutné věnovat se vývoji nástrojů, které zjednoduší tvorbu webových dokumentů a zpříjemní uživateli užití webových stránek na jakémkoliv zařízení. Z toho vyplývá, že na tyto dokumenty jsou kladeny stále vyšší požadavky co se týče přizpůsobování velikosti okna, přičemž se mluví o takzvanému responzivním obsahu. Jedním z nových, zato často používaných nástrojů pro rozložení, je Flexible Box Layout, zkráceně Flexbox, který byl vytvořen právě za těmito účely. Ten nabízí zjednodušené zarovnání, rozmístění a pořadí prvků na stránce. Flexbox je součástí nejnovější verze Kaskádových stylů z roku 2014, je ovšem stále ještě ve vývoji.

Tato bakalářská práce má za úkol rozšířit experimentální zobrazovací stroj CSSBox, který slouží pro vyhodnocování webových dokumentů a případně jejich následnému zpracování, právě o vlastnosti Flexboxu. Dále je třeba implementované rozšíření otestovat pomocí vhodné testovací sady a zhodnotit dosažené výsledky.

V kapitole 2 jsou vysvětleny základní technologie, které je potřeba nastudovat pro snadnější pochopení této práce. V této části se čtenář seznámí s popisem jazyků HTML a CSS, přičemž je zaměřeno hlavně na pochopení jazyka jako celku a jeho účelu. Dále je stručný přehled CSS verzí, CSS Box modelu a používaných rozložení obsahu. Následuje kapitola 3, která se podrobně zabývá modulem Flexbox. Důraz je kladen hlavně na pochopení, v čem je toto nové rozložení výhodné použít, jaké novinky ve světě rozložení webových stránek přináší a také vymezení pojmů s ním spjatých.

V kapitole 4 je rozbor knihoven CSSBox a jStyleParser, se kterými bylo v rámci implementace této práce pracováno. Poté je nastíněna struktura a hierarchie boxů použitá v tomto projektu, tyto boxy představují prvky webové stránky jako jsou bloky, text, tabulky atd. Taktéž jsou zde popsány některé nedostatky knihovny CSSBox a vysvětlen styl pozicování v tomto projektu.

V následující kapitole 5 je zdokumentován návrh rozšíření o podporu Flexboxu a přesunutí rozložení hlavních typů boxů do vytvořených tříd kvůli větší přehlednosti kódu. Následuje rozbor implementace v kapitole 6, kde je důkladně popsána realizace navržených rozšíření pro knihovnu CSSBox.

Nakonec je uveden popis postupu při testování, výběr testovací sady a rozbor a vyhodnocení výsledků, v poslední části jsou pak deklarovány možnosti dalšího vývoje a rozšíření pro přesnější zobrazení Flexbox prvků.

Kapitola 2

Základní technologie

V této kapitole budou čtenáři vysvětleny využívané technologie ve webovém prostředí, které je nutné znát pro snadnější pochopení zbylých kapitol, zejména pak kapitoly věnované implementaci (viz. kapitola 6). V první části se čtenář seznámí s pojmy HTML, DOM model a CSS. U CSS budou nastíněny výhody a nevýhody, jednotlivé verze a celkově jeho vývoj. Dále bude vysvětlen CSS Box Model následován popisem dvou nejčastějších typů elementů dle vlastnosti CSS s názvem display, těmito elementy jsou blokový a inline. Na konci této kapitoly jsou popsány používané typy rozložení obsahu a krátce zmíněn nejnovější, ovšem již často používaný typ responzivního rozložení – Grid Layout.

2.1 HTML

HTML [15] (Hypertext Markup Language) je značkovací jazyk sloužící k vytváření webových stránek v systému WWW (World Wide Web). Jinými slovy se jedná o jazyk pro tvorbu HTML dokumentů. Definují se v něm tagy (značky), které reprezentují příkazy jazyka. Ty se zapisují do špičatých závorek. Existují tagy párové a nepárové, dle toho zda je nutné zadat ukončovací tag. Ten se zapisuje stejně jako počáteční s tím rozdílem, že v ukončovacím tagu se před názvem tagu vyskytuje znak lomítko. Webové prohlížeče slouží pro zobrazení HTML dokumentů, nezobrazují však tagy. Ty totiž slouží k definování, jak se obsah tagu (obsah mezi počátečním a ukončovacím tagem) zobrazí. Základními tagy jsou `<html>`, `<head>`, `<title>` a `<body>`. Tag `<html>` definuje kořenový element HTML stránky, tag `<head>` specifikuje metadata o dokumentu, tag `<title>` je nepovinnou součástí tagu `<head>` a vyjadřuje hlavní titulek stránky a nakonec tag `<body>` obsahuje skutečný obsah stránky. Pro upřesnění tagů je jim možné nastavovat atributy. V dnešní době se však již používají hlavně na nastavení selektoru pro Kaskádové styly (viz. 2.3) a do speciálních tagů v hlavičce HTML dokumentu.

Tento jazyk se používá s dalšími technologiemi jako jsou jazyky CSS (Kaskádové styly), JavaScript a další, protože v HTML se pouze nadefinuje text, obrázky a jiné komponenty, nikoliv pokročilé chování či formátování těchto komponent, které tyto rozšiřující technologie poskytují. Nepovažuje se tedy za programovací jazyk, nekompiluje se, je snadno čitelný a díky své struktuře také snadno zpracovatelný jádrem prohlížeče. Nyní je od roku 2014 ve verzi HTML5, který sjednotil původní verze s jazykem XHTML (Extensible Hypertext Markup Language).

2.2 DOM model

Document Object Model [12] neboli objektový model dokumentu je API[1] pro popis a práci s objekty v HTML, XHTML a XML[2] dokumentech. Vytváří stromovou strukturu objektů, která se nazývá DOM strom. Těmito objekty jsou elementy, atributy, text a další. Díky DOM stromu dokážeme rozlišit nadřazené, podřazené elementy a také elementy na stejné úrovni.

W3C[3] DOM se nazývá multiplatformní specifikace, která způsobila sjednocení rozhraní webových prohlížečů, které dříve používaly různá rozhraní, a tím ztěžovali práci jejich vývojářům. Obecně také zakazuje používání elementů a atributů, které nejsou ve standardu definovány. [1]

2.3 CSS

CSS (Kaskádové styly) [11] řeší vzhledovou stránku webového dokumentu. Jedná se o deklarativní stylový jazyk vytvořený konsorciem W3C, který je jediným používaným jazykem pro vzhled webových dokumentů, především díky své jednoduchosti a intuitivnímu použití. Přidává stylování (například fonty, barvy či animace) do webových dokumentů. Syntaxe je poměrně jednoduchá a sestává se ze stylových pravidel. Tato pravidla se skládají ze selektoru a bloku deklarací. Selektor vybírá na jakou část HTML kódu bude styl aplikován a blok deklarací udává jaké styly budou použity. Selektory mohou být zadávány buď jako element, třída nebo id. Na následujícím obrázku 2.1 je h1 selektor elementu a ve složených závorkách je seznam stylových deklarací v bloku. Samotná deklarace se dá ještě rozdělit na dvě části – název vlastnosti a jeho hodnota.

```
h1 {
      background-color: red;
      text-align: center;
}
```

Obrázek 2.1: Příklad použití CSS pro obarvení a vycentrování nadpisů h1.

Výhody použití CSS

Užití CSS přináší v porovnání se samotným HTML převážně výhody, těmi jsou hlavně rozsáhlé možnosti formátování a rychlejší a přehlednější kód, kvůli oddělení struktury od stylu. Velkým přínosem je rovněž podpora dynamické změny CSS stylů pomocí jazyku JavaScript. CSS není výhodné jen pro stylování HTML dokumentů, ale také pro formátování jazyka XML[4]. Další a rozhodně ne poslední výhodou je, že pomocí CSS lze vytvářet různé styly pro různá výstupní zařízení. [5]

[1] Application Programming Interface – rozhraní pro programování aplikací

[2] eXtensible Markup Language – značkovací jazyk umožňující snadno uchovávat strukturovaná data a vyhledávat v nich

[3] World Wide Web Consortium – mezinárodní konsorcium vyvíjející webové standardy pro World Wide Web

[4] XML (Extensible Markup Language) – jazyk pro vytváření konkrétních značkovacích jazyků

Nevýhody použití CSS

Hlavní nevýhodou je ne vždy dokonalá podpora CSS v prohlížečích. Často se v nich vyskytují implementační chyby i delší dobu po vydání nové verze, které neodpovídají specifikaci CSS, takže v různých prohlížečích se výsledný dokument může zobrazit různě. Tento problém se však poslední dobou stále zlepšuje a v prohlížečích již mnoho chyb oproti specifikaci nebývá. [5]

2.3.1 Verze CSS

V následujících odstavcích jsou krátce uvedeny základní informace o dosavadních verzích [4] tohoto jazyka. Je kladen důraz hlavně na přidané vlastnosti jednotlivých verzí. Je nutné zmínit, že ne o všech vlastnostech ze starších verzí platí, že jsou podporovány i v novějších verzích, některé jsou dokonce zavrhnuté.

CSS 1

První verze CSS byla vydána na konci roku 1996 a popisuje jazyk CSS jako jednoduchý model pro stylování HTML tagů. Obsahovala základní vlastnosti pro fonty, barvu pro text a pozadí. Také byly představeny vlastnosti pro šířku mezi slovy, řádky apod. Byly zavedeny základní selektory, hodnoty a většina jednotek, které se používají dodnes. Tato prvotní verze bohužel nebyla dlouho podporována všemi používanými prohlížeči.

CSS 2

Verze CSS2 přinesla zejména vlastnosti pro stránkové rozložení, konkrétně pozicování, přetékání obsahu a překrývání. Také vznikl CSS Box Model (viz. 2.3.2) nebo třeba podpora tabulek. V neposlední řadě přibyly sofistikovanější metody selektorů. Na tuto verzi vyšla opravná verze CSS 2.1.

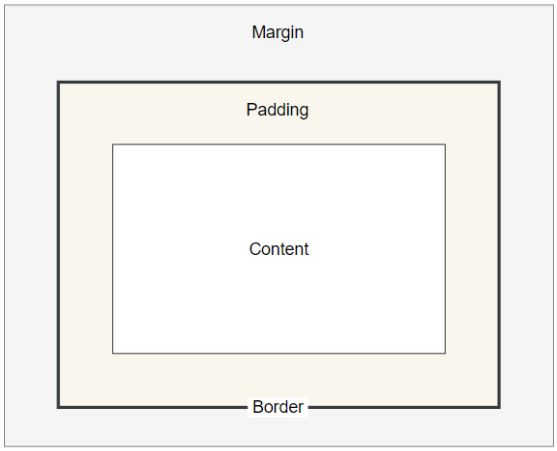
CSS 3

Nejnovější verze [8] reagovala na vznik velkého množství mobilních zařízení, od kterého se odvíjí spousta změn. Obohatila CSS o nové typy rozložení: Flexbox a Grid Layout (flexibilní a mřížkové rozložení) a přibyla také spousta nových vlastností. Při jejich přidávání byl kladen důraz na vizuální stránku textu či třeba na obrázky použité jako pozadí. Opět byly přidány nové metody selektorů a novinka v podobě dotazů na média (viz. podkapitola 2.4). V této verzi jsou již také podporovány jednoduché transformace a animace.

2.3.2 CSS Box Model

Termín CSS Box Model [6] označuje standard pro vykreslování. V rámci tohoto standardu je každý HTML element reprezentován neviditelnou obdélníkovou oblastí. Tato oblast se dělí na čtyři podoblasti, které ji definují. Jsou jimi `padding` a `margin` (vnitřní a vnější okraje), `border` (rámeček) a `content` (obsah), které jsou vysvětleny v následujících odrážkách. Názornější vysvětlení poskytuje diagram na obrázku 2.2.

- `obsah` (`content`) je plocha dostupná pro vykreslení obsahu tohoto elementu
- `vnitřní okraj` (`padding`) elementu se nazývá volná oblast kolem obsahové části z druhé strany ohraničená rámečkem.
- `rámeček` (`border`) odděluje vnitřní a vnější okraje
- `vnější okraj` (`margin`) se podobá vnitřnímu okraji, ovšem vnější okraj se nepovažuje za součást elementu. Vytváří prázdnou oblast mezi sousedními elementy.



Obrázek 2.2: Grafická reprezentace CSS Box Modelu.

Tyto oblasti mohou mít libovolné velikosti zadané CSS vlastnostmi, uvedenými v závorkách u každé z nich, pouze oblast obsahu se nejčastěji nastavuje pomocí vlastností *width* a *height*. Často však je jediná nenulová oblast právě obsah.

Blokové a inline elementy

Každý HTML element má hodnotu CSS vlastnosti display, která určuje typ elementu. Její nejčastější hodnoty jsou *block* a *inline*. Blokové elementy jsou například odstavec (`<p>`), formulář (`<form>`) či článek (`<article>`). Začínají vždy na začátku nového řádku a roztáhnou se přes celou dostupnou šířku jejich rodičovského bloku. V případě, že rodičem je kořenový element (tzv. Viewport), zabírají celou šířku stránky. Jde jim sice libovolně nastavit šířka a výška, ovšem pokud se nastaví šířka, tak zbylá dostupná šířka v řádku připadne pravému marginu tohoto elementu. Oproti tomu inline (řádkové) elementy nemusejí začínat na začátku řádku a neroztahují se více než potřebují pro velikost svého boxu. Příkladem tohoto typu elementu je třeba obrázek (`<img>`). Dalšími použitelnými hodnotami vlastnosti display jsou například inline-block, list-item, grid či flex. Speciální elementy `<div>` (blokový) a `<span>` (inline) jsou samotné bez významu, používají se pouze pro naformátování podelementů pomocí CSS vlastností. [14]

2.4 Typy rozložení

Rozložení (layout) [10] je vzor, který definuje strukturu webové stránky. Dá se dělit dle několika parametrů. Nejjednodušší dělení je podle počtu sloupců, kdy se dělí na jednosloupcové,

dvousloupcové či třísloupcové. Vytvoření rozložení s vyšším počtem sloupců je zřídkakdy použitelné v praxi. Nynější nejvýznamnější dělení je však dle schopnosti přizpůsobování se velikosti okna prohlížeče. Existující typy jsou popsány v následujících podkapitolách. Flexibilní rozložení bude komplexně rozebráno samostatně v kapitole 3.

Fixní rozložení

Fixní rozložení (fixed layout) [10] stránky používá přednastavenou velikost šířky, která se nijak nemění vůči šířce prohlížeče. Tato hodnota je nejčastěji zadávaná v pixelech, tedy absolutních jednotkách. Rozdílná zařízení budou zobrazovat prvky tohoto rozložení různě, takže vykreslená stránka se může chovat velmi nepředvídatelně. Například při přednastavené šířce 1200 pixelů se po stáhnutí prohlížeče na menší hodnotu objeví nežádoucí horizontální rolovací lišta. Tato metoda je již zastaralá a nepoužívá se.

Tekuté rozložení

Tekuté rozložení (fluid layout) [10] oproti fixnímu nepoužívá absolutní jednotky, ale procenta či relativní jednotky. Pojem relativní jednotky zahrnuje například speciální ex nebo eM jednotky. Tato použitá velikost poté určuje zabraný prostor v prohlížeči. Tekuté rozložení se zdá být optimální, ovšem na příliš velkých nebo naopak malých obrazovkách většinou nevypadá dobře, neboť se může příliš roztáhnout nebo naopak smrsknout.

Responzivní rozložení

Responzivní rozložení (responsive layout) využívá kombinaci tekutého rozložení a přizpůsobivých prvků. Může být také ovlivněn tzv. CSS dotazy na média (CSS Media Queries). Dotaz na médium je logický výraz, umožňující zjistit šířku prohlížeče a na jejím základě poté rozložit obsah. Při snižování či zvyšování šířky se rozložení s podporou dotazů na média roztahuje jako tekuté rozložení, ovšem při dosažení určité šířky označené tzv. media query breakpointem se rozkládání změní a přizpůsobí se šířce. Příklad na obrázku 2.3 nastavuje třídě content zadané CSS vlastnosti za předpokladu, že je šířka prohlížeče menší nebo rovna 800 pixelům. K tomu je třeba nastavit této třídě vlastnosti chování v ostatních případech. V posledních letech se tento typ rozložení stává čím dál více populárnější. [10] Podtypy responzivního rozložení jsou hlavně mřížkové rozložení, popsané dále v této kapitole a flexibilní rozložení, které je tématem této práce a bude podrobně popsáno v kapitole 3.

```
@media screen and (max-width: 800px) {
    .content {
       float: none;
       width: 100%;
    }
}
```

Obrázek 2.3: Příklad použití dotazů na média.

Grid Layout

Jedním z mocných a důležitých responzivních rozložení, na kterém jsou dnešní moderní weby stavěny, je Grid Layout [9] (mřížkové rozložení). Jedná se o nový CSS modul definující dvojrozměrný systém uspořádání webové stránky. Jak již název napovídá, rozložení vytvoří hypotetickou mřížku (kontejner pro obsah) a pomocí vertikálních a horizontálních linek je rozdělena na hypotetické oblasti, takzvané stopy (track). Tyto stopy představují prostor pro umisťování grid položek. Kontejner se dá nadefinovat pomocí CSS vlastnosti `display`, které se nastaví hodnota *grid*. Jeho mřížka se dá různě přizpůsobit dle potřeby. S příchodem Grid Layoutu byla vytvořena i nová jednotka s názvem fr, která reprezentuje část z dostupného místa kontejneru. Ve Flexboxu mají podobný efekt vlastnosti `flex-shrink` a `flex-grow` vysvětlené v podkapitole 3.5.

Kapitola 3

Flexbox

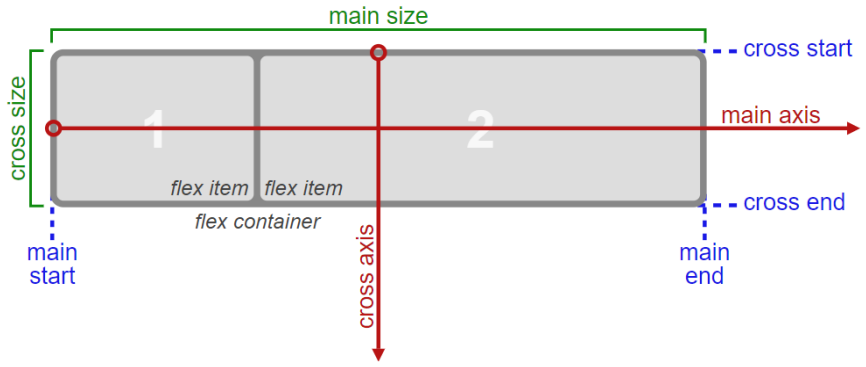
Cílem této kapitoly je čtenáře dopodrobna seznámit s flexibilním rozložením, které vzniklo jako lepší alternativa k obtékání (CSS vlastnost float). V první části jsou o tomto novém typu rozložení vysvětleny základní informace. Poté v podkapitolách 3.2 a 3.3 získá čtenář přehled o terminologii související s Flexboxem, který mu pomůže porozumět chování tohoto rozložení. Následující podkapitoly jsou věnovány rozmístění a zarovnání obsahu v tomto typu boxu. Poté jsou shrnuty výhody použití Flexboxu a na závěr je Flexbox porovnán s Grid Layoutem.

3.1 Základní informace

Flexible Box Layout, zkráceně Flexbox (flexibilní rozložení, popř. rozložení pomocí pružných boxů) je první nástroj v CSS skutečně určený k rozkládání obsahu. Funguje na podobném principu jako rozložení bloků, ovšem je více zaměřen na zarovnání a distribuci volné plochy. Je určený hlavně pro rozložení obsahu uvnitř stránky, ovšem lze jej použít i pro celostránkové rozložení. Flex znamená v překladu pružný, přizpůsobivý. Flexboxy jsou tedy pružné elementy rozložení. [8] Definují se pomocí nových hodnot vlastnosti display – `flex` a `inline-flex`. Touto definicí z elementu vznikne flex kontejner, který může být orientován buď řádkově nebo sloupcově. Je také možné zvolit opačný směr, tedy zprava doleva pro řádek a zdola nahoru pro sloupec. Přímí potomci flex kontejneru se stanou flex položkami, které mohou být řazeny v jedné nebo více liniích. Na ty se neaplikují některé zastaralejší CSS vlastnosti, konkrétně float, clear a vertical-align. Na oplátku však Flexbox poskytuje nové vlastnosti a stává se velmi mocným nástrojem v distribuci volného místa a zarovnání obsahu, které jsou na webových stránkách v dnešní době vyžadovány a dříve bylo tyto požadavky mnohem náročnější splnit. [13] Podpora Flexboxu v moderních prohlížečích je takřka stoprocentní, proto je velmi vhodný k okamžitému použití.

3.2 Terminologie

S Flexboxem se pojí pár zavedených termínů, kterým je třeba pro další kapitoly porozumět. Jsou jimi flex kontejner, flex položka, osy a rozměry Flexboxu a flex linky. Také je zde popsán rozdíl mezi flex a inline-flex hodnotou CSS vlastnosti display. Obrázek 3.1 lehce nastiňuje některé termíny se kterými se čtenář setká v následujících podkapitolách.



Obrázek 3.1: Základní zavedené termíny demonstrované na řádkovém flex kontejneru. [13]

Flex kontejner a flex položka

`Flex kontejnerem` (flex container) se element stává nastavením CSS vlastnosti `display` na hodnotu *flex* nebo *flex-inline*. Jeho přímí potomci se tímto stanou flex položkami. Je důležité si zapamatovat, že flex rozložení se vztahuje právě pouze a jen na příme potomky. Rozdíl mezi hodnotami vlastnosti `display` je následující:

- `flex` – tato hodnota způsobí blokový kontejner, tedy bude začínat na začátku řádku a zabere celou dostupnou šířku
- `inline-flex` – tato hodnota způsobí inline (řádkový) kontejner, tedy nemusí začínat na začátku řádku a zabere pouze šířku, kterou potřebuje (viz. 2.3.2)

`Flex položka` (flex item) představuje část přímého obsah kontejneru, konkrétně jeden element. Pokud je to text (a neobsahuje pouze bílé znaky), je obalen do anonymního bloku, který se stane položkou místo textu. Flex položka může být současně i flex kontejnerem, tedy Flexbox podporuje zanořování flex kontejnerů. Na absolutně pozicované položky se neaplikují flexibilní rozložení, nýbrž klasicky fixní, statické.

Položkám i kontejneru lze nastavit klasické vlastnosti určující rozměr obsahu, tedy šířka, výška a maximální a minimální hodnoty, kterých může jejich obsah nabývat.

Hlavní a vedlejší osa

Hlavní a vedlejší osa [13] společně určují směr, jakým budou položky v kontejneru rozloženy. Hlavní osa (main axis) je dána hodnotou vlastnosti `flex-direction`. Pokud je zadaná na hodnoty `row` nebo `row-reverse`, je hlavní osa horizontální, v případě `column` nebo `column-reverse`, je vertikální. Dále lze tedy i kontejner považovat za horizontální, potažmo vertikální. Vedlejší osa (cross axis) je vždy kolmá k nastavené hlavní ose. Hodnoty s reverse mění osy o 180° oproti normálu, takže osa x směřuje zprava doleva a osa y zdola nahoru.

Vertical-mode

Tato vlastnost má také podíl na vyhodnocení směru hlavní osy, neboť určuje směr textu, a může nabývat hodnot *vertical-rl* nebo *vertical-lr*, tzn. vertikální směr textu otočen o 90° doleva nebo doprava. Tyto hodnoty vymění hlavní a vedlejší osu kontejneru.

Hlavní a vedlejší rozměr

Velikosti pro obsah ve Flexboxu se nazývají hlavní (main-size) a vedlejší (cross-size) rozměr dle osy, ve které se nacházejí. Ve flex kontejneru vymezují prostor určený pro rozkládání položek. Tyto rozměry mohou být, nezávisle na vlastnosti `flex-direction`, určeny pomocí klasických CSS vlastností pro rozměr[1], proto je možné, že položky mohou přetékat i mimo rozměry kontejneru, především na vedlejší ose. V případě nezadání hodnoty pro hlavní rozměr se kontejner chová rozdílně v závislosti na hlavní ose – pokud je horizontální, nastaví se na šířku prohlížeče, pokud vertikální nastaví se ze začátku na 0 a s přidáváním položek bude zvětšován o jejich hlavní rozměr (v tomto případě výšku). Vedlejší rozměr kontejneru při nezadání zmíněných CSS vlastností je v horizontálním kontejneru řešen jako součet vedlejších rozměrů (v tomto případě výšek) flex linek, které budou probírány v následující podkapitole, ve vertikálním kontejneru se nastaví na šířku prohlížeče.

Flex linky

Flex položky jsou uspořádány a zarovnány ve flex linkách (flex lines). Ty představují hypotetické rozdělení flex kontejneru na části. Slouží pro seskupení položek a jejich horizontální a vertikální zarovnání dle Flexbox algoritmu. Flex linky obsahují určitý počet položek, vždy však minimálně jednu. Zda bude mít kontejner jednu nebo více linek (dále označováno jako single-line a multi-line) určuje jeho vlastnost `flex-wrap`. V případě hodnoty *wrap* nebo *wrap-reverse* této vlastnosti se bude kontejner chovat jako multi-line přičemž jeho položky budou řazeny vedle sebe v hlavním rozměru a zalamovány v případě, že se již další na hlavní osu kontejneru nevejde[2]. Tímto se další linka vytvoří a posune ve vedlejší ose oproti předchozí lince o její vedlejší rozměr. Poté se do ní přetékající položka vloží. V případě hodnoty *nowrap* jsou položky umisťovány do jedné linky a nehledí na šířku obsahu kontejneru, tzn. že jejich obsah může přetéct mimo horizontální kontejner pokud je jeho šířka nebo maximální šířka nastavena (stejně tak s výškou ve vertikálním kontejneru). Pokud není, a obsah přeteče, tak se kontejner v hlavním rozměru roztáhne (platí pouze pro vertikální kontejner). Je důležité podotknout, že multi-line kontejnery, které mají pouze jednu linku, neaplikují chování single-line kontejnerů.

Pojmy hlavní a vedlejší rozměr se vztahují i k flex linkám. Linka má svůj hlavní rozměr stejný, jako je hlavní rozměr kontejneru. U vedlejšího rozměru je to komplikovanější. Ten je vždy právě takový jako je součet největšího vedlejšího rozměru ze všech položek v lince se vzdáleností této položky od začátku vedlejší osy linky. [13]

3.3 Nastavení hlavního rozměru flex položek

Zjistit rozměr flex položek v hlavní ose je nutné, jak pro jejich vykreslení a napozicování, tak pro zjištění a nastavení hlavního rozměru vertikálního flex kontejneru. Nejprve je však potřeba nastavit položce tzv. `hypotetický rozměr`. Ten se nastavuje pomocí vlastnosti `flex-basis` a lze nastavit různými typy jednotek (px, %, eM, ...). Pokud je `flex-basis` nastavena na hodnoty *auto*, *content* či vůbec nenastavena, hypotetický rozměr se určí pomocí vlastností `width` či `height` (v závislosti na vlastnosti `flex-direction`). Pokud ani takto není rozměr nastaven, nastaví se na minimální hodnotu obsahu (*min-content*) v hlavní ose kontejneru.

[1] width, min-width, max-width pro horizontální osu; height, min-height, max-height pro vertikální osu

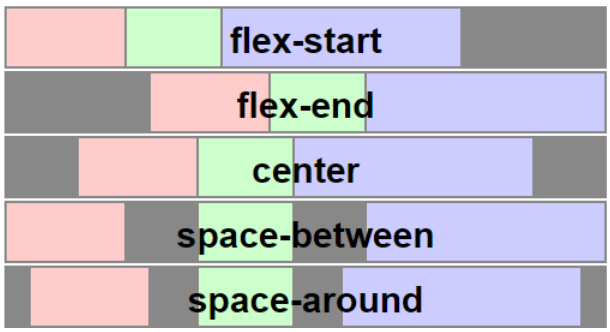
[2] pro vertikální kontejner bez nastaveného hlavní rozměru se tedy položky rozkládají jakoby byl single-line

Hypotetický rozměr je poté ohraničen vlastnostmi určující minimální a maximální rozměry v hlavní ose. Konečný rozměr položky je nastaven až po aplikování tzv. flex faktorů (viz. podkapitola 3.5), které ho mohou ještě přizpůsobit.

3.4 Zarovnání flex položek ve flex kontejneru

Flex položky se rozkládají a zobrazují, jak již bylo uvedeno, v hlavní ose jedna vedle druhé dle pořadí ve zdrojovém kódu. Toto pořadí lze částečně změnit hodnotou *wrap-reverse* vlastnosti `flex-wrap`, případně v rámci linky pomocí hodnot *row-reverse* nebo *column-reverse* vlastnosti `flex-direction`. Tímto způsobem však nelze vyřešit náročnější požadavky na pořadí, byla tedy vytvořena vlastnost `order` pro flex položky, přijímající celá čísla, díky které lze nastavit přesné pořadí položek v kontejneru.

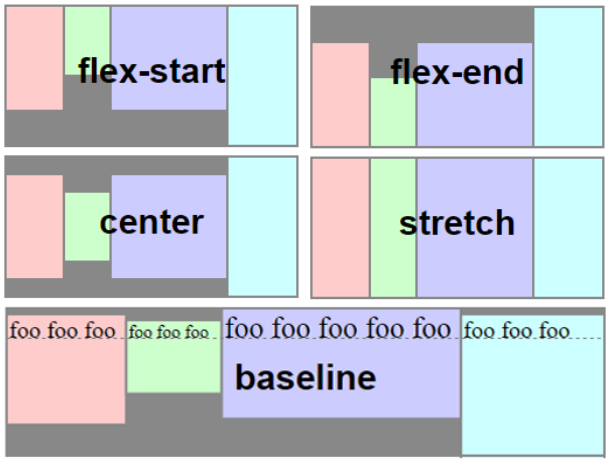
Zarovnání položek v kontejneru je možné v obou osách několika vlastnostmi. Zarovnání v hlavní ose linek, je možné pomocí vlastnosti `justify-content`. Tato vlastnost nelze nastavit jednotlivým položkám, pouze kontejneru jako celku. Je důležité zmínit, že je aplikována až po vyřešení hlavního rozměru, flex faktorů (viz. 3.5) a auto marginů. Má efekt vždy, nejen pokud v lince zbylo volné místo, které distribuje položkám nebo do prostoru kolem nich, ale i pokud položky přetečou přes kontejner. Tehdy tato hodnota určuje, jakým směrem (popřípadě oběma směry v případě hodnoty *center*) budou položky přetékat. Obrázek 3.2 ukazuje všechny možné hodnoty a jejich efekt na položky.



Obrázek 3.2: Efekt všech možných hodnot vlastnosti `justify-content` na položky horizontálního kontejneru. Převzato z [13].

Vlastnost `align-items` zarovnává položky na vedlejší ose jejich flex linek. S touto vlastností lze položky v kontejneru rozmístit pěti způsoby: k začátku, středu nebo konci vedlejší osy flex linky, popřípadě roztáhnout položky do celého vedlejšího rozměru flex linky. Speciální hodnotou je *baseline*, která zarovná položky dle obsahu do jedné roviny. Tato hodnota jako jediná může i změnit výšku flex linky. Výchozí hodnotou a také nejpoužívanější je však roztáhnutí položek. Tato vlastnost se taktéž definuje na kontejner, nikoliv na položky. Efekt zarovnání položek dle jejích možných hodnot zobrazuje obrázek 3.3.

V případě potřeby nastavení zarovnání ve vedlejší ose jen některým položkám je nutné použít vlastnost `align-self`, která má stejné hodnoty jako `align-items` (viz. 3.3), ovšem ty se aplikují pouze na položku nebo skupinu položek, jimž byla zadána. V případě zadání obou vlastností je prioritně použita `align-self`. [7]



Obrázek 3.3: Efekt všech možných hodnot vlastnosti `align-items` na položky horizontálního kontejneru. Převzato z [13] (upraveno).

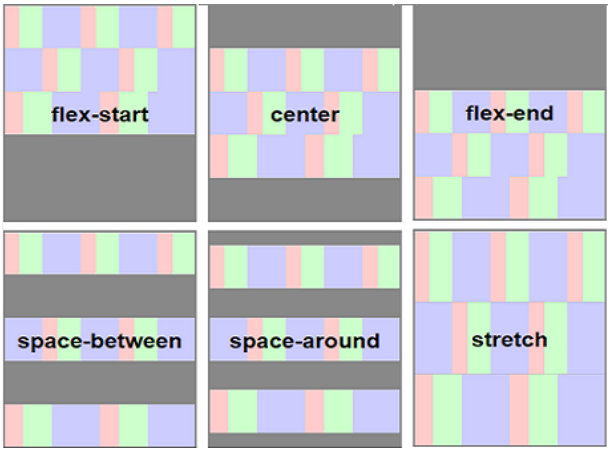
3.5 Rozmístění volného místa v lince

Mimo běžného zabrání místa položkami lze také nastavit položkám, aby se roztahovali do celé linky nad rámec své velikosti v případě volného místa v hlavním rozměru linky. Toto nadbytečné místo lze přidělit žádné, jedné nebo více položkám. K tomuto účelu byla vytvořena vlastnost `flex-grow`, určující jak moc se položka může roztáhnout oproti ostatním v lince[3]. Na stejném principu funguje i vlastnost `flex-shrink`, která však dělá pravý opak, tedy v případě nedostatku místa se položka smrskne. Tyto dvě vlastnosti se dohromady často označují jako flex faktory. Je důležité dodat, že přestože se flex položky mohou tímto způsobem smršťovat a natahovat, tak vždy musí tolerovat zadané minimální a maximální hodnoty a nikdy tato omezení nepřekročit. [7]

3.6 Zarovnání flex linek ve flex kontejneru

Volné místo lze využít nejen v hlavní ose kontejneru ale i ve vedlejší. Vlastnost `align-content` určuje, jak se volné místo ve vedlejším rozměru kontejneru bude distribuovat mezi flex linky a kolem nich. Vztahuje se pouze na multi-line kontejnery, na single-line sice lze nastavit, ale nemá žádný efekt. Narozdíl od vlastností `align-items` a `justify-content` se nevztahuje k jedné určité lince, nýbrž ke všem linkám kontejneru najednou. Použitelné hodnoty a jejich efekt jsou zobrazeny na obrázku 3.4. Výchozí hodnotou je *stretch*, tedy celý volný prostor je rozdistribuován rovnoměrně mezi všechny linky. Toto roztažení může mít za následek i změny v uspořádání položek, například položka s hodnotou vlastnosti `align-self` *stretch*, se roztáhne do celého nově změněného vedlejšího rozměru linky. Tato vlastnost také určuje, jakým směrem má obsah multi-line kontejneru ve vedlejší ose přetéci, pokud již nezbývá volné místo. To se samozřejmě může stát pouze v případě nastaveného vedlejšího rozměru kontejneru. [7]

[3] vytváří se poměr hodnota flex-grow položky ku součtu hodnot flex-grow všech položek v lince



Obrázek 3.4: Zarovnání flex linek v horizontálním flex kontejneru použitím vlastnosti `align-content`. Převzato z [13] (upraveno).

Zkratky pro vlastnosti

Pro Flexbox byly vytvořeny i dvě vlastnosti, které nepřidávají novou funkcionalitu, pouze vhodně sjednocují jiné vlastnosti, které spolu souvisí. Jsou jimi:

- `flex-flow`: spojení vlastností `flex-direction` a `flex-wrap`, které společně určují hlavní a vedlejší osu kontejneru a zda bude kontejner single-line nebo multi-line
- `flex`: spojení vlastností `flex-basis`, `flex-grow` a `flex-shrink`, které specifikují základní velikost a celkovou flexibilitu položky, tedy jak moc se může zvětšovat, popř. zmenšovat dle potřeby oproti ostatním položkám

Ne všechny hodnoty těchto vlastností je vždy nutné zadat, parsery používané v prohlížečích jsou na to připraveny. V případě nezadání některých vlastností se aplikují jejich výchozí hodnoty. Při zadání jedné vlastnosti jak zkratkou, tak klasicky, se aplikuje ta hodnota, která byla zadaná jako poslední.

Přestože je možné tyto vlastnosti používat i odděleně, je důrazně doporučeno používat právě tyto zkratky. Například proto, že samostatné nastavení flex faktorů může učinit kód mnohem méně přehledný.

Absolutní pozicovaní flex položek

Absolutně pozicované položky kontejneru se rozkládají speciálně, podobně jako ostatní absolutně pozicované boxy. Jejich pozice v kontejneru je určena stejně, jakoby byla v kontejneru pouze jediná položka. [13]

Výhody použití Flexboxu

V předchozích podkapitolách byly popsány vlastnosti Flexboxu a jeho chování, v této budou shrnuty výhody z nich vyplývající. Jsou jimi především flexibilita položek dle dostupného místa, možnost vykreslovat položky v jakémkoliv směru i pořadí, či snadné, ovšem propracované zarovnání položek v rámci linky i celého kontejneru. Flexbox je vhodné použít pro jeho jednoduchost, intuitivnost a téměř stoprocentní podporu novými verzemi prohlížečů. [7]

Porovnání Flexboxu a Grid Layoutu

Na začátku je nutné zmínit, že tyto dvě metody se nenahrazují a každá byla vytvořena pro jiné účely. Grid Layout je vhodný pro dvourozměrné rozložení, tzn. je jak řádkový, tak sloupcový. Je vhodný spíše na rozložení celé webové stránky. Naproti tomu Flexbox se považuje za modul pro jednorozměrné rozložení, tedy buď řádkové nebo sloupcové, a používá se spíše pro rozložení komponent (formulářů, navigací, apod.) uvnitř webové stránky. Proto mají i typově rozdílné CSS vlastnosti. Flexbox má více zarovnávacích vlastností, oproti tomu Grid Layout mnoho vlastností určených k přizpůsobení mřížky. Některé vlastnosti Flexboxu mají efekt i na Grid Layout, jejich příkladem je `justify-content`. U obou metod se shodně používá výrazů kontejner, položka, linka a dalších. [7]

Kapitola 4

Projekt CSSBox

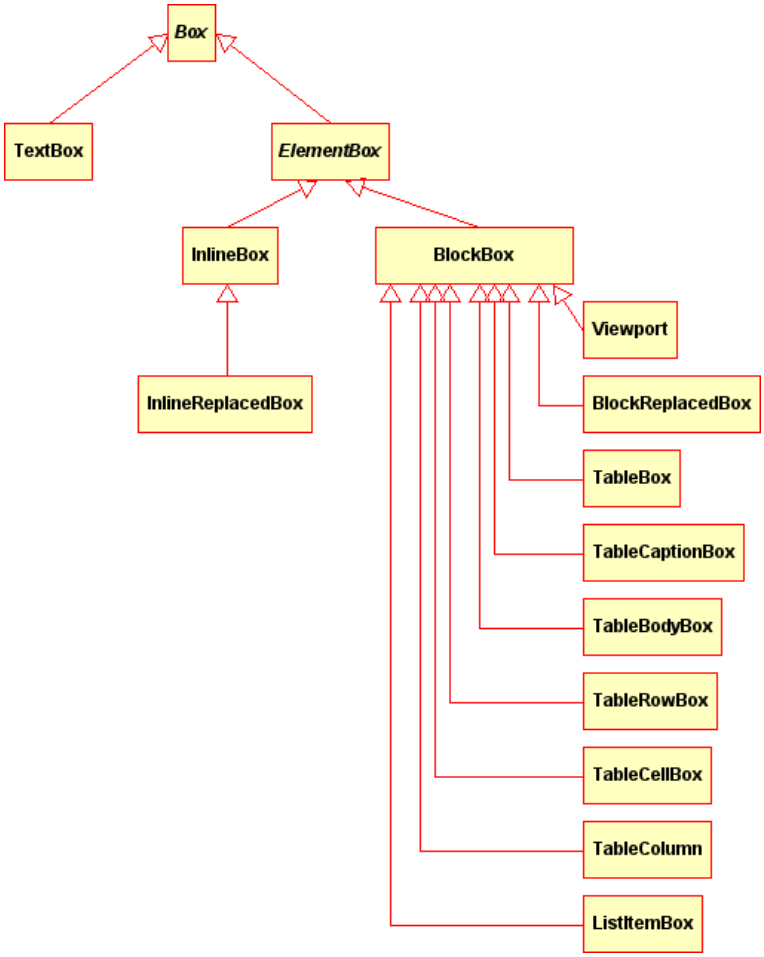
Tato kapitola popisuje co je a k čemu slouží projekt CSSBox. Zabývá se jeho vznikem, strukturou a mimo jiné i zpracováním vstupu na výstup. Taktéž je v další části krátce popsána knihovna jStyleParser, kterou projekt CSSBox využívá. Závěrem této kapitoly je podkapitola věnující se nedostatkům knihovny CSSBox.

4.1 O projektu

CSSBox je (X)HTML/CSS renderovací stroj vytvářený v jazyce Java. Tento experimentální projekt vznikl na Fakultě informačních technologií Vysokého učení technického v Brně a v současné době je pouze rozšiřován o nové moduly. Jeho primárním účelem je poskytnout kompletní informace o obsahu renderované stránky a jejím rozložení. Taktéž poskytuje data pro další zpracování. Stroj CSSBox může být také mimo jiné použit pro procházení vykreslených dokumentů v Java Swing aplikacích, neboť má vytvořeny demo aplikace. V těchto aplikacích se mimo jiné dobře ladí implementace nových rozšíření této knihovny.

Vstupem stroje je a sada stylových předpisů a dokument v podobě DOM stromu, který je získán z HTML nebo XML souboru pomocí NekoHTML parseru. Dále je zpracován soubor CSS vlastností a každému elementu je vypočítán efektivní styl. Získané výsledné rozložení dokumentu je reprezentováno stromem boxů. Výsledný objektově orientovaný model webové stránky může být buď zobrazen přímo nebo sloužit pro další zobrazovací algoritmy. Každý box v uvedeném stromu boxů může mít potomky a znázorňuje obdélníkovou oblast ve výsledné stránce, která odpovídá konkrétnímu vykreslenému HTML elementu. Jeden element může být reprezentován několika boxy, například pokud element `<p>` (odstavec) je víceřádkový, je vytvořen box pro samotný odstavec a navíc pro každý řádek textu. Box je reprezentován objektem, který je odvozen od abstraktní třídy Box. Kořenem stromu boxů je vždy uzel `Viewport`, který reprezentuje dostupnou plochu pro prohlížeč. Nyní v CSSBoxu existuje kolem deseti typů boxů dle jejich CSS vlastnosti display určující odpovídající element. [2] Obrázek 4.1 ukazuje typy boxů a jejich hierarchii. Šipky naznačují dědičnost.

Tento projekt je vyvíjen za pomoci verzovacího nástroje Git. Jeho zdrojové kódy jsou volně dostupné na webové adrese `https://github.com/radkovo/CSSBox` ve službě GitHub. Dokumentace a popis projektu se nacházejí na Sourceforge. Díky tomu, že je tvořen v čisté Javě a nejsou využity žádné externí knihovny, jedná se multiplatformní projekt.



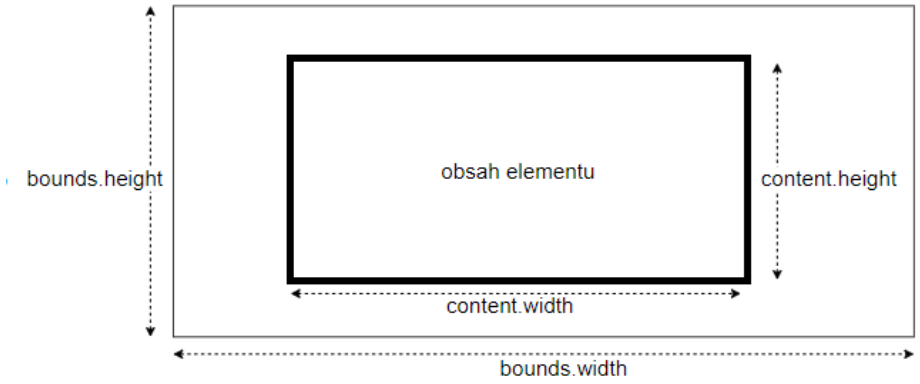
Obrázek 4.1: Dosavadní hierarchie boxů v CSSBoxu (převzato z [2])

4.2 Pozicování v knihovně CSSBox

V knihovně CSSBox lze elementy (uzavřené do boxů dle CSS Box Modelu – viz. 2.3.2) napozicovat velmi snadno. K tomu se využívá atributů typu Rectangle z abstraktních tříd `Box` a `ElementBox`, ze kterých všechny inline i blokové boxy dědí. Těmito atributy jsou `bounds` a `content`. `Content` v sobě zahrnuje jen čistě prostor pro obsah, je tedy ohraničen šířkou a výškou obsahu bez paddingů, marginů a borderu. `Bounds` reprezentuje celkovou oblast pro box, definovanou v CSS Box Modelu, tedy `content` včetně marginů, paddingů a borderu. Tyto atributy mají své vnitřní atributy dle typu Rectangle, tedy `width` (šířka), `height` (výška) a souřadnice `x`, `y`. Atributy nastavující rozměry jsou názorně zobrazeny na obrázku 4.2.

Pro nastavení pozice lze využít metody setPosition(int x, int y), která nastaví hodnoty atributů `bounds.x` a `bounds.y`. Je ale nutné nastavit i všechny ostatní vnitřní atributy, neboť bez jejich explicitního nastavení budou nulové, což je ve většině případech nežádoucí[1]. Pozice boxu je nastavována vůči levému hornímu rohu jeho rodičovského boxu, absolutní pozice vůči levému hornímu rohu stránky.

[1] pokud však například nemá element nastavenou CSS vlastnost display:none



Obrázek 4.2: Atributy v CSSBoxu určující rozměry boxů

4.3 Knihovna jStyleParser

Tato knihovna je součástí projektu CSSBox a slouží jako parser CSS stylových předpisů s vlastním aplikačním rozhraním, vytvářený stejně jako knihovna CSSBox v čisté Javě. Umožňuje efektivní zpracování CSS v Javě a mapování hodnot na datové typy, které jazyk Java umožňuje. Zpracovává předpisy podle CSS 2.1 (a nově také velkou část vlastností CSS3) a mapuje hodnoty na datové struktury jazyka Java. Z knihovny CSSBox lze snadno díky této knihovně přistupovat k nastaveným stylům pomocí takzvaných Termlistů a získat jejich hodnotu. Je testována dle specifikace CSS. [3]

4.4 Nedostatky knihovny CSSBox

V rámci studování architektury a dokumentace, a také při samotné implementaci bylo zjištěno několik nedostatků a omezení v knihovně CSSBox. Ty se mohou podepsat na vzhledu vykreslované stránky. Častým problémem bývá nesprávný rozměr textu v elementech, který bohužel nebylo možné eliminovat.

CSSBox zatím nedokáže zpracovat flashový obsah, cookies, chybí pokročilá podpora JavaScriptu a také implementace protokolu HTTPS.

Důležitým omezením je nepodporování desetinných čísel v CSS jednotkách[2], nejspíše za účelem vyšší rychlosti počítání hodnot a vykreslování obsahu.

Při testování bylo také zjištěno nepodporování vlastnosti `linear-gradient` a pro Flexbox důležité vlastnosti `writing-mode`. Proto byly při testování vytvářeny takové stránky, které tyto vlastnosti, flashový obsah ani cookies nepoužívaly. Webové stránky získané z testovací sady, nepoužitelné kvůli těmto nedostatkům, byly buď nejprve ručně upravovány a až poté použity pro testování, nebo ze sady vyřazeny.

[2] hodnota je převedena na datový typ Integer odříznutím desetinné části

Kapitola 5

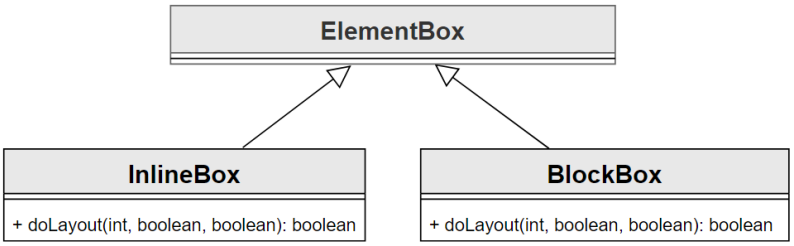
Návrh rozšíření a úprav knihovny CSSBox

V této kapitole bude nejprve popsána hlavní část aktuální struktury rozložení obsahu v knihovně CSSBox. Dále je rozebrán návrh upravující tuto strukturu za účelem možnosti zakomponovat rozložení Flexbox a také snadněji rozšiřovat knihovnu CSSBox o další nové typy rozložení. Nakonec je popsán návrh rozšíření knihovny CSSBox o podporu Flexboxu.

Při návrhu bylo postupováno tak, aby tento zásah neovlivnil funkčnost, pouze skutečně rozšiřoval projekt o nové funkce. Popisovaný návrh algoritmu Flexbox byl částečně inspirován algoritmem na oficiální stránce pro tento typ rozložení[1].

5.1 Aktuální způsob rozložení obsahu hlavních typů boxů

Hlavními typy boxů v knihovně CSSBox, od kterých se odvíjí takřka všechny ostatní, jsou `BlockBox` a `InlineBox`, které reprezentují blokové a inline elementy (viz. podkapitola 2.3.2). Jejich aktuální způsob rozložení obsahu je znázorněn na obrázku 5.1.



Obrázek 5.1: Diagram znázorňující stávající způsob rozložení dvou hlavních typů boxů a jejich potomků, které metodu `doLayout()` nepřepisují.

Tyto dvě hlavní třídy disponují metodou `doLayout()`, která je implementována přímo v nich a zajišťuje celý proces rozložení jejich obsahu. Tento přístup znemožňuje boxům přidělit jiný způsob rozložení obsahu, což je pro podporu nových efektivnějších typů rozložení potřeba změnit. Návrh této změny je popsán v další podkapitole.

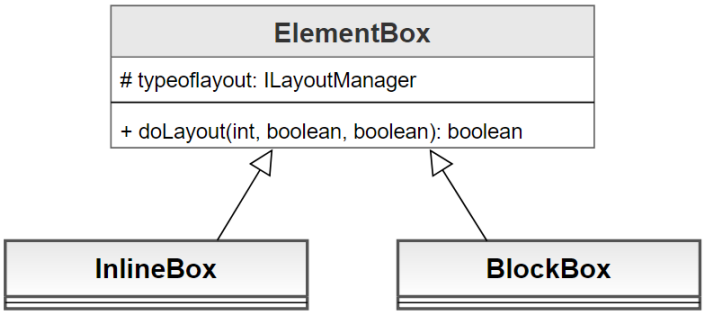
[1] https://www.w3.org/TR/css-flexbox-1/#layout-algorithm

5.2 Rozšíření o layout managery

V rámci tohoto rozšíření je potřeba upravit stávající strukturu rozložení jeho oddělením od hlavních typů boxů (`InlineBox` a `BlockBox`), za účelem možnosti přidělit boxům i jiný typ rozložení než mají výchozí. Proto byly po dohodě s vedoucím navrženy layout managery, které tímto boxům zapouzdřují jejich rozložení a kompletně ho řeší. Vedlejším důvodem použití tohoto přístupu bylo také odstínění rozložení těchto boxů od jejich tříd v rámci zpřehlednění kódu.

Bylo tedy navrhnuto obecné rozhraní `org.fit.cssbox.layout.ILayoutManager` s metodou `doLayout()`, které je implementováno třídami `org.fit.cssbox.layout.InlineBoxLayoutManager` a `org.fit.cssbox.layout.BlockBoxLayoutManager`. Metodu `doLayout()` je nutné z tříd `InlineBox` a `BlockBox` přesunout právě do jim odpovídajících layout managerů. Zároveň je v každé instanci layout manageru nutné vědět o boxu, který rozkládá, respektive pro který byl vytvořen. To lze vyřešit snadno pomocí atributu třídy odpovídajícího layout manageru.

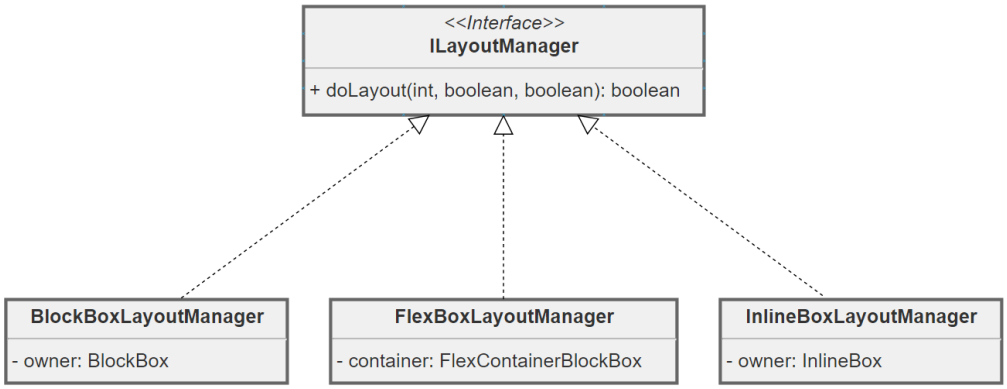
Také bude vhodné vytvořit vlastní layout manager pro Flexbox s názvem `org.fit.cssbox.layout.FlexBoxLayoutManager`, ve kterém bude metoda `doLayout()` implementována dle Flexbox algoritmu. Tato technika podporuje polymorfismus, a tedy BlockBox, InlineBox a nová třída pro flex kontejner mohou zdědit atribut `typeoflayout` typu ILayoutManager od své nadřazené třídy `ElementBox`, přičemž dle potřeby se různým instancím těchto tříd může přidělit různý layout manager již v jejich konstruktorech. Návrh této struktury zobrazují diagramy na obrázcích 5.2 a 5.3.



Obrázek 5.2: Diagram znázorňující návrh změny rozložení základních typů boxů.

Tento přístup je využitelný například pokud by se z jindy blokově rozloženého BlockBoxu měl stát flex kontejner rozložený pomocí Flexbox algoritmu. Jedná se také o vhodné řešení do budoucna – pokud by v CSS standardech přibyl další modul pro rozložení, který by byl v projektu třeba implementovat, jako například již vytvořený Grid Layout (popsaný v podkapitole 2.4), stačilo by pro něj vytvořit nový layout manager.

Výše popsaný návrh a následná implementace layout managerů byly tvořeny ve spolupráci se studentem Ondřejem Novákem, který rovněž rozšířil knihovnu CSSBox o nový typ rozložení obsahu, konkrétně právě o zmíněný Grid Layout [9].



Obrázek 5.3: Diagram znázorňující návrh struktury layout managerů.

5.3 Flexbox

Hlavní náplní této práce bylo rozšíření knihovny CSSBox o podporu rozložení označované jako Flexbox. Aby ho bylo možné implementovat, bylo potřeba navrhnout postup řešení. Diagram na obrázku 5.4 rozděluje navržený algoritmus do 8 hlavních částí. Třídám pro kontejner i položku byl zvolen jako předek třída BlockBox. Jelikož Java nepodporuje vícenásobnou dědičnost, bylo navrženo, že třída pro flex položku by představovala novou mezivrstvu mezi BlockBoxem a jeho potomky kromě třídy Viewport (viz. 4.1), které by nyní dědili právě od třídy pro flex položku, neboť všechny tyto typy boxů se mohou stát flex položkou, tedy potřebují mít přístup k flex vlastnostem a zároveň k vlastním, specifickým pro dané boxy. Také třída pro flex kontejner by měla být rovněž potomkem třídy pro flex položku, protože flex kontejner může být současně i flex položkou.

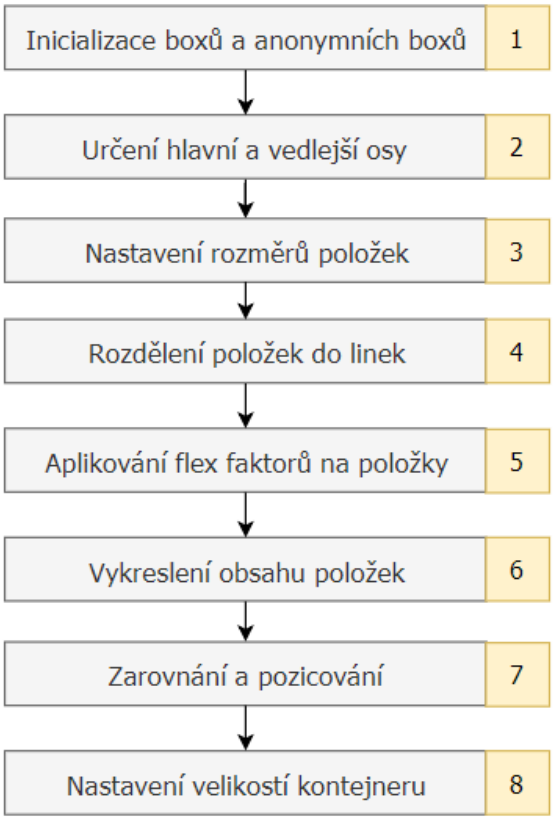
V následujících odstavcích budou jednotlivé fáze popsány podrobněji. Postup některých kroků již byl vysvětlen v kapitole 2, proto již nebude zmíněn. Tento návrh se vztahuje k horizontálnímu kontejneru, u vertikálního se fáze vykreslení obsahu položek provede před vytvořením flex linek, jinak je postup stejný.

V kroku 1 je potřeba detekovat, že je box flex kontejnerem, vytvořit pro něj instanci a nainicializovat ji. To stejné se musí udělat také pro přímé potomky kontejneru, které se stanou flex položkami. V případě, že je přímý potomek kontejneru instance Text boxu, je nutné obalit tento box do anonymního blokového boxu a ten se poté stane flex položkou místo Text boxu. Těmto boxům se za použití jStyleParseru získají a nastaví hodnoty dle nastavených CSS vlastností ve stylových předpisech, pro kontejner se také vytvoří instance layout manageru pro Flexbox. Položky je také nutné seřadit dle CSS vlastnosti `order`.

Ve druhé části je třeba zjistit, která osa bude hlavní, tedy zda se bude jednat o `horizontální` nebo `vertikální` kontejner. To je ovlivněno vlastnostmi kontejneru `flex-direction` a `writing-mode`. V této ose se poté budou rozkládat položky ve flex linkách.

Poté se v kroku 3 určí rozměr položek ve vedlejší ose a hlavně také jejich hypotetický hlavní rozměr. Oba tyto rozměry ještě musí být ohraničeny minimálními a maximálními hodnotami, pokud jsou zadány.

Ve čtvrté části jsou vytvořeny flex linky a těm poté přiřazovány položky. Linka musí mít minimálně jednu položku, dokonce i kdyby se její první položka do kontejneru v hlavním rozměru nevešla. Linkám je zvětšován vedlejší rozměr dle položky s největším vedlejším



Obrázek 5.4: Diagram nejdůležitějších fází Flexbox algoritmu.

rozměrem. Bylo navrhnuto vytvoření obecné abstraktní třídy `FlexLine`, ze které budou dědit třídy pro flex linky horizontálního a vertikálního kontejneru.

V kroku 5 jsou na hlavní rozměr aplikovány flex faktory, tedy v případě volného místa v hlavním rozměru kontejneru jsou položky roztaženy dle jejich nastavených hodnot vlastnosti `flex-grow`. Podobně v případě přesáhnutí hlavního rozměru kontejneru jsou položky smrsknuty dle hodnot vlastnosti `flex-shrink`.

Po dokončení nastavení hlavního rozměru položek je již možné vykreslit jejich obsah. To se provede na základě toho, zda jsou obsahem pouze inline elementy, či nikoliv.

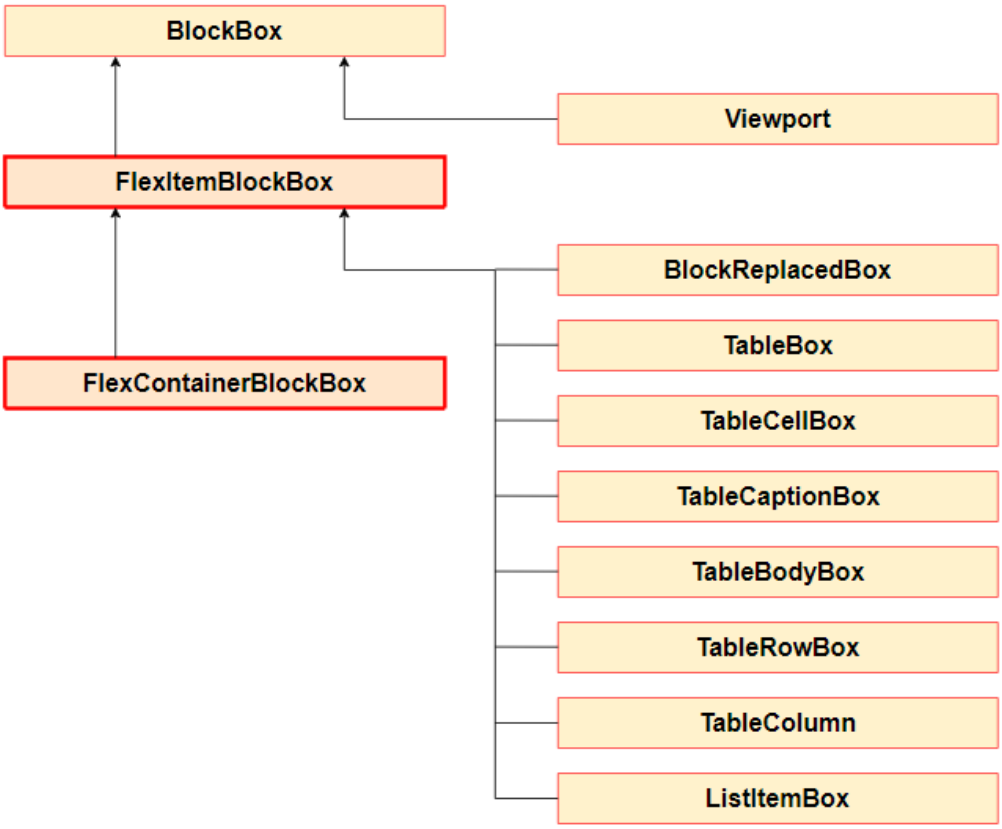
Nyní je třeba umístit linky do kontejneru a v případě volného místa ve vedlejším rozměru kontejneru linky rozmístit dle vlastnosti `align-content`. Následně je třeba nastavit položkám pozici v linkách na základě zarovnávacích vlastností `align-items`, `align-self` a `justify-content`.

Nakonec se již jen nastaví celkové velikosti boxu pro kontejner.

5.4 Navrhnuté změny v hierarchii boxů

Hierarchie boxů knihovny CSSBox před zásahem do struktury je zobrazena na obrázku 4.1 v kapitole 4. Navrhnuté změny v této hierarchii v rámci rozšíření o podporu flexibilního rozložení jsou vidět na obrázku 5.5, na kterém je pro přehlednost zobrazena pouze ta část hierarchie, ve které byly změny navrhnuty. Účel těchto změn je, aby všechny typy boxů,

které nyní dědí ze třídy BlockBox, kromě třídy Viewport[2], měly přístup k vlastnostem a metodám Flexbox položek, což je samozřejmě možné díky naznačené dědičnosti.



Obrázek 5.5: Návrh změny hierarchie boxů v knihovně CSSBox, zvýrazněné boxy jsou nově přidané (diagram inspirován převzatým obrázkem 4.1).

[2]Viewport - třída reprezentující box vykreslované stránky

Kapitola 6

Implementace

Tato kapitola se zabývá detailním popisem implementace rozšíření knihovny CSSBox o podporu Flexbox rozložení. Implementace byla prováděna dle návrhu, který byl popsán v kapitole 5. Některé související fáze návrhu byly sjednoceny do větších podkapitol. V těch je nejprve popsána inicializace boxů, poté nastavení potřebných rozměrů a mimo jiné i zarovnání flex linek a flex položek v kontejneru.

Popsané fáze rozložení týkající se flex položek probíhaly sekvenčně, tedy až po aplikování na všechny položky se přešlo do další fáze. Podobně na tom byly i flex linky. Díky tomuto oddělenému přístupu byla implementace snadnější a také se lépe ladila. Navíc bylo možné lehce zpřehlednit kód oddělením některých těchto fází do samostatných metod.

V rámci implementace bylo odhaleno několik nedokonalostí knihovny jStyleParser týkající se CSS vlastností Flexboxu, které byly konzultovány s vedoucím a opraveny.

Implementace layout managerů již dále popisována není, neboť se jednalo o okrajovou náplň této práce a vše podstatné již bylo nastíněno v kapitole 5, přičemž při implementaci se postupovalo přesně podle návrhu.

6.1 Inicializace boxů

Před samotným spuštěním Flexbox algoritmu je třeba elementům vytvořit instance příslušných boxů. Elementu s vlastností `display`:*flex* se ve třídě `BoxFactory` vytvoří instance třídy `FlexContainerBlockBox`, která představuje box pro flex kontejner. Ve stejné třídě se vytvářejí i instance třídy `FlexItemBlockBox` pro flex položky za předpokladu, že přímo nadřazený box je instancí flex kontejneru. Pokud se má položkou stát TextBox, představující čistý inline text, je nejprve obalen do anonymního blokového boxu, který se místo něj stane položkou, a jsou mu předány vlastnosti tohoto TextBoxu. Tyto změny byly provedeny v metodách `createElementInstance()` a `createAnonymousBox()` opět ve třídě `BoxFactory`.

Flex kontejneru je také v jeho konstruktoru vytvořena instance třídy FlexBoxLayoutManager do atributu `typeoflayout`, která je zděděna od ElementBoxu. Tento LayoutManager se v následujících fázích využívá k rozložení obsahu flex kontejneru dle Flexbox algoritmu.

Získání stylů CSS

Boxům pro kontejner a položky jsou také v konstruktoru přiřazeny CSS vlastnosti, získané ze stylových předpisů, patřící k elementům, které tyto boxy představují. Pro to bylo vyu-

žito metody `setStyle()`, která je volána hierarchicky i na všechny boxy, z kterých dědí[1]. K tomuto zpracování se využívá konstrukce `style.getProperty("`*nazev-stylu*`")`, dostupná z knihovny jStyleParser. Jejich nastavené hodnoty jsou uloženy do atributů těchto tříd. Většina hodnot vlastností, vztahujících se k Flexboxu, jsou v podobě řetězců, například *flex-start* nebo *wrap-reverse*, ovšem některé vlastnosti, jako třeba `order` nebo `flex-shrink` mají hodnoty číselné. V případě vlastností s číselnými hodnotami metoda `getProperty()` nezíská hodnotu, ale informaci, jak a zda byla hodnota zadána. Například získání vlastnosti `flex-basis` touto metodou pouze určuje, zda byla zadána klasicky číselnou jednotkou (length), v procentech (percentage), hodnotou auto apod. Skutečná číselná hodnota se z takto definovaných vlastností získá pomocí instance třídy `CSSDecoder`, konkrétně metodou `getLength()`[2]. Vlastnostem nespicifkovaným ve stylových předpisech je v rámci implementace přiřazena jejich výchozí hodnota popsaná v CSS specifikaci.

6.2 Počátek vytvoření rozložení pro FlexBox

Tvorba rozložení Flexboxu začne zavoláním metody `doLayout()` na příslušné instanci layout manageru flex kontejneru. Kontejneru je nejprve v této metodě nastaven hlavní a vedlejší rozměr. Poté je vytvořen ArrayList[3] flex položek tohoto kontejneru pomocí metody `getSubBox()`, který je vhodnou strukturou pro uchování objektů hlavně díky snadnému přístupu k objektům a bude používán po celou dobu na flex položky a v pozdější fázi implementace i na flex linky. Bylo by však možné použít takřka jakýkoliv typ Listu. Položkám (nikoliv však obsahu položek) jsou rovnou v této fázi odnastavené vlastnosti float a clear, protože na ně nemají mít žádný efekt. Následně je zavolána metoda pro samotné rozložení obsahu, tedy položek kontejneru. Určení metody závisí na směru kontejneru, v případě horizontálního bude obsah tvořen metodou `layoutItemsInRow()`, při vertikálním kontejneru metodou `layoutItemsInColumn()`, přičemž zadaná CSS vlastnost `writing-mode` na směr kontejneru nemá význam, protože není podporována knihovnami CSSBox a jStyleParser. Těmto metodám je předáván zmíněný ArrayList s položkami kontejneru, který byl před tímto krokem vytvořen. Tyto podtypy rozložení Flexbox obsahu byly takto rozděleny za účelem přehlednějšího kódu. Ač sice mají některé části dosti podobné, někdy jsou v jejich chování velké rozdíly, a dokonce bylo výhodnější pro vertikální kontejner implementovat některé fáze, deklarované v kapitole 5, v odlišném pořadí.

Pořadí položek

Před nastavením rozměrů byla provedena změna pořadí položek v kontejneru založená na hodnotách atributu `flexOrder`, který uchovává hodnotu flex vlastnosti `order` pro danou položku. Na to bylo opět příhodné použít zmíněný ArrayList, na který se dá aplikovat řazení pomocí volání statické metody `sort()` z třídy Collections. Toto řazení lze velmi lehce přizpůsobit, což bylo v této situaci nutné. Stačilo ve třídě FlexItemBlockBox implementovat rozhraní Comparable a přepsat metodu `compareTo()`, aby se neporovnávaly mezi sebou instance objektů, ale právě hodnoty atributu `flexOrder`. Řazení bylo provedeno vzestupně, tedy prvek s nejmenší hodnotou atributu `flexOrder` bude v listu na indexu 0.

[1] konkrétně se jedná o třídy Box, ElementBox a BlockBox

[2] například flexBasisValue = dec.getLength(getLengthValue("flex-basis"), false, 0, 0, container.mainSize);

[3] ArrayList – generická kolekce objektů z Java frameworku Collections, zjednodušeně se jedná o dynamické pole

6.3 Nastavení rozměrů položek

Pro získání hlavního rozměru je potřeba několika kroků. Nejprve je získaná CSS vlastnost `flex-basis`.

V případě zadání této vlastnosti číselnou jednotkou nebo procenty[4], je ihned její hodnota uložena do atributu `hypotheticalMainSize`. Pokud je tato hodnota příliš malá a obsah by se v tomto směru do položky s tímto rozměrem nevešel, je hodnota upravena dle směru hlavní osy na fixní hodnotu vlastnosti width nebo height (pokud je tato hodnota větší), popřípadě na velikost obsahu.

Pokud je vlastnost `flex-basis` zadána na hodnotu *auto* nebo není zadaná vůbec, tak se taktéž použije fixních hodnot vlastností `width` nebo `height` dle směru hlavní osy. Pokud takto nejsou zadány ani fixní hodnoty, hodnota atributu `hypotheticalMainSize` se nastaví na šířku (nebo výšku) svého obsahu.

Takto vytvořený hlavní rozměr je poté ohraničen minimální a maximální hodnotou dle návrhu v podkapitole 5.3. Tento proces probíhá v metodě `boundFlexBasisByMinAndMax()`. V jejím těle se využívá atributů `min_size` a `max_size` typu Dimension, zděděných ze třídy BlockBox, ze kterých lze tyto hodnoty snadno získat. Pokud nejsou nastaveny, jejich hodnota je -1.

Hlavní rozměr bude ještě dále upravován flex faktory v podkapitole 6.5.

Vedlejší rozměr flex položek je nastaven snadno získáním hodnoty z vlastnosti `width` nebo `height` (případně min-width a min-height) dle směru hlavní osy, dále je upravován až po vykreslení obsahu.

6.4 Vytvoření flex linek

Pro snadnější úpravu hlavního rozměru položek flex faktory bylo pro horizontální kontejner výhodné implementovat jejich rozdělení do flex linek již po částečném nastavení hlavního rozměru. Pro vertikální kontejner jsou linky vytvářeny až po vykreslení obsahu, u obou variant je však tato fáze implementována stejně. Rozdělení do linek je důležité hlavně pro pozdější proces zarovnání.

Byla vytvořena abstraktní třída `FlexLine`, ze které dědí třídy `FlexLineRow` a `FlexLineColumn`. O celkové vytvoření flex linek se stará metoda `createAndFillLines()`, která nejprve vytvoří první linku a vloží ji do vytvořeného ArrayListu pro flex linky. Následně tato metoda prochází seřazené položky v ArrayListu a zavolá v abstraktní třídě implementovánou metodu `registerItem()`, která přiřadí předanou položku do linky, popřípadě indikuje překročení svého zbylého prostoru. To je následně vyřešeno vytvořením nové linky a umístěním položky do ní. Linky jsou vytvářeny na základě směru kontejneru, tedy buď je vytvořena instance třídy `FlexLineRow` nebo `FlexLineColumn`, přičemž každá má poté různý styl zarovnání implementovaný uvnitř tříd.

Linkám je po vykreslení a úpravě velikostí položek nastaven vedlejší rozměr dle největšího vedlejšího rozměru položky v této lince, pro potřeby zarovnání pomocí vlastností `align-items` a `align-self`. Lince v single-line kontejneru je její vedlejší rozměr nastaven na vedlejší rozměr kontejneru.

[4]pokud je flex-basis zadána procenty, kontejner je v horizontálním směru a není zadán hlavní rozměr kontejneru, tato hodnota se nepoužije (jakoby nebyla zadána)

6.5 Aplikování flex faktorů

Hlavní rozměr může být také modifikován nastavenými flex faktory, které ho ovlivňují vůči hlavnímu rozměru kontejneru. Neboť již jsou položky umístěny ve flex linkách (u obou typů kontejneru), je snadné zjistit zbývající volný prostor v lince.

V případě horizontálního kontejneru je nezbytné flex faktory aplikovat ještě před vykreslením obsahu (viz. podkapitola 6.6), protože na zmíněné vykreslení je potřeba znát šířku položky, která může být flex faktory ovlivňována. Je tedy zvětšována či zmenšována hodnota hypotetického hlavního rozměru v atributu `hypotheticalMainSize`, naopak u vertikálního kontejneru je již nastaven hlavní rozměr přímo ve zděděném atributu `content.height` a je tedy modifikován ten.

Flex-grow

Nejprve byla implementována vlastnost `flex-grow`. Pomocí cyklu byly zjištěny součty hlavních rozměrů (včetně paddingu, marginu a borderu) a flex-grow hodnot v jednotlivých linkách. Zbylý hlavní rozměr kontejneru si mezi sebe mohou rozdělit například jen 2 položky v poměru 3:1, proto byl vypočítán jeden díl tohoto zbylého místa pomocí následujícího jednoduchého vzorce:

$$pieceOfRemainSize = \frac{mainSize - mainSizeOfItemsInLine}{countOfFlexGrowValue} \tag{6.1}$$

Ve vzorci 6.1 figurují tyto proměnné:

- `pieceOfRemainSize`: proměnná, do které se přiřadí vypočtený díl
- `mainSize`: hlavní rozměr kontejneru
- `mainSizeOfItemsInLine`: součet hlavních rozměrů položek v právě modifikované lince
- `countOfFlexGrowValue`: součet flex-grow hodnot položek v právě modifikované lince

Takto získaný díl byl pouze vynásoben hodnotou flex grow každé položky a k hlavnímu rozměru této položky přičten, pokud však nebyl záporný. Díl se zápornou hodnotou by znamenal, že položky v této lince překračují hlavní rozměr kontejneru, což je ošetřeno jednoduchou podmínkou a případným předčasným ukončením metody.

Tyto hodnoty jsou poté ohraničeny maximálním rozměrem v hlavní ose, přičemž bylo opět využito atributu `max_size`. Pokud by některá z položek skutečně tuto hodnotu překročila, vzniklo by v kontejneru opět volné místo, které je nutné rozdistribuovat stejně jako před aplikováním této vlastnosti.

Flex-shrink

Podpora vlastnosti `flex-shrink` byla implementována víceméně stejně jako vlastnost flex-grow. Opět byly pomocí cyklu zjištěny součty hlavních rozměrů položek, a jejich hodnot vlastnosti flex-shrink, v jednotlivých linkách. Výpočet jednoho dílu z místa, které přesahuje hlavní rozměr kontejneru byl získán následujícím lehce pozměněným vzorcem oproti vzorci 6.1:

$$pieceOfOverflowSize = \frac{mainSizeOfItemsInLine - mainSize}{countOfFlexShrinkValue} \tag{6.2}$$

Ve vzorci 6.2 se vyskytují stejné proměnné jako ve vzorci 6.1, samozřejmě s rozdílem, že ve jmenovateli zlomku je součet flex-shrink hodnot položek v právě modifikované lince.

Takto získaný díl byl taktéž vynásoben hodnotou vlastnosti flex shrink každé položky a od hlavního rozměru této položky tentokrát odečten, pokud však nebyl záporný. Díl se zápornou hodnotou by znamenal, že položky v této lince nepřekračují hlavní rozměr kontejneru, což je ošetřeno stejně jako v metodě pro vlastnost flex-grow.

Tyto hodnoty jsou následně ohraničeny minimálním rozměrem v hlavní ose, přičemž bylo využito atributu `min_size`. Pokud by některá z položek skutečně tuto hodnotu překročila, položky by v této lince kontejner znovu přesahovaly, a bylo by tedy nutné rozdistribuovat toto záporné místo stejným algoritmem znovu.

6.6 Vykreslení obsahu položek

Pro vykreslení obsahu položek je použito metod `layoutBlock()` a `layoutInline()` z nadřazené třídy BlockBox na základě atributu `contblock` této třídy, udávajícím zda tato položka obsahuje blokové elementy. Těmto metodám se nejdříve musí nastavit šířka určená pro obsah do atributu `availwidth`. Pro horizontální kontejner je položkám do tohoto atributu nastavena hodnota jejich `hypotheticalMainSize`, v případě vertikálního – `content.width`, zvětšená o velikost levých a pravých hodnot marginu, paddingu a borderu. Taktéž se tento atribut musel změnit ve třídě BlockBoxLayoutManager při vykreslení bloků uvnitř položek. V té byly nastaveny stejné hodnoty jako položkám, ovšem bez marginu, paddingu a borderu.

Po tomto provedeném vykreslení musí být položkám upraven pravý margin, který je v BlockBoxu nastaven až do pravého kraje svého nadřazeného boxu, což je pro Flexbox nežádoucí. To je provedeno v metodě `fixRightMargin()`, ve které je pouze znovu získán ze stylových předpisů pomocí jStyleParseru a nastaven. Položkám jsou nakonec pouze upraveny výšky a šířky v atributech `bounds` a `content` dle dřívěji získaných rozměrů.

6.7 Zarovnání a napozicování flex linek a flex položek

Při implementaci zarovnání bylo pro oba směry kontejneru postupováno přesně dle návrhu. Bylo využito rozfázování zarovnání dle vlastností, čímž vznikl přehlednější kód. Byla tedy prvně určena velikost vedlejšího rozměru linek dle vlastnosti `align-content`, dále implementována podpora vlastností `align-items` a `align-self`, tedy nastavení pozice položek ve vedlejší ose, přičemž pozice v hlavní ose byla prozatím nastavena na nulu. Ta byla poté změněna dle hodnoty vlastnosti `justify-content`. Souběžné zpracování těchto vlastností by sice mohlo přinést kratší kód, ale rozhodně by ho znepřehlednil.

Vlastnost align-content

Nejprve se nastavila pozice a rozměr flex linkám dle vlastnosti `align-content`. K tomu bylo nejdříve potřeba každé lince nastavit souřadnici ve vedlejší ose (y-ová v horizontálním, x-ová ve vertikálním). Ta byla získána jednoduše sečtením vedlejších rozměrů linek předešlých. Na základě vedlejších rozměrů linek a kontejneru je následně dopočítáno zbylé místo.

Nyní již nic nebrání upravení souřadnice ve vedlejší ose dle nastavené hodnoty této vlastnosti. Pro hodnotu *stretch* (roztažení) bylo zapotřebí si ukládat vedlejší rozměr před jeho nastavením, protože jeho změnění negativně ovlivnilo součet vedlejších rozměrů předešlých linek, a tedy zbylé místo nebylo rozkládáno rovnoměrně. Taktéž bylo implementováno správné pozicování pro hodnoty *space-between* a *space-around* v single-line kontejneru, které je odlišné od jejich běžného pozicování.

V případě nulového zbylého místa single-line kontejneru, nebo pokud není nastaven vedlejší rozměr v horizontálním kontejneru, `align-content` nemá žádný vliv na pozici linek.

Vlastnosti align-items a align-self

Dále bylo v rámci implementace řešeno zarovnání položek ve vedlejším rozměru linek podle vlastností `align-items` a `align-self`. Implementace této části byla poměrně obtížná, hlavně kvůli některým specifickým hodnotám těchto vlastností a také kvůli hodnotě *wrap-reverse* vlastnosti `flex-direction`, která efekt některých hodnot výrazně změnila – nestačilo tedy pouze otočit pořadí linek v ArrayListu pomocí metody Collections.reverse(), ikdyž to také usnadnilo vyřešení původního problému. Při nastavování pozic položek v lince pomocí metody `setPosition(x, y)` byla zatím souřadnice hlavního rozměru nastavována na 0, přičemž v další fázi bude tato hodnota měněna při aplikování vlastnosti `justify-content`. Změna vedlejšího rozměru v lince může ovlivnit i předchozí, již nastavené položky (hodnotami *stretch* a *baseline*), proto když se tak stane, je nutné znovu nastavit i relevantní předchozí položky. Zjednodušené chování má proto implementované vždy první položka v každé lince v rámci optimalizace. Pro ní je nejprve zjištěno, zda nemá zadanou hodnotu vlastnosti `align-self`, neboť ta má před vlastností `align-items` přednost. Pro tyto dvě vlastnosti jsou souřadnice vedlejšího rozměru nastavovány zcela stejně, až na hodnotu *inherit*, která je dostupná pouze pro `align-self` a získává hodnotu ze stylů kontejneru.

Hodnota *baseline* je velmi specifická. Značí zarovnání dle obsahu, např. u textu dle spodní hranice prvního řádku. K její implementaci dopomohla metoda `getBaselineOffset()` z třídy `VisualContext`[5]. Dokonce mohla v některých případech i zvětšit vedlejší rozměr linky, a tím případně i vedlejší rozměr kontejneru. Takto se tato hodnota však projevuje pouze v horizontálním kontejneru, pro vertikální se tato hodnota chová stejně jako *flex-start*.

Vlastnost justify-content

Jako poslední typ zarovnání Flexboxu byla aplikovaná vlastnost `justify-content`, která určuje pozici položek v hlavním rozměru. Bylo samozřejmě postupováno po linkách, které jsou opět předány ve struktuře ArrayList. Nejprve byl vypočítán celkový hlavní rozměr položek v lince. Pokud byl větší nebo rovný hlavnímu rozměru kontejneru, nešlo proces tohoto typu rozložení přeskočit, jako u některých předchozích vlastností, neboť má efekt i v tomto případě. Dále bylo rozhodnuto na základě vlastnosti `flex-direction`, zda se jedná o kontejner s otočenými osami (hodnoty *row-reverse* a *column-reverse*) či nikoliv. Dle této vlastnosti byly položky v této lince zarovnávány od konce k začátku nebo opačně. Poté již byla každé položce nastavena souřadnice dle hodnoty vlastnosti `justify-content`, ovšem stále se u všech hodnot, kromě hodnoty *center*, muselo zvažovat zda má či nemá kontejner otočené osy. V těchto méně používaných variantách kontejneru, ve kterých má otočené osy, bylo pro nastavení správné pozice využito hlavního rozměru kontejneru. Výpis 6.1 ukazuje, jak byla nastavovaná pozice pro hodnotu *flex-end* v horizontálním kontejneru.

[5] VisualContext – tato třída se stará o vizuální stránku boxu

```
if (justifyContent == JUSTIFY_CONTENT_FLEX_END) {
        if (isDirectionReversed())
                item.setPosition(widthOfItems, item.bounds.y);
        else
                item.setPosition(mainSize - widthOfPreviousItems -
                                item.bounds.width , item.bounds.y);

} else if (justifyContent == JUSTIFY_CONTENT_CENTER) {
        ...
```

Výpis 6.1: Část kódu nastavující pozici položky dle CSS vlastnosti `justify-content`

Taktéž byl při testování zjištěn a následně ošetřen spíše výjimečný stav, kdy flex linka má pouze jednu položku. Tehdy se hodnota *space-around* chová stejně jako hodnota *center* a hodnota *space-between* jako *flex-start*, ostatní hodnoty mají běžné chování.

6.8 Nastavení rozměrů kontejneru

Posledním krokem implementace bylo zajistit, aby se kontejner správně zobrazoval v kontextu stránky. Bylo tedy nutné po vykreslení položek nastavit jeho rozměry. To bylo provedeno nastavením jeho atributů `bounds.width`, `bounds.height`, `content.width` a `content.height`. Podle směru kontejneru jim byl přiřazen hlavní nebo vedlejší rozměr, jenž byl ještě zvětšen o margin, padding a border kontejneru.

6.9 Co nebylo implementováno

V rámci této práce byla pro knihovnu CSSBox implementována takřka kompletní podpora rozložení pomocí flexibilních boxů. Všechny CSS vlastnosti, popsané v předešlých kapitolách, byly nejprve implementovány pro horizontální kontejner a až poté pro vertikální. Omezenou funkčnost mají CSS vlastnosti `flex-grow` a `flex-shrink` – pokud se některá z položek zmenšila (zvětšila) na své minimum (maximum), již se nepodařilo rozdělit zbylý (nedostatečný) prostor, který nezabrala tato položka.

Taktéž nebylo implementováno chování hodnoty *baseline* CSS vlastností `align-self` a `align-content`, pokud položka nedisponuje žádným textovým obsahem. V tomto případě je ale nesmyslné tuto hodnotu použít.

U kontejneru se nepodařila implementovat CSS vlastnost `float` a u položek není podporována hodnota *auto* vlastnosti `margin`. Z testování také vyplynulo, že se správně nezobrazují obrázky, pokud mají být flex položkami.

6.10 Další možnosti rozšíření

Bylo by vhodné na tuto práci navázat a rozšířit knihovnu CSSBox i o podporu inline-flex kontejnerů, které nebyly součástí zadání této práce. Taktéž je ji třeba rozšířit o podporu CSS vlastnosti `float` u flex kontejnerů, která se stále ještě využívá. Pro flexibilní rozložení by bylo vhodné implementovat do knihovny CSSBox podporu CSS vlastnosti `writing-mode`, která může ovlivnit směr flex kontejneru.

Kapitola 7

Testování

V této kapitole je nejprve popsána oficiální testovací sada pro rozložení Flexbox, dále její použití v této práci a následně popis alternativního testování. Dále jsou popsány vlastní testovací webové stránky a závěrem zhodnoceny výsledky testování a implementace.

7.1 Testovací sada

Pro testování podpory rozložení Flexbox v prohlížečích byla vytvořena oficiální testovací sada dostupná z webových stránek W3C[1], na kterých se nachází i další oficiální testovací sady pro různé CSS moduly. Je velmi rozsáhlá, ovšem pro účely testování tohoto rozšíření do knihovny CSSBox nebyly všechny testy z této sady použitelné z různých důvodů, například kvůli použití hodnoty *inline-flex* vlastnosti `display` nebo z důvodu nepodporování některých použitých vlastností ze strany knihovny CSSBox. Některé testy mají také speciální požadavky jako je například využití JavaScriptu, speciálních fontů či animací.

7.2 Použití testovací sady

Projekt CSSBox je vybaven testovacím rozhraním, které podporuje automatické testování modulů CSS. Pro použití tohoto rozhraní je třeba předat třídě `TestBatch` URL se souborem `reftest-toc.htm`, ve kterém se nachází tabulka s odkazy na všechny testy této sady. Zmíněná oficiální testovací sada tímto souborem však nedisponuje, nemohla tedy být v tomto rozhraní použita. Testovací sada s tímto souborem byla získána ze serveru *github.com*[2], přičemž tato obsahuje stejné testy jako zmíněná testovací sada z webových stránek W3C. Bohužel však byla naposledy aktualizována před dvěma lety, a tak nejsou k dispozici všechny testy a některé jsou vůči oficiální testovací sadě neaktuální.

Třída `TestBatch` zmiňovaný soubor načte a získá z něj odkazy na jednotlivé testy. Poté se v testovací třídě `ReferenceComparisonTest` spouští jeden test za druhým, tyto testy jsou spouštěny v odděleném vlákně s timeoutem. Z těchto testů jsou získávány výsledné hodnoty a porovnávány s referenčními hodnotami. Ty se ještě před spuštěním testů musí nastavit v souboru `test_reference.csv` ve tvaru: *název testu,referenční hodnota*[3].

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

[2] https://github.com/jgraham/css-test-built/tree/master/css-flexbox-1_dev/html

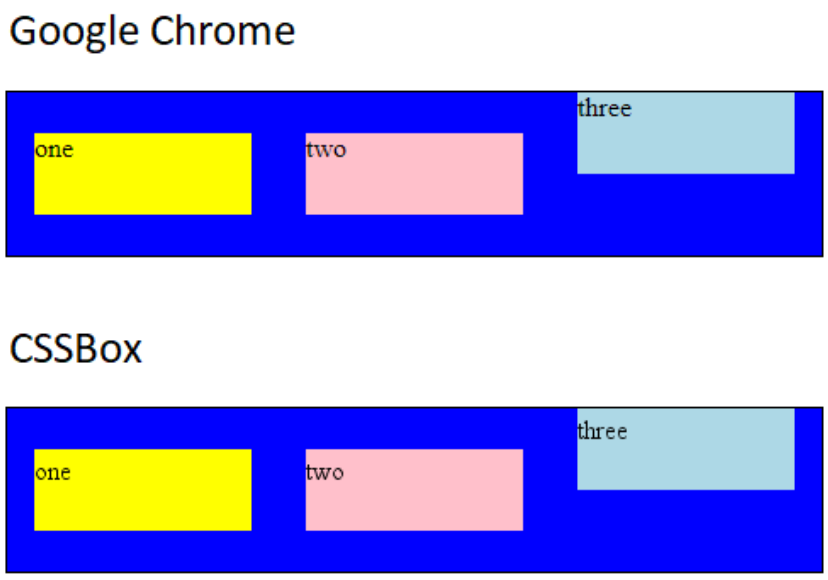
[3] Referenční hodnota je v rozmezí 0.0 až 1.0, přičemž hodnota 0.0 znamená, že test neprojde, pokud se vyskytne sebemenší neshoda. Naopak hodnota 1.0 znamená, že test projde vždy.

Za referenční hodnotu byla zvolena 0.01, tedy maximálně 1% odchylka, a spuštěny testy. Bohužel se po bližším přezkoumání několika výsledků testů s hodnotou 1.0 (tedy zcela špatný výsledek) zjistilo vizuálním porovnáním výsledků s prohlížečem, že tato hodnota nesedí, ba dokonce, že měl test projít s takřka nulovým výsledkem[4]. Také bylo nevhodné, že se musely testovat zcela všechny testy této testovací sady, ve které jsou některé testy na inline-flex kontejner a Grid Layout, které nejsou součástí zadání této práce a knihovna CSSBox zatím nemá podporu těchto modulů. Dalším objeveným negativem bylo, že testy, které během zpracování skončily s chybou, se nepočítaly jako chybné.

Nakonec se od automatického testování upustilo, neboť se neprokázalo jako vhodné k použití, hlavně kvůli výše zmíněným důvodům.

Alternativní způsob testování

Jako alternativa k automatickému testování bylo vybráno vizuální kontrolování výsledků testů vůči výsledkům prohlížeče Google Chrome (verze 73). Na tento proces byla použita oficiální testovací sada popsaná v podkapitole 7.1, což znamenalo výhodu oproti předchozímu přístupu, který musel používat zastaralou verzi této sady. Pro zobrazení vykreslené stránky bylo využito demo aplikace projektu CSSBox s názvem `BoxBrowser`, který zobrazuje HTML elementy a text takřka stejně jako běžný prohlížeč. Tato aplikace byla používána již od rané fáze implementace. Obrázek 7.1 ukazuje ukázku porovnání vykreslení zmíněného prohlížeče a projektu CSSBox.



Obrázek 7.1: Porovnání vykreslení jednoho testu z oficiální testovací sady pro Flexbox (konkrétně test s názvem flexbox_align-self-baseline) – nejprve pomocí prohlížeče Google Chrome, poté pomocí zobrazovacího stroje CSSBox.

[4] například testy s názvem flex-001 a flex-003, dostupné na přiloženém CD

Bohužel i tato testovací sada nebyla zcela vhodná. Obsahuje totiž několik testů se zastaralými CSS vlastnostmi[5] nebo jejich hodnotami[6]. Také se v této sadě nachází mnoho testů na podporu inline-flex kontejneru, vlastnosti `writing-mode` a některé i na Grid Layout. Z těchto důvodů bylo pro testování využito pouze 452 testů z celkových 613. Některé testy byly ručně upravovány a až poté použity k testování. Těmi byly například testy s CSS vlastností `linear-gradient`, `float` u flex kontejneru, popřípadě testy s inline-flex kontejnery, které testovaly jiné CSS vlastnosti.

Tento styl testování nebyl sice tak rychlý jako automatické testování, popsané v předchozí podkapitole, ovšem díky vizuálnímu porovnávání a použití nástroje Chrome DevTools[7] v prohlížeči bylo snadné nalézt a opravit implementační chyby.

7.3 Vlastní testovací webové stránky

Pro základní otestování a demonstraci funkčnosti bylo vytvořeno několik vlastních testovacích webových stránek. Ty testují všechny možné hodnoty hlavních vlastností modulu Flexbox ve všech možných směrech flex kontejneru (dle vlastnosti `flex-direction`). Díky těmto testům byla mimo jiné odhalena chyba s hodnotou *flex-start* u vlastností `align-items` a `align-self` v horizontálním kontejneru orientovaném zdola nahoru, při které neodpovídala pozice položek. Ukázka jedné z vlastních testovacích stránek je zobrazena na obrázku 7.2. Všechny vlastní testovací webové stránky jsou k dispozici na přiloženém CD.

7.4 Výsledky testování

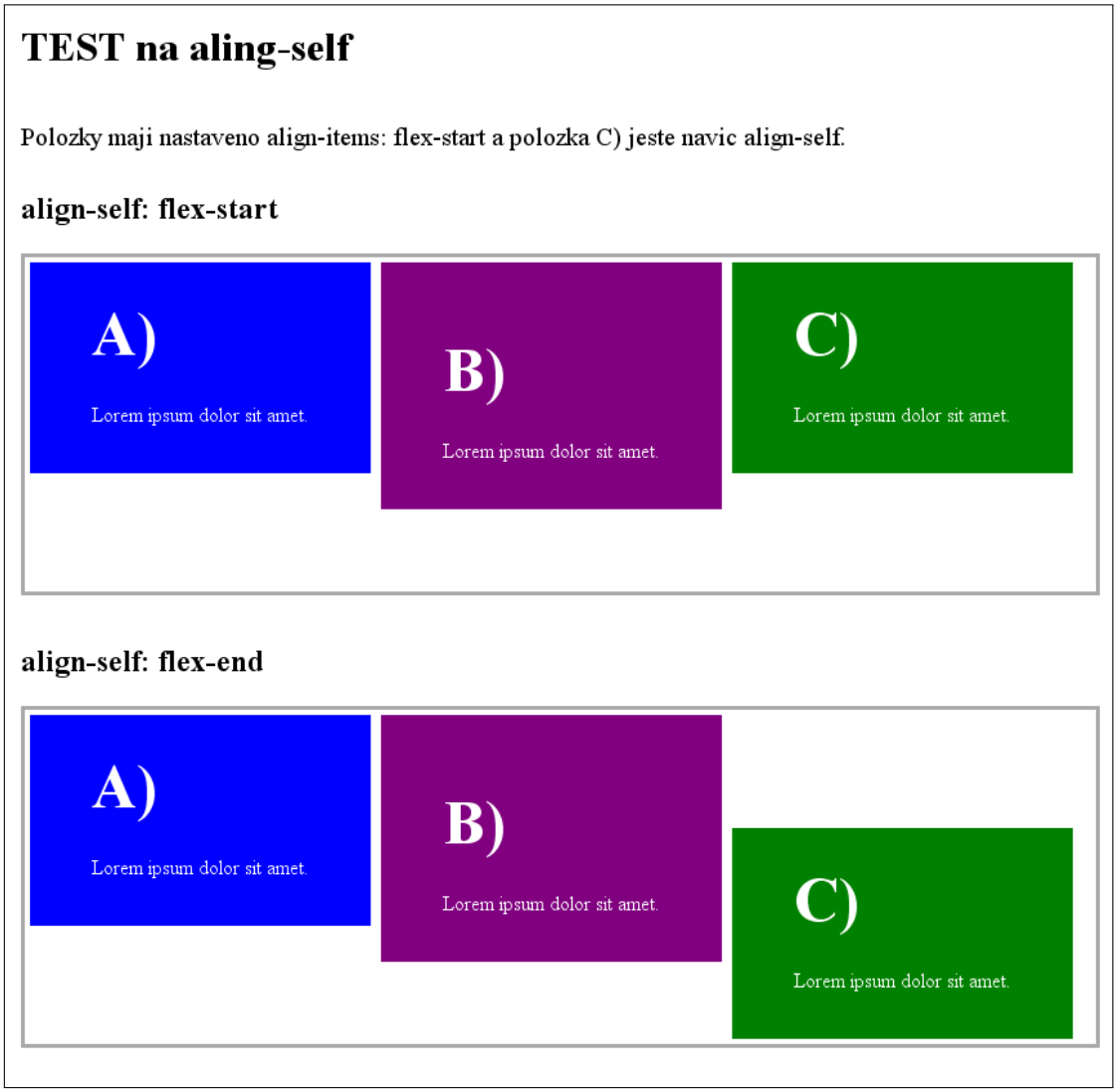
Výsledky vizuálního porovnání testů z oficiální testovací sady (viz. 7.1) byly zaznamenávány do tabulky v programu Microsoft Excel, která je dostupná na přiloženém CD. K těmto výsledkům byly také psány doplňující informace popisující naskytlé chyby, jak byly testy upraveny apod. Vlastní testovací webové stránky do těchto tabulek zahrnuty nebyly.

Na základě testování byla odhalena a vyřešena spousta chyb. Byly doladěny veškeré možnosti hodnot pro vlastnost `flex-basis`. Taktéž se díky testování ukázalo, že pokud má flex kontejner pouze jednu flex položku, některé hodnoty vlastností `align-self` a `align-items` se chovají odlišně. Toto chování bylo opraveno a je demonstrováno ve vlastní testovací sadě. Testy rovněž ukázaly mnoho malých vad, týkajících se zarovnávacích vlastností, které byly ihned odladěny.

Také se objevilo několik nedostatků knihovny jStyleParser, které byly oznámeny vedoucímu projektu. Jedním z nich bylo nepodporování záporných hodnot u vlastnosti `order`.

Celkový poměr úspěšných testů ku neúspěšným je `377:74`, tedy úspěšně prošlých testů bylo celkově `83.59%`. Tento výsledek je negativně ovlivněn tím, že v mnoha testech byly použity obrázky jakožto flex položky, pro které se nepovedlo implementovat podporu.

[5] například vlastnost `flow-into`
[6] například *end* u vlastnosti `align-content`
[7] https://developers.google.com/web/tools/chrome-devtools/



Obrázek 7.2: Část vlastní testovací webové stránky vykreslené pomocí knihovny CSSBox v demo aplikaci s názvem BoxBrowser.

Kapitola 8

Závěr

Cílem této práce bylo seznámit se s projektem zobrazovacího stroje CSSBox a jeho architekturou, poté prostudovat nové rozložení obsahu s názvem Flexbox a projekt CSSBox o tento typ rozložení rozšířit.

Nejprve byla pečlivě prostudována dokumentace projektu CSSBox a specifikace modulu Flexbox. Před implementací tohoto rozložení však byla ještě provedena úprava stávající struktury. Byl navržen a implementován systém layout managerů, díky kterému lze boxům přiřadit i jiný typ rozložení než mají výchozí. Po ní již následovala implementace rozložení Flexbox, pro který byla vytvořena a pomocí testování doladěna, téměř kompletní podpora.

Deklarovaných cílů bylo tedy dosaženo a rozšíření Flexbox bude po menších úpravách možné zařadit do projektu CSSBox. Jako další postup by bylo vhodné rozšířit projekt CSSBox o podporu inline-flex kontejnerů. Dalším rozšířením by mohla být lepší podpora jazyka JavaScript ve zpracovávaných webových dokumentech, což však bude obtížný úkol hlavně z hlediska údržby, neboť tento jazyk se neustále vyvíjí.

Literatura

[1] alattalatta: JavaScript technologies overview. [Online; navštíveno 3.4.2019]. URL https://developer.mozilla.org/en-US/docs/Web/JavaScript/JavaScript_technologies_overview

[2] Burget, R.: CSSBox Manual. [Online; navštíveno 16.4.2019]. URL http://cssbox.sourceforge.net/manual/

[3] Burget, R.: Knihovna jStyleParser. [Online; navštíveno 18.4.2019]. URL http://www.fit.vutbr.cz/~burgetr/prods.php?id=63

[4] CoreLangs: CSS Versions. [Online; navštíveno 8.4.2019]. URL http://www.corelangs.com/css/basics/versions.html

[5] Croft, J.; Llyod, I.; Rubin, D.: *Mistrovství v CSS*. Computer Press, 2007, ISBN 978-80-251-1705-7.

[6] Lazaris, L.: *CSS okamžitě*. Albatros Media, 2014, ISBN 978-80-251-4176-2.

[7] Meyer, E. A.; Weyl, E.: *CSS The Definitive Guide*. O'Reilly media, 2018, ISBN 978-1-449-39319-9.

[8] Michálek, M.: *Vzhůru do CSS3*. Michálek Martin - Vzhůru dolů, 2017, ISBN 978-80-260-8438-9.

[9] Novák, O.: *Implementace mřížkového rozložení ve zobrazovacím stroji CSS*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta informačních technologií, Brno, 2019.

[10] Pettit, N.: Which Layout? Static, Liquid, Adaptive, or Responsive. [Online; navštíveno 9.4.2019]. URL https://blog.teamtreehouse.com/which-page-layout

[11] Snížek, M.: *CSS pro zelenáče*. Neocortex, 2004, ISBN 80-86330-14-1.

[12] Tvorba-Webu.cz: DOM: Document Object Model. [Online; navštíveno 3.4.2019]. URL https://www.tvorba-webu.cz/dom/

[13] W3C: CSS Flexible Box Layout Module Level 1. [Online; navštíveno 3.4.2019]. URL https://www.w3.org/TR/css-flexbox-1/

[14] W3Schools: HTML Block and Inline Elements. [Online; navštíveno 26.4.2019]. URL https://www.w3schools.com/html/html_blocks.asp

[15] W3Schools: HTML Introduction. [Online; navštíveno 1.4.2019].
URL https://www.w3schools.com/html/html_intro.asp

Příloha A

Obsah CD

Na přiloženém CD se nacházejí následující soubory a adresáře:

- `CSSBox` – zdrojové kódy knihovny CSSBox včetně vlastních kódů rozšiřující tuto knihovnu o podporu flexibilního rozložení
- `xondra49-Flexbox.pdf` – technická zpráva ve formátu PDF
- `xondra49-diff.txt` – textový soubor (diff) demonstrující změněné a nově vytvořené soubory knihovny CSSBox
- `README` – readme soubor
- `own-tests` – adresář s vlastními testy
- `test-set-results.xlsx` – tabulka zobrazující výsledky testování
- `doc-codes` – adresář se zdrojovými kódy technické zprávy

Příloha B

Manuál pro zprovoznění projektu

Implementace tohoto rozšíření byla tvořena ve vývojovém prostředí IntelliJ IDEA, proto tento manuál bude orientován na zprovoznění projektu v tomto prostředí, které se již očekává nainstalované a připravené. V následujícím seznamu je popsán krok po kroku postup pro zprovoznění projektu:

1. naklonovat zdrojové kódy knihoven CSSBox a jStyleParser ze serveru Github (knihovna CSSBox je dostupná i na přiloženém CD) pomocí následujících příkazů:

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

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

2. importovat projekt do prostředí IntelliJ IDEA pomocí Apache Maven souboru s názvem `pom.xml` z naklonované knihovny CSSBox

3. v IntelliJ IDEA kliknout na záložku Maven Projects a přidat Maven projekt pomocí tlačítka `Add Maven Projects` použitím souboru s názvem `pom.xml` z naklonované knihovny jStyleParser

4. ve stejné záložce vygenerovat zdrojové kódy tlačítkem `Generate Sources And Update Folders For All Projects`

5. funkčnost lze zkontrolovat puštěním některé z demo aplikací