Česká zemědělská univerzita vPraze

Provozně ekonomická fakulta

Katedra informačních technologií

Bakalářská práce

Flexbox a grid layout

Ivan Barsukov

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Ivan Barsukov

Informatika

Název práce

Flexbox a grid layout

Název anglicky

Flexbox and grid layout

Cíle práce

Bakalářská práce je tematicky zaměřena na problematiku flexboxu a grid layoutu. Hlavním cílem práce je na základě analýzy a modelových příkladů demonstrovat vhodnost použití dané technologie. Dílčí cíle práce jsou:
- charakteristika principu flexboxu a grid layout včetbě důvodu jejich vývoje a alternativ,
- analýza podpory v prohlížečích,
- souhrn vývoje formátování webových stránek a aplikací.

Metodika

Metodika řešené problematiky bakalářské práce je založena na studiu a analýze odborných informačních zdrojů. Vlastní práce spočívá ve vytvoření modelových příkladů za použití flexboxu a grid layoutu a na jejich základě demonstrovat a analyzovat vhodnost použití zkoumané technologie. Na základě syntézy teoretických poznatků a výsledků praktické části budou formulovány závěry bakalářské práce.

Doporučený rozsah práce

30-40 stran

Klíčová slova

flexbox, grid layout, webová stránka, CSS, WWW

Doporučené zdroje informací

A Complete Guide to Flexbox | CSS-Tricks. CSS-Tricks [online]. Dostupné z: https://css-tricks.com/snippets/css/a-guide-to-flexbox/

David Sawyer McFarland. CSS: The Missing Manual, 4th Edition. O'Reilly Media, 2015. ISBN-13: 978-1491918050

Eric A. Meyer. CSS Floating: Floats and Float Shapes. O'Reilly Media, 2016. ISBN-13: 978-1491929643

Eric A. Meyer. Grid Layout in CSS: Interface Layout for the Web. O'Reilly Media, 2016. ISBN-13: 978-1491930212

Estelle Weyl. Flexible Boxes in CSS: Free Yourself with Flexbox. O'Reilly Media, 2017. ISBN-13: 978-1491930045

World Wide Web Consortium (W3C). World Wide Web Consortium (W3C) [online]. Dostupné z: https://www.w3.org/

Předběžný termín obhajoby

2017/18 LS – PEF

Vedoucí práce

Ing. Pavel Šimek, Ph.D.

Garantující pracoviště

Katedra informačních technologií

Elektronicky schváleno dne 30. 10. 2017

Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 1. 11. 2017

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 04. 03. 2018

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Flexbox a grid layout" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne datum odevzdání _____

Poděkování

Rád bych touto cestou poděkoval Ing. Pavlu Šimkovi, Ph.D. za jeho cenné rady, připomínky, odborné konzultace a čas, který mi věnoval během přípravy této bakalářské práce.

Flexbox a grid layout

Abstrakt

Tato bakalářská práce se zabývá novými technologiemi CSS3, možnostmi jejich využití při tvorbě webových aplikací a je zaměřena na problematiku vývoje webových layoutů. Práce je rozdělena na teoretickou a praktickou část.

Teoretická část obsahuje dílčí kapitoly, které popisují historický vývoj webových layoutů, definice a principy fungování a popis stávajících standardů.

Praktická část je zaměřena na komparaci dvou nových modulů CSS, představuje analýzu silných a slabých stránek a popisuje problematiku podpory webových prohlížečů. Dále jsou zpracovány webové stránky, kde jsou aplikovány nové vlastnosti CSS3 s ohledem na osvědčené postupy. Použité vlastnosti jsou popsány v teoretické části práce.

Klíčová slova: flexbox, grid layout, webová stránka, CSS, WWW, layout

Flexbox and grid layout

Abstract

This bachelor thesis deals with new CSS3 technologies, opportunities for their use in web application development and focused on problems of web layout development. The thesis is divided into the theoretical and practical part.

The theoretical part contains partial chapters describing the historical development of web layouts, definitions and principles of operation, and description of existing standards.

The practical part is focused on the comparison of two new CSS modules, analyzes the strengths and weaknesses and describes the issue of support for web browsers. Additionally, web pages where new CSS3 features are applied to best practices. The used properties are described in the theoretical part of the thesis.

Keywords: flexbox, grid layout, web page, CSS, WWW, layout

Obsah

1. Úvod

Od samého počátku existence webu bylo vytvoření rozložení stránky jedním z hlavních problémů webových vývojářů. Dobrý layout je základem použitelnosti každé webové stránky. Přitom návrh rozvržení není nic jednoduchého, zejména u větších webů. První stránky neměly žádný layout, jednalo se pouze o prostý text. S rozvojem internetu se objevila nutnost nějakým způsobem zachovat a strukturovat jednotlivé prvky na stránce. První grafické rozvržení bylo sestaveno pomocí tabulkového layoutu. Tento způsob měl v sobě mnoho nedostatků: tabulka se nezobrazovala plně, dokud nebyl načten celý obsah, kód vypadal těžkopádné. V tuto chvíli prohlížeče začaly podporovat jazyk CSS, takže tabulky nahradily plovoucí prvky. Ale nepodařilo se okamžitě přejít na novou technologii. Hlavní problém je v tom, že ne všichni uživatelé internetu aktualizují svůj software. Tento problém je aktuální i dnes a dělá hodně starostí webovým vývojářům, které zákazník nutí podporovat starší verze prohlížečů. S příchodem CSS rozvržení stránky začali dělat pomocí plovoucích prvků. Tento způsob žije a je do dneška používán skoro všude. Ale hlavní problém je v tom, že tento způsob původně nebyl vytvořen pro tento účel. Tato metoda byla určena pro obtékání obrázků textem, proto to tím pádem nesplňuje všechny potřeby vývojářů při vytváření layoutu.

Účastníci konsorcia W3C chápali tento nedostatek. Proto byl v roce 2009 vydán první návrh modulu flexbox, který umožnil vytvoření flexibilních bloků, které se mohly přizpůsobovat v závislosti na okolním prostředí. Prohlížeče začaly podporovat flexbox poprvé přibližně v roce 2012. Práce trvala velmi dlouho, vývojáři několikrát měnili strukturu tohoto modulu.

Souběžně s tím začal vývoj grid layoutu. Jednalo se o první modul, který byl vynalezen záměrně na vytvoření rozvržení. Jeho podpora se objevila ve většině prohlížečů až v roce 2017. Jeho tvůrci se snažili vzít v úvahu všechny existující nedostatky. Tento modul přinesl ještě více flexibility a možností. Hlavní výhodou je, že je velmi snadné se ho naučit, i za přítomnosti velkého množství CSS vlastností.

2. Cíl práce a metodika

2.1. Cíl práce

Bakalářská práce je tematicky zaměřena na problematiku flexboxu a grid layoutu. Hlavním cílem práce je na základě analýzy a modelových příkladů demonstrovat vhodnost použití dané technologie. Dílčí cíle práce jsou:

- charakteristika principu flexboxu a grid layout včetně důvodu jejich vývoje a alternativ,
- analýza podpory v prohlížečích,
- souhrn vývoje formátování webových stránek a aplikací.

2.2. Metodika

Metodika řešené problematiky bakalářské práce je založena na studiu a analýze odborných informačních zdrojů. Vlastní práce spočívá ve vytvoření modelových příkladů za použití flexboxu a grid layoutu a na jejich základě demonstrovat a analyzovat vhodnost použití zkoumané technologie. Na základě syntézy teoretických poznatků a výsledků praktické části budou formulovány závěry bakalářské práce.

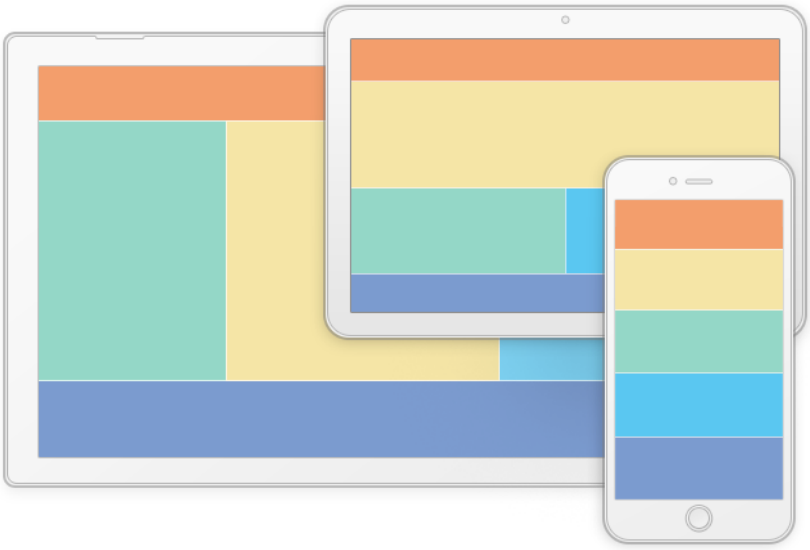
Praktická část je současně zaměřená na responzivní design a znovupoužitelnost kódu, která je založená na zachovaní co nejnižší specifičnosti selektorů. V neposlední řadě se také věnuje sémantické straně, která zvýší přístupnost webu pro vyhledávací roboty a hlasové čtečky.

Závěr této práce hodnotí rozdíl dvou nových modulů a ukazuje možnosti jejich použití na skutečných případech.

3. Teoretická východiska

3.1. Layout

Layout, čili grafické rozvržení, rozložení nebo schéma stránky a jejich jednotlivých prvků na ni. Termín se používá jak pro webové stránky, tak i pro tiskové stránky. [7] Layout udržuje logickou organizaci jednotlivých částí stránky a říká, kde bude umístěn logotyp, hlavička, navigace a patička, vyhledávání a další součásti stránky. Dobrý layout je základem použitelnosti každé webové stránky. Přitom navržení rozvržení není nic jednoduchého, zejména u větších webů. [8] Tímto procesem se zabývají weboví designéři a zkušení specialisté na User Experience (UX), [9] kteří by zároveň měli navrhovat přizpůsobení zobrazení obsahu stránek kvůli tomu, že si stránku budou moci prohlížet návštěvníci na širokém spektru velikosti obrazovek. [10][11] Zatímco před pár lety znamenal responzivní design pouze zmenšování rozlišení směrem dolů, dnes už to není tak úplně pravda. [12] Existuje spousta zařízení (např. televize, obrazovky), která mají extrémně vysoká rozlišení. Je proto důležité dbát na to, aby weby vypadaly dobře všude. [12]



Obrázek 3.1 - 1. Responzivní design
Zdroj: https://js.devexpress.com/Documentation/16_1/Guide/UI_Widgets/UI_Widget_Categories/Layout_Widgets/

3.2. Float

Vlastnost float zaujímá zvláštní místo v CSS. Před jeho vznikem bylo umístění různých elementů stránky vedle sebe možné pouze za pomoci tabulek. [13] Float nahradil atribut *align*,

který se už nepoužívá a není validní v HTML5. [3 s.1][18] Tato vlastnost není dědičná a dělá z elementu plovoucí prvek. Tyto elementy jsou vyjmuty z klasického rozložení stránky a jsou neplovoucím obsahem obtékány. [14] Pomocí takového chování lze připravit celé rozvržení webové stránky, a tak se zbavit tabulkového rozvržení, které má spoustu nevýhod. Dále se obtékání používá při plavání obrázků, aby dobře seděly v textu. [15] Například když je nutné zobrazit obrázek z okrajů, a přitom ve volném prostoru na opačné straně zobrazit text. [16] Může nabývat čtyř hodnot [3 s.1-2]:

Zdrojový kód č. 3.2 -1. Vlastnost *float*.

```
.element {
        float: none;
        /*  výchozí hodnota, element neplave */

        float: left;
        /*  element plave po levé straně, neplovoucím obsah je zprava */

        float: right;
        /*  element plave po pravé straně, neplovoucím obsah je zleva */

        float: inherit;
        /* zděděna hodnota rodičovského elementu */
}
```

Element s nastaveným floatem se stane blokovým, i když původně byl řádkovým, proto je zbytečné nastavovat *display: block*. A proto můžeme k takovému prvku použít vlastnosti jako *width*, *height*, *margin* a *padding*. [3 s. 6] Při použití této vlastnosti dojde k následujícímu:

- Prvek se zobrazí jako blok s šířkou podle velikosti svého obsahu.
- Inline elementy posouvají, aby do uvolněného místa vešel floatovaný element, a obtéká ho z druhé strany. Obyčejné bloky se chovají, jako by žádný prvek neexistoval.
- Prvek je vyjmut z toku dokumentu a posouvá se doleva (*float: left*) nebo doprava (*float: right*), dokud se nedotkne okrajů rodiče nebo jiného prvku s vlastností *float*.
- Pokud chybí horizontální prostor pro to, aby prvek vešel, obtékání se očekávaným způsobem neprojeví. Element se posouvá dolů, dokud nezačne zapadat.

Například, když máte několik elementů za sebou se stejnou hodnotou *float*, a změníte ji na opačnou (*left* na *right*, *right* na *left*), tak tím prohodíte pořadí sloupců.

3.2.1. Clear

Představte si, že máte *div*, který má pozadí a uvnitř tohoto *divu* vytvoříte dva sloupce. Zjistíte jednu nepříjemnou skutečnost a to fakt, že se pozadí rodičovského *divu* nezobrazí. [2 s.240] Prohlížeč totiž neví, kam až tyto dva *divy* sahají, protože jsou vyjmuty z běžného toku dokumentu. Budou-li mít všechny elementy nastaveny *float* jinak než na *none*, vyplavou z rodičovského elementu — proti tomu lze použít právě vlastnosti *clear*. Vlastnost *clear* definuje místo zobrazení dalších elementů pod plovoucím prvkem. Může nabývat pěti hodnot: [3 s. 20]

Zdrojový kód č. 3.2.1 - 1. Vlastnost *clear*.

```
.element {
        clear: none;
        /* výchozí hodnota, plovoucí obsah je povolen zleva i zprava */

        clear: left;
        /* zakazano obtekaní elementu s float: left */

        clear: right;
        /* zakazano obtekaní elementu s float: right */

        clear: both;
        /* ukončení obtékání ze všech stran */

        clear: inherit;
        /* zděděna hodnota rodičovského elementu */
}
```

Nejdůležitější je hodnota *both*. Element s takovou hodnotou (a všechny elementy následující) se nebude snažit připlout k předcházejícím s *float: left* nebo *float: right*. [13]

Zdrojový kód č. 3.2.1 - 2. Zrušení obtékání pomocí elementu.

```
<div class="isFloated"></div>
<div class="isFloated"></div>
<div class="isFloated"></div>
<div style="clear: both;"></div>
```

3.2.2. Clearfix

Myšlenka účelného přidaní nového elementu do zdrojového kódu HTML není tak hezká, protože prázdný element nemá žádnou sémantiku a z prvního pohledu není jasné, k čemu je tam ten element. Nebo můžete také potkat zastaralé zrušení obtékání pomocí atributu clear:

Zdrojový kód č. 3.2.2 - 1. Zrušení obtékání pomocí atributu *clear*.

```
<br clear="all">
```

Proto byl vymyšlen clearfix hack.

Zdrojový kód č. 3.2.2 - 2. Kontejner s plovoucími prvky.

```
<div class="container">
  <div class="isFloated"></div>
  <div class="isFloated"></div>
  <div class="isFloated"></div>
</div>
```

Zdrojový kód č. 3.2.2 - 3. Clearfix.

```
.container:before,
.container:after {
  content: "";
  display: table;
}
.container:after {
  clear: both;
}
```

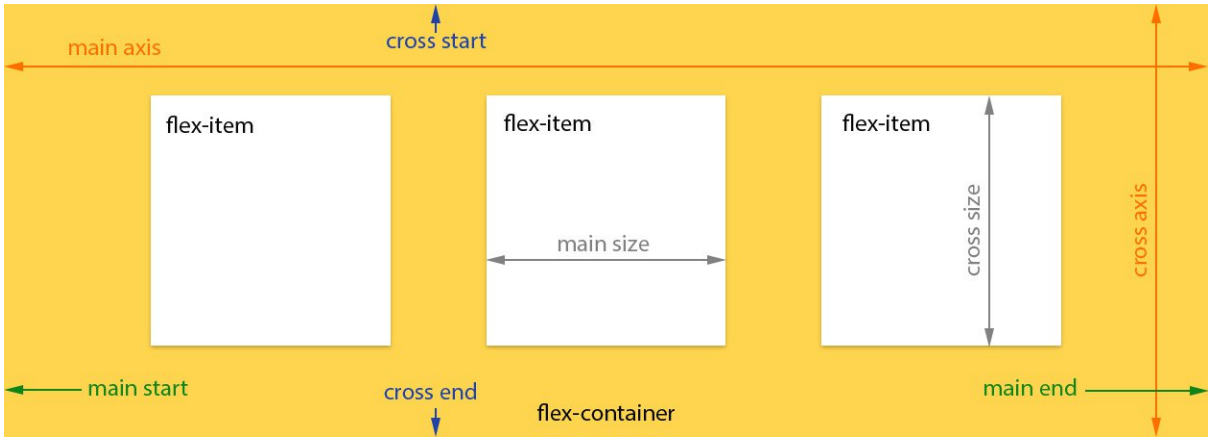
Daný způsob používá CSS frameworky (např. Bootstrap do 4. verze, Foundation do 6. verze), layout kterých je založen na vlastnosti *float*. Funguje to i v IE8, kdybyste potřebovali podporovat prohlížeče nižších verzí, můžete použít velmi populární návod [19] s podrobným popisem veškerých technik.

3.3. Flexbox

Flexbox byl projektován pro vytváření pružných elementů layoutu a uchovává v sobě mnoho jemností. Jednou z hlavních předností flexboxu je totiž schopnost vyplňování zbylých prostorů bez nutnosti přepočítávání Javascriptem. [5 s.17] Než začneme popisovat vlastnosti, seznámíme se základy modelu flexbox. [20] Flex rozložení se skládá z rodičovského kontejneru nazývaného flex kontejner a flex položky, každý přímý potomek kontejneru. Stojí za zmínku, že flex položka může být zároveň flex kontejnerem. Bude se chovat jako flex položka, tj. bude dodržovat pravidla rozmístění a zarovnání svého rodiče, a jako samostatný flex kontejner. [5 s.25] V tomto případě se jeho podřízené elementy první úrovně budou řídit pouze jeho flexibilními pravidly. [1]

Základem flexboxu je myšlenka osy. Flexbox je nástroj dvourozměrného rozložení a používá k práci dvě osy — hlavní (main axis) a příčnou (cross axis). [5 s.10] Hlavní osa je hlavním

směrem pohybu prvků uvnitř kontejneru. [21] Ve výchozím nastavení je tato osa směrována zleva doprava, ale její směr, jak uvidíte později, lze libovolně změnit. Příčná osa je směr pohybu prvků, pokud se nevejdou do kontejneru ve směru hlavní osy. [1]



Obrázek 3.3 - 1. Základy modelu
Zdroj: http://babel.es/es/blog/blog/junio-2016/flexbox-css3-layout

3.3.1. Vlastnost *display*

Model flexbox je spojen s určitou hodnotou CSS vlastnosti *display*. Tato vlastnost je sama o sobě víceúčelová a určuje, jak má být prvek zobrazen v dokumentu. Ve flex modelu může mít tato vlastnost dvě hodnoty: *flex* a *inline-flex*. [20] Jeho první hodnota určuje blokové chování kontejneru, druhá řádkové. Po nastavení jedné ze dvou hodnot vlastností se každý podřízený prvek první úrovně automaticky stává flex položkou, která je zarovnaná v řadě (podél hlavní osy) o sloupec stejné výšky rovnající se výšce bloku kontejneru, a také přestanou fungovat vlastnosti pro podřízené prvky jako *float*, *clear*, *vertical-align* a *column-**. [21]

3.3.2. Hlavní osa, *flex-direction*

V normálním toku dokumentu jsou bloky a text uspořádány zleva doprava a shora dolů. Ve známém blokovém modelu jsou směry „doleva“, „doprava“, „nahoru“ a „dolů“ neměnné. [5 s.36] Ale uvnitř flex kontejneru se tyto pojmy mohou změnit, protože tam lze měnit obvyklý směr toku. [20] Ve výchozím nastavení je hlavní osa směrována zleva doprava, ale může být otočená ve všech směrech pomocí vlastnosti *flex-direction*, která je určená pro flex kontejner. Hodnoty vlastností: [21]

Zdrojový kód č. 3.3.2 - 1. Vlastnost *flex-direction*.

```
.flex-container {
 flex-direction: row;
 /* výchozí hodnota, hlavní osa směřuje zleva doprava */
```

```
  flex-direction: row-reverse;
  /* hlavní osa směřuje zprava doleva */

  flex-direction: column;
  /* hlavní osa směřuje shora dolů */

  flex-direction: column-reverse;
  /* hlavní osa směřuje zdola nahoru */
}
```

Obrázek 3.3.2 - 1. Směr hlavní osy
Zdroj: https://css-tricks.com/snippets/css/a-guide-to-flexbox/

Příčná osa je vždy kolmá hlavní ose a otáčí se spolu s ní. [1] Pokud hlavní osa směřuje vodorovně, pak příčná osa směřuje dolů. Pokud hlavní osa směřuje svisle, pak příčná osa směřuje vpravo. To není úplně logické chování, na které je třeba si zvyknout. Tudíž příčná osa nikdy nesměřuje nahoru nebo doleva. Vlastnost pro otáčení příčné osy neexistuje. [21]

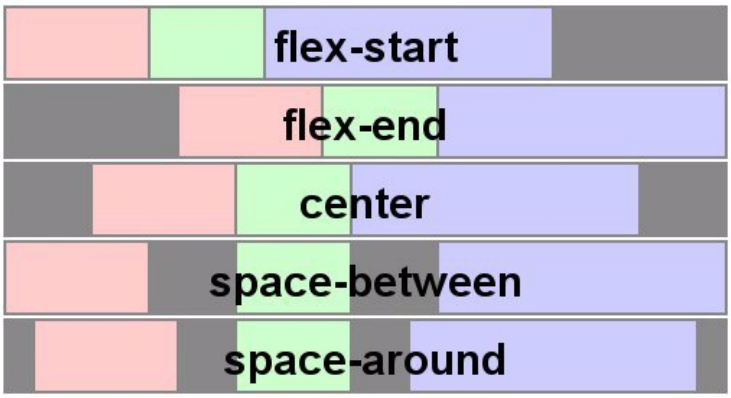
3.3.3. Zarovnání podél hlavní osy, *justify-content*

Pro zarovnání položek na hlavní ose používá vlastnost *justify-content*, která je určená pro flex kontejner. Ve výchozím nastavení jsou prvky umístěny na začátku hlavní osy, ale jejich pozice může být změněna. Hodnoty vlastností: [21]

Zdrojový kód č. 3.3.3 - 1. Vlastnost *justify-content*.

```
.flex-container {
  justify-content: flex-start;
  /* výchozí hodnota, prvky se nacházejí na začátku hlavní osy */

  justify-content: flex-end;
  /* prvky se nacházejí na konci hlavní osy. Všimněte si, že
  justify-content: flex-end nemění pořadí prvků, jak se stane,
  když se změní směr osy na flex-direction: row-reverse.
  Prvky se jednoduše přitlačují na konec hlavní osy */

  justify-content: center;
  /* prvky se nacházejí ve středu hlavní osy */

  justify-content: space-between;
  /* prvky jsou rovnoměrně rozmístěny podél hlavní osy,
  vzdálenosti mezi sousedními prvky jsou stejné,
```

```
  mezi prvky a hranami kontejneru není prostor */

  justify-content: space-around;
  /* jako space-between, pouze mezi prvky a hrany kontejneru
  je mezera rovnající se polovině vzdálenosti mezi sousedními prvky */
}
```



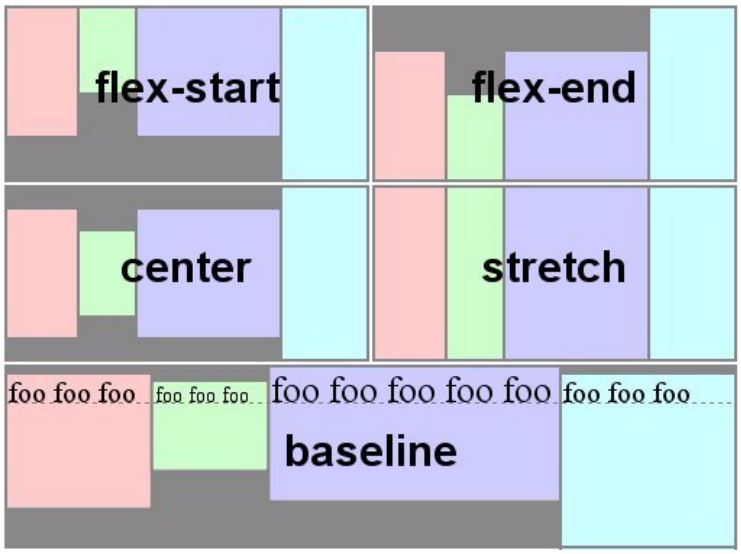
Obrázek 3.3.3 - 1. Hodnoty vlastnosti *justify-content*
Zdroj: https://www.w3.org/TR/css-flexbox/

3.3.4. Zarovnání podél příčné osy, *align-items*

Pro zarovnání položek na příčné ose se používá vlastnost *align-items*, která je určená pro flex kontejner. [20] Ve výchozím nastavení jsou prvky natažené podél směru příčné osy, ale jejich pozice může být změněna. Hodnoty vlastností: [21]

Zdrojový kód č. 3.3.4 - 1. Vlastnost *align-items*.

```
.flex-container {
  align-items: stretch;
  /* výchozí hodnota, prvky jsou natažené tak,
  aby se obsadit veškeré dostupné místo kontejneru podél příčné osy */

  align-items: flex-start;
  /* prvky jsou umístěny na začátku příčné osy */

  align-items: flex-end;
  /* prvky jsou umístěny na konci příčné osy */

  align-items: center;
  /* prvky jsou umístěny uprostřed příčné osy */

  align-items: baseline;
  /* zarovnání na účaří prvního řádku */
}
```



Obrázek 3.3.4 - 1. Hodnoty vlastnosti *align-items*
Zdroj: https://www.w3.org/TR/css-flexbox/

Pokud je flex prvkům nastavena příčná velikost (*cross size*), pak prvky nebudou natažené podél příčné osy, ale budou umístěny na začátku příčné osy. [5 s.40] Používání výrazů „horizontální nebo vodorovné zarovnání“ není správné, protože v závislosti na směru osy za to odpovídají různé vlastnosti. Pří použití *flex-direction: row* za horizontální zarovnání odpovídá *justify-content*, ale pokud otočíte hlavní osu kolmo, pak tuto práci již bude provádět vlastnost *align-items*. [20] Chcete-li se vyhnout takovému zmatku, vždy zdůrazněte, podle které osy děláte zarovnání.

3.3.5. Sobecké zarovnání, *align-self*

Rozložení prvků podél hlavní osy se určuje pro flex kontejner a platí stejně pro všechny flex položky. Nastavit nějakému prvku odlišné rozložení podél hlavní osy od jiných nelze. [5 s.42] A to je celkem logické, protože pak by prvky do sebe narážely. S příčnou osou je vše jednodušší. Lze říci, že každý prvek má vlastní osu, a můžete určovat zarovnání pro každý zvlášť. K tomu se používá vlastnost *align-self*, která se nenastavuje pro flex kontejner, ale pro flex prvek. Vlastnost *align-self* má stejné hodnoty jako *align-items*, s výjimkou hodnoty *auto*, která je zároveň výchozí hodnotou. [21] *Auto* bere rodičovskou hodnotu *align-items* nebo *stretch*, pokud rodič neexistuje. Hodí se pro vytvoření výjimky ze zarovnání.

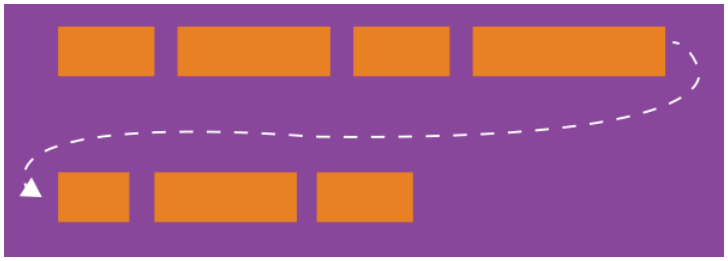
Obrázek 3.3.5 - 1. Zarovnání jednotlivých prvků
Zdroj: http://babel.es/es/blog/blog/junio-2016/flexbox-css3-layout

3.3.6. Zalamování položek, *flex-wrap*

Pokud bude flex položek v kontejneru více, než se může vejít na jeden řádek, pak se budou zmenšovat až na minimální možnou šířku (varianta se zvažuje, když je hlavní osa směrována horizontálně). [1] Dokonce i když šířku je nastavená pevně, mechanismus flexboxu jí může zmenšit. Pokud se do kontejneru nevejdou ani po snížení, pak vyjdou ven za jeho hranice, ale budou i nadále umístěny v jedné řadě. To je podobné, jako chování buněk v tabulce. [1] Toto chování lze změnit vlastností flex kontejneru *flex-wrap*. Pokud je přenos flex položek povolen, pak jsou řady prvků umístěny podél hlavní osy. Hodnoty vlastností: [5 s.33]

Zdrojový kód č. 3.3.6 - 1. Vlastnost *flex-wrap*.

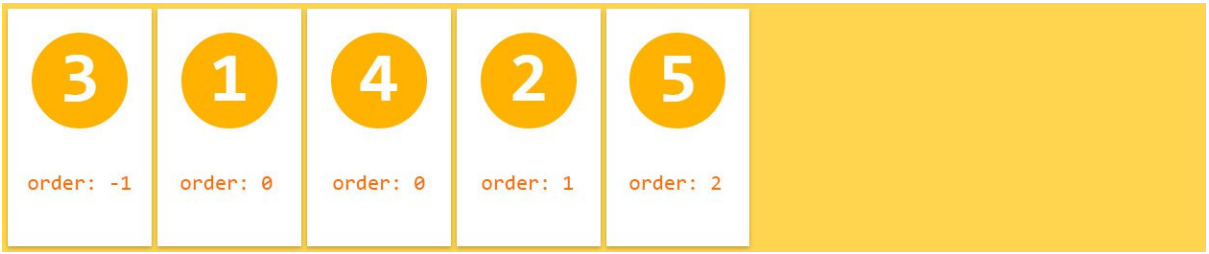
```
.flex-container {
  flex-wrap: nowrap;
  /* výchozí hodnota, zalamování je zakázáno, prvky
  jsou zarovnány podél hlavní osy v jednom řádku */

  flex-wrap: wrap;
  /* prvky jsou uspořádány do několika řádků,
  pokud se nevejdou do kontejneru */

  flex-wrap: wrap-reverse;
  /* stejně jako wrap, pouze řady budou uspořádány v opačném pořadí.
  První na konci příčné osy a poslední na začátku */
}
```



Obrázek 3.3.6 - 1. Zalamování položek
Zdroj: https://css-tricks.com/snippets/css/a-guide-to-flexbox/

3.3.7. Pořadové číslo flex položky, *order*

To je velmi užitečná vlastnost, protože s jeho pomocí je možné měnit pořadí flex položek v toku bez změny HTML kódu. Výchozí hodnota je 0, což znamená dodržení stejného pořadí, jako ve zdrojovém HTML. [21] Když se hodnota liší od výchozí, jsou prvky uspořádány podle vzestupu čísel. Pořadové číslo je dáno celým číslem, kladným nebo záporným. To znamená, že s hodnotou *order* „-1“ bude prvek umístěn jako první, při „1“ jako poslední podél hlavní osy. Pokud dva prvky mají stejnou hodnotu vlastnosti order, prvním podél hlavní osy bude ten prvek, který je vyšší ve zdrojovému kódu. [20]



Obrázek 3.3.7 - 1. Změna pořadí
Zdroj: http://babel.es/es/blog/blog/junio-2016/flexbox-css3-layout

3.3.8. Specefičnost vlastnosti margin

Existuje jedno překvapení: vnější okraje se neslučují ani horizontálně, ani vodorovně. Všechno je v mechanismu distribuce volného místa: [20]

1. naleznou se prvky, které mají vnější okraje s hodnotou *auto*
2. veškeré volné místo v příslušných směrech je přiděleno těmto okrajům
3. pokud je prvků s automatickými okraji v jednom směru více, místo mezi nimi je distribuováno rovnoměrně
4. až poté jsou spuštěny mechanismy zarovnání

Proto *margin: auto* ovlivňuje polohu pružných prvků na obou osách a také „rozbijí“ vyrovnávací mechanismy (ignoruje vlastnosti *justify-content*, *align-items* a *align-self*), vždyť zarovnání vzniká, když existuje volný prostor. [5 s.45-46] Pokud však veškeré volné místo „přešlo“ do automatických okrajů, není nic na zarovnání. Například jestli položka má automatický okraj shora nebo zespodu, pak takový prvek bude přitlačován buď k horní části kontejneru, nebo ke dnu. A pokud jsou zadány automatické okraje z opačných stran, pak bude prvek umístěn ve středu flexbox kontejneru vzhledem k tomu, že volné místo je rozděleno okraji rovnoměrně. [1] To samé funguje i s vodorovnými hodnotami *margin*.

3.3.9. Výchozí rozměr položky, *flex-basis*

Flex-basis specifikuje základní velikost flex prvku (velikost podél hlavní osy). Pokud hlavní osa směřuje vodorovně, pak hlavní velikost je šířka (*width*) a příčná velikost je výška (*height*). [21] Pokud hlavní osa směřuje svisle, pak je všechno naopak. Pokud *flex-basis* není definován, nebo jeho hodnota je *auto* (výchozí hodnota), základní velikost je převzata ze šířky nebo výšky, což může záviset na velikosti obsahu, pokud ta velikost není přímo definovaná. Vlastnost *flex-basis* má přednost před vlastnosti *width* a *height*, a pokud u flex položky jsou zadány všechny tři vlastnosti, pak *flex-basis* v závislosti na směru hlavní osy přepíše buď šířku nebo výšku. [20]

3.3.10. Koeficient roztažení prvků, *flex-grow*

Pokud uvnitř flex kontejneru na hlavní ose zůstává volné místo, lze požádat, aby se flex element zvýšil a obsadil toto místo. To se provádí pomocí vlastnosti *flex-grow*, kterou lze nazvat „koeficientem chamtivosti“ flex položky. [5 s.48] Vlastnost *flex-grow* má nezáporné číselné hodnoty, její výchozí hodnota je 0. [20] Pokud je hodnota *flex-grow* nulová, flex element „nevyžaduje“ zbývající volný prostor ve flex kontejneru a nebude se zvyšovat tak, aby obsadil toto místo. Pokud je hodnota *flex-grow* větší než nula, pak se flex prvek bude zvyšovat a obsadí zbývající volné místo.

Pokud hned několik flex elementů má hodnotu *flex-grow* větší než nula, rozdělí si volný prostor mezi sebou. [5 s.48] Volné místo bude přidáno flex položkám v poměru hodnoty jejich „koeficientu chamtivosti“. Například pokud jsou ve flex kontejneru dva prvky:

Zdrojový kód č. 3.3.10 - 1. Vlastnost *flex-grow* s koeficienty 1 a 2.

```
.element-1 { flex-grow: 1; }
.element-2 { flex-grow: 2; }
```

Pak druhý prvek má dvakrát více volného místa než první.

Zdrojový kód č. 3.3.10 - 2. Vlastnost *flex-grow* s koeficienty 50 a 100.

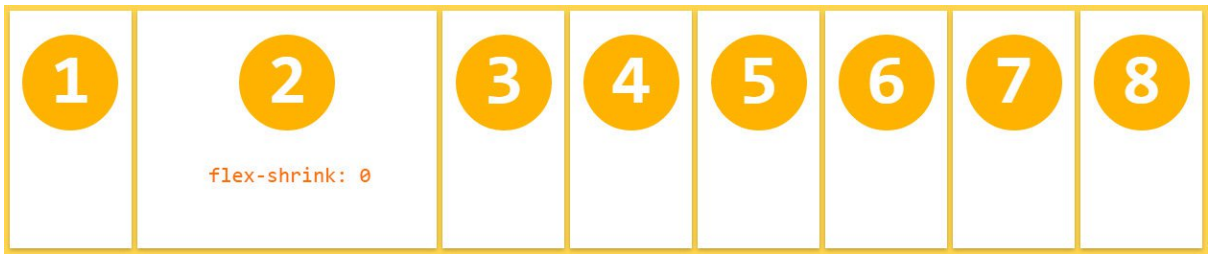
```
.element-1 { flex-grow: 50; }
.element-2 { flex-grow: 100; }
```

I když změníme hodnoty koeficientů těchto prvků na hodnoty 50 a 100, nic to nezmění, protože poměr koeficientů se nezměnil: 100 je dvakrát více než 50. [21] To znamená, že hodnota koeficientu sama o sobě není důležitá, ale důležitý je její poměr s koeficienty zbývajících prvků. [1]

Obrázek 3.3.10 - 1. Vlastnost *flex-grow*
Zdroj: http://babel.es/es/blog/blog/junio-2016/flexbox-css3-layout

3.3.11. Koeficient stlačení prvků, *flex-shrink*

Pokud je součet základních velikostí flex položek větší, než je velikost flex kontejneru, pak vzniká negativní prostor. Mechanismus distribuce funguje nejen pro volný, ale i pro negativní prostor. Flex elementy jsou schopny distribuovat negativní prostor mezi sebou a zmenšovat se. Za zmenšení flex prvků odpovídá vlastnost *flex-shrink*, kterou lze nazvat „koeficientem stlačení". [5 s.51] Vlastnost *flex-shrink* má nezáporné číselné hodnoty, její výchozí hodnota je 1. Pokud je hodnota *flex-shrink* větší než nula, pak se flex prvek bude zmenšovat a „absorbovat" část záporného prostoru, pokud takový existuje. [1] Pokud je hodnota *flex-shrink* rovna nule, flex prvek se nebude zmenšovat. Flex prvky se snaží být co nejvíce „flexibilní" a nevypadnout z jeho kontejneru, tím pádem má *flex-shrink* výchozí hodnotu 1. Vlastnost *flex-shrink* je velmi podobná vlastnosti *flex-grow*. Udává poměr, v němž flex položky „absorbují" negativní prostor. Čím vyšší je hodnota koeficientu stlačení, tím větší je negativní prostor, který bude absorbován, a tím větší bude stlačený element.



Obrázek 3.3.11 - 1. Vlastnost *flex-shrink*
Zdroj: http://babel.es/es/blog/blog/junio-2016/flexbox-css3-layout

3.3.12. Zkratka *flex*

Pomocí zkratky flex je možné současně nastavit koeficienty roztažení, stlačení a základní velikost flex položky. Vlastnost flex se skládá ze tří komponentů, které jsou psány přes mezeru v následujícím pořadí: *flex-grow*, *flex-shrink* a *flex-basis*. [21] Dva následující příklady jsou identické:

Zdrojový kód č. 3.3.12 - 1. Vlastnost *flex*.

```
.flex-item {
  flex: 1 2 300px;
```

```
}

.flex-item {
   flex-grow: 1;
   flex-shrink: 2;
   flex-basis: 300px;
}
```

Výchozí hodnota je *0 1 auto*. [5 s.52]

W3C doporučuje použití zkráceného pravidla, protože správně odmítá jakékoli neurčené hodnoty, aby se přizpůsobil typickému použití. [21]

3.3.13. Zkratka *flex-flow*

Pomocí zkratky *flex-flow* je možné současně nastavit směr hlavní osy a víceřádkovost příčné osy. Vlastnost *flex-flow* se skládá ze dvou komponentů, které jsou psány přes mezeru v následujícím pořadí: *flex-direction* a *flex-wrap*. [20] Dva následující příklady jsou identické:

Zdrojový kód č. 3.3.13 - 1. Vlastnost *flex-flow*.

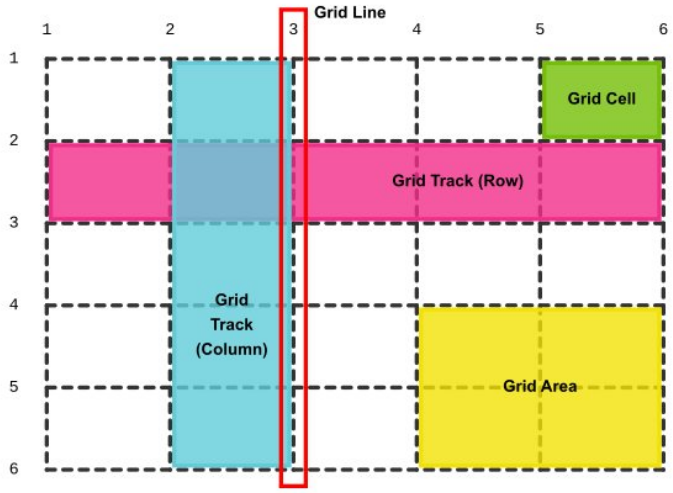
```
.flex-container {
  flex-flow: column wrap-reverse;
}

.flex-container {
  flex-direction: column;
  flex-wrap: wrap-reverse;
}
```

Výchozí hodnota je *row nowrap*.

3.4. Grid layout

Flexbox je skvělý, ale jeho hlavním účelem je pomáhat s umístěním prvků v jedné dimenzi, a to buď horizontální nebo vertikální. [4 s.12] Pro celou stránku s dvourozměrným uspořádáním jsme se většinou spoléhali na frameworky jako Bootstrap nebo Foundation, které nám poskytují řadu tříd, které lze použít k vytvoření rozvržení. [22] Nový modul CSS Grid Layout by měl všechno změnit. Grid byl pod flagom (angl. *behind a flag*) ve většině prohlížečů po dobu více než pěti let. Na vývoj bylo vynaloženo více času, aby se zabránilo vypuštění velkého počtů bugů, jako to bylo s flexboxem. [23]

Nejdříve se musíme seznámit s několika pojmy, které byly zavedeny do CSS Grid. Zdá se nám to povědomé, protože Grid vychází z tabulkového layoutu. Proto obsahuje řádky (*row*), sloupce (*column*) a buňky (*cell*), stejně jako tabulka. [4 s.15] Kromě toho zavádí nový pojem oblast (*grid area*), která může zabírat prostor několika řádků a sloupců. [24] Grid area nabízí způsob, jak označit části mřížky tak, aby položky mohly snadno zapadnout do těchto pojmenovaných oblastí. [22]



Obrázek 3.4 - 1. Základní pojmy
Zdroj: https://webkit.org/blog/7434/css-grid-layout-a-new-layout-module-for-the-web/

3.4.1. Display: grid

Základní prvek gridu je jeho kontejner, který plní stejnou funkci jako kontejner flexboxu. [24] HTML element se stává grid kontejnerem nastavením vlastnosti *display* na hodnotu *grid* nebo *inline-grid*. Poslední hodnota funguje stejně jako *inline-block* nebo *inline-flexbox*. Tedy prvek se navenek chová jako inline a pohybuje se s okolním textem, ale uvnitř se chová jako grid. [4 s.20]

3.4.2. Umístění prvků

Grid se skládá z řádků a sloupců. Ty se nastavují vlastnostmi *grid-template-rows* a *grid-template-columns*. [24] Z nich vyplývá, kolik sloupců a řádků bude mřížka mít a jaké budou jejich rozměry. Na stránce to bude vypadat jako tabulka s neviditelnými hranami. Grid položky se automaticky rozmístí do buněk ve stejném pořadí, jako ve zdrojovém kódu. [26] Obsadí řadu po řádě, zleva doprava.

Zdrojový kód č. 3.4.2 - 1. Vlastnost *grid-template-rows* a *grid-template-columns*.

```
.grid-container {
  grid-template-rows: 100px auto;
```

```
  grid-template-columns: 200px 1fr 2fr;
}
```

Neboli se dá použít zkratku pro zápis těchto dvou vlastností.

Zdrojový kód č. 3.4.2 - 2. Vlastnost *grid-template*.

```
.grid-container {
  grid-template: 100px auto / 200px 1fr 2fr;
}
```

Hodnota *auto* zmámená to, že položka zabere tolik místa, kolik potřebuje její obsah. Funguje naprosto stejně, jako by byla nastavena vlastnost *height* nebo *width*. Ale Grid přináší jednu novou jednotku – fr (angl. *flex factor*). [4 s.32] Pokud je takových jednotek více, rozdělí si prostor v poměru podle čísel před *fr*. V daném příkladu se uvádí, že druhý sloupec zabere třetinu zbývajícího místa, ale až poté, co prvnímu sloupci bude nastavená šířka 200 pixelů. Respektive poslední sloupec zabere dvě třetiny. Podobné chování u flexboxu zajišťuje vlastnost *flex-grow*.

3.4.3. Funkce *minmax*

Fragmenty můžeme používat s novou funkcí *minmax()*, která dovolí nastavení rozměru v určitém rozsahu. Je to taková alternativa pro vlastnosti *min-width*, *max-width*, *min-height* a *max-height*. [4 s.41]

Zdrojový kód č. 3.4.3 - 1. Funkce *minmax*.

```
.grid-container {
  grid-template-columns: auto minmax(500px, 1fr);
}
```

Tímto stylem jsme určili, že druhý sloupec by měl zabírat veškerý volný prostor, ale minimální šířka má být 500 pixelů.

3.4.4. Funcke *repeat*

Opakování stejných hodnot v definici sloupců nebo řádků umí být otravné. Proto je k dispozici funkce *repeat*, která vytvoří příslušný počet buněk. Tyto dva příklady dělají totéž a rozdělí řádek na pět stejně širokých sloupců. [4 s.37]

Zdrojový kód č. 3.4.4 - 1. Funcke *repeat*.

```
.grid-container { grid-template-columns: 1fr 1fr 1fr 1fr 1fr; }
.grid-container { grid-template-columns: repeat(5, 1fr); }
```

3.4.5. Pojmenované oblasti

Pro pohodlnější pracování s gridem můžeme jednotlivé buňky v řádcích a sloupcích přiřadit do oblastí. Oblasti se určují jako série řetězců v uvozovkách, které reprezentují řádky a jména oddělená mezerami, které představují buňky. [4 s.37] Současně by měla tvořit obdélník, nelze mít oblast nesouvislou nebo zahnutou. Počet mezer není rozhodující. Pro lepší znázornění se doporučuje, aby byly řádky zarovnány pod sebe. [24]

Zdrojový kód č. 3.4.5 - 1. Pojmenované oblasti.

```
.grid-container {
  grid-template: 100px auto / 200px 1fr 2fr;
  grid-template-areas:
    "header header header"
    "adv footer aside";
}
```

Jakmile byly nadefinovány a pojmenovány oblasti, lze následně jednotlivé prvky stránky umístit do příslušných pozic pomocí vlastnosti *grid-area*.

Zdrojový kód č. 3.4.5 - 2. Umístění do pojmenované oblasti.

```
.header { grid-area: header };
.adv    { grid-area: adv };
.footer { grid-area: footer };
.aside  { grid-area: aside };
```

Tyto elementy by samozřejmě měly být položkami grid kontejneru. Pokud v definici oblastí nebude dostatek jmen (např. zapomeneme uvést třetí „header“ v první řádce), budou takové buňky prázdné a nevyužité. Prázdnou buňku lze také označit zapsáním tečky. [24]

3.4.6. Nepojmenované oblasti

U velmi složitých layoutů (např. 10 na 10) není snadná práce s pojmenovanými oblastmi. Proto Grid umožňuje umisťování položek gridu pomocí určení oddělovačů (*grid lines*). Oddělovač je mezera mezi řádky nebo sloupci. Poloha položky se udává pomocí vlastností, které určují počáteční a koncový oddělovač v řádku a sloupci. [4 s.50] K tomu slouží čtyři vlastnosti *grid-column-start*, *grid-column-end*, *grid-row-start* a *grid-row-end*.

Zdrojový kód č. 3.4.6 - 1. Určení pozice pomocí oddělovačů.

```
.element {
  grid-column-start: 1;
```

```
  grid-column-end: 3;
  grid-row-start: 1;
  grid-row-end: 5;
}
```

Tento zápis je příliš dlouhý, proto W3C vymyslel několik zkratek pro tyto vlastnosti. Níže uvedené kódy jsou absolutní identické tomu, které byly uvedeny dříve.

Zdrojový kód č. 3.4.6 - 2. Určení pozice pomocí oddělovačů.

```
.element {
  grid-column: 1 / 3;
  grid-row: 1 / 5;
}

.element {
  grid-area: 1 / 1 / 5 / 3;
}
```

Za lomítkem může být také hodnota *span N*, která znamená, že daný prvek se má roztáhnout přes *N* sloupců nebo řádků. [4 s.39]

Zdrojový kód č. 3.4.6 - 3. Kličové slovo *span*.

```
.element {
  grid-ared: 1 / 1 / span 4 / span 2;
}
```

Záporná čísla se počítají od konce. Takže kdybychom chtěli roztáhnout element přes celou šířku, stačí vlastnosti *grid-column* nastavit hodnotu *1 / -1*. Musíme si dát pozor na to, že explicitní umístění může rozšiřovat stávající mřížku. Pokud umístíte obsah do sloupců nebo řádků, které neexistují, budou automaticky vytvořeny.

3.4.7. Oddělovače

Stejně jako u tabulek (atribut *cellspacing*) se dá nadefinovat vzdálenost mezi grid položkami. Navíc se tímto způsobem nevytvoří vnější okraje, což je velká výhoda, protože to neovlivní ostatní elementy stránky. [24] Vlastnost *grid-column-gap* odpovídá za vertikální prostor mezi sloupci, kdežto *grid-row-gap* za horizontální mezi řádky. Výchozí hodnota obou vlastností je 0.

Zdrojový kód č. 3.4.7 - 1. Vzdálenost mezi položkami.

```
.element {
  grid-column-gap: 20px;
```

```
  grid-row-gap: 5px;
}
```

Pro zkrácení danou část kódu lze také zapsat jinak.

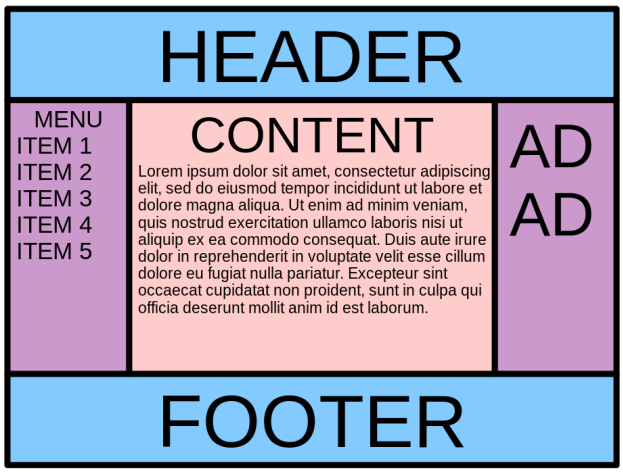
Zdrojový kód č. 3.4.7 - 2. Vzdálenost mezi položkami.

```
.element {
  grid-gap: 5px 20px;
}
```

4. Vlastní práce

4.1. Svatý Grál

Pro první ukázku, byl vybrán klasický webový layout - Svatý Grál (*Holy Grail*). Svatý Grál je tří sloupcový layout, který má nahoře hlavičku a dole patičku, obě dvě plné délky. Daný příklad ukáže rozdíl metodik tvorby dvěma různými způsoby. Postupně se toto rozvržení bude zlepšovat přidáním dalších částí kódů tak, aby se na konec co nejvíc přiblížilo k reálnému případu, který používá většina webových stránek.



Obrázek 4.1 - 1. Svatý Grál
Zdroj: https://www.magespecialist.it/blog/2017/04/11/cssday-2017-resoconto-rapido/

Za zmínku stojí, že v dalších příkladech zdrojových kódů, je použitá metodologie BEM (Block, Element, Modifier) pro pojmenování tříd v CSS. Tato metodologie se stává de facto standardem při tvorbě webových stránek a aplikací, pomáhá při organizace a opětovném použití kódu. Jedním z jejích cílů je zachovat co nejnižší specifičnost (váhu) selektorů. Předtím než je možné vytvořit kaskádové styly, je třeba definovat strukturu HTML kódu. Pro úsporu místa je uvedena pouze část kódu, která je v značce *body*, plný vzor lze najít v příloze této bakalářské práce.

Zdrojový kód č. 4.1 - 1. HTML struktura pro Svatý Grál.

```
<div class="container">
  <header class="header"></header>
  <nav class="nav"></nav>
  <main class="main"></main>
```

```
  <aside class="aside"></aside>
  <footer class="footer"></footer>
</div>
```

4.1.1. Flexbox

V příkladu s flexboxem je bohužel nutné zabalit tři centrální elementy (*nav*, *main*, *aside*) do rodičovského kontejneru pro usnadnění dalšího vývoje. Pro tento účel je používaná obyčejná značka *div* s názvem třídy *content*. Dále je třeba zařídit, aby se každý element uvnitř kontejneru choval jako flex položka. Proto hlavní a centrální kontejnery mají nastavenou hodnotu *flex* vlastnosti *display*. Ale zároveň je nutné, aby elementy *header*, *content* a *footer* byly vyskládány ve svislém směru; tj. aby elementy *nav*, *main* a *aside* byly zobrazeny v jednom řádku.

Zdrojový kód č. 4.1.1 - 1. Nastavení pro kontejnery.

```
.container {
  display: flex;
  flex-direction: column;
}

.content {
  display: flex;
}
```

U elementu *content* není nutné nastavovat vyskládaní do řádku, protože výchozí hodnota už zajišťuje takové chování.

Obrázek 4.1.1 - 1. Svatý Grál. Flexbox, první krok
Zdroj: vlastní screenshot

Na obrázku vidíte výsledné rozmístění jednotlivých elementů. Další krok je určení šířky elementů *nav* a *aside* a nastavení takového chovaní elementu *main*, aby zabíral veškeré volné místo podél vodorovné osy. Obvykle mají boční sidebary pevně nastavenou šířku, pro tento příklad byly vybrány hodnoty 100 a 200 pixelů. Zde je možné použít vlastnost *flex-basis*, ale

v kódu je nastaven klasický *width*, kvůli tomu, že později při optimalizaci zobrazení na různých zařízeních dojde ke změně hodnoty vlastnosti *flex-direction*, a tím se změní základní velikost z *width* na *height*.

Zdrojový kód č. 4.1.1 - 2. Nastavení pro centrální kontejner.

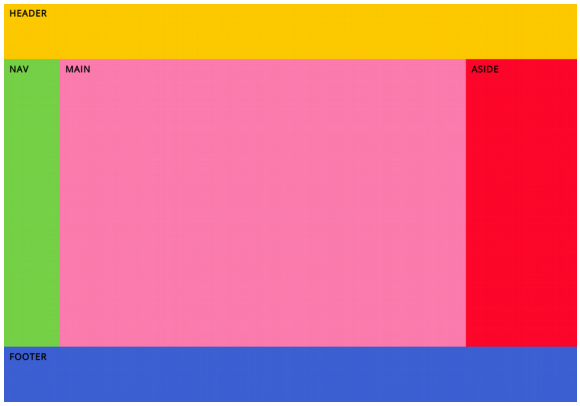
```
.nav   { width: 100px; }
.main  { flex-grow: 1; }
.aside { width: 200px; }
```

Obrázek 4.1.1 - 2. Svatý Grál. Flexbox, druhý krok
Zdroj: vlastní screenshot

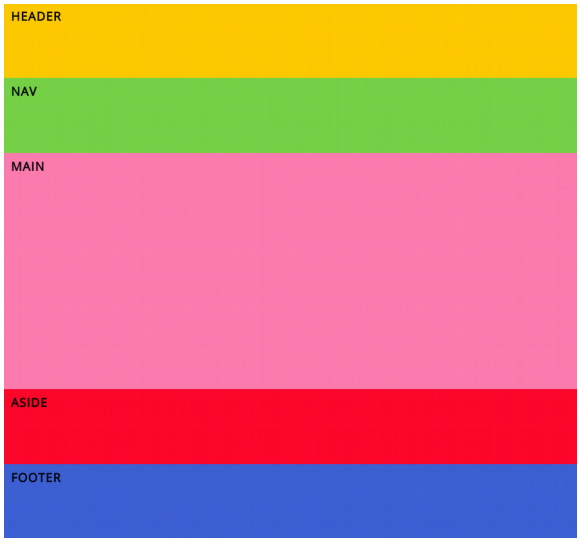
Aby takový layout bylo možně používat na reálném projektu, chybí dvě věci: responzivní design a patička přilepená na spodku obrazovky, tzn. *sticky footer*. Pro zjednodušení byl v tomto případě vytvořen pouze jeden breakpoint, při kterém se *nav* a *aside* přesunou nahoru a dolů od bloku *main*. Přilepení se zajišťuje roztažením výšky hlavního kontejneru stránky na celou výšku okna prohlížeče a dovolením centrálnímu kontejneru obsadit volný prostor.

Zdrojový kód č. 4.1.1 - 3. *Sticky footer* a mobilní verze.

```
.container { min-height: 100vh; }
.content   { flex-grow: 1; }

@media (max-width: 768px) {
  .content {
    flex-direction: column;
  }

  .nav, .aside {
    width: 100%;
  }
}
```



Obrázek 4.1.1 - 3. Svatý Grál. Flexbox
Zdroj: vlastní screenshot



Obrázek 4.1.1 - 4. Svatý Grál. Flexbox. Mobilní verze
Zdroj: vlastní screenshot

4.1.2. Grid layout

Nyní bude vytvořená stejná mřížka, ale za pomocí modulu *CSS*, který byl přímo navržený pro tvorbu layoutů. Výchozím bodem, stejně jako u flexboxu, je hodnota vlastnosti *display*. Prvním krokem je pojmenování oblasti rozmístění budoucích elementů pomocí vlastnosti *grid-template-areas*.

Zdrojový kód č. 4.1.2 - 1. Pojmenování oblasti.

```
.container {
  display: grid;
  grid-template-areas:
    "header header header"
    "nav main aside"
    "footer footer footer";
}
```

Jakmile je *display* nastaven na hodnotu *grid*, každý přímý potomek se stává grid položkou. Díky tomu, že byly vytvořeny pojmenované oblasti lze přiřadit jednotlivé prvky k individuálním místům. Což v důsledku vede k tomu, že na první pohled bude jasná sémantická role každého z bloků. Prvky pak budou vloženy do pojmenované oblasti pomocí vlastnosti *grid-area*, jejíž hodnotou je příslušné jméno.

Zdrojový kód č. 4.1.2 - 2. Vložení do pojmenované oblasti.

```
.header { grid-area: header; }
.nav    { grid-area: nav; }
.main   { grid-area: main; }
```

```
.aside  { grid-area: aside; }
.footer { grid-area: footer; }
```

Dále je třeba ještě nadefinovat, jak mají vypadat sloupce a řádky a přilepit patičku na spodek obrazovky. I zde jsou použity stejné rozměry jako v předchozím případě, aby se oba příklady vizuálně shodovali.

Zdrojový kód č. 4.1.2 - 3. Definice velikosti.

```
.container {
  min-height: 100vh;
  grid-template-columns: 100px 1fr 200px;
  grid-template-rows: 100px 1fr 100px;
}
```

Pro vytvoření responzivního designu stačí změnit rozmístění pojmenovaných oblastí tak, že každý blok bude na novém řádku. Důležité je také nezapomenout změnit velikosti řádků a sloupců.

Zdrojový kód č. 4.1.2 - 4. Mobilní verze.

```
@media (max-width: 768px) {
  .container {
    grid-template-areas:
      "header" "nav" "main" "aside" "footer";
    grid-template-columns: 1fr;
    grid-template-rows: 100px 100px 1fr 100px 100px;
  }
}
```

Výsledek se ničím vizuálně neliší od předchozího příkladu vytvořeného pomocí flexboxu, z toho důvodu zde není jeho obrázek. Nutno podotknout, že stejného výsledku se dá dosáhnout prostřednictvím menšího počtů vlastností, použitím tzn. *shorthand* vlastnosti, ale tím se ztrácí výraznost a prostota kódu.

4.2. Galerie obrázků

Tento komplikovanější příklad dobře ukazuje další možností a flexibilitu grid layoutu. Jak dále uvidíte, není to tak složité, jak to na první pohled vypadá. Jako vzor byla vybrána galerie obrázků, která je umístěna na webu kinopoisk.ru. Kinopoisk je webový projekt podobný IMDB nebo ČSFD, který byl před několika lety koupen korporací Yandex. Pro lepší představu je níže uveden finální výsledek, který je vizuálně úplně stejný jako na kinopoisk.ru.

Galerie na webu Kinopoisku však využívá absolutní pozicování a z toho důvodu není responzivní. Galerie vytvořená v této bakalářské práci se jeho využití vyhýbá.

Obrázek 4.2 - 1. Galerie obrázků
Zdroj: vlastní screenshot

Zdrojový kód HTML bude velmi prostý. Byl vytvořen kontejner, do kterého byly přidány jednotlivé fotografie s unikátní a obecnou třídou tak, aby bylo možné nastavit stejné chováni všem elementům. (Používání identifikátorů je v rozporu s metodologií BEM.)

Zdrojový kód č. 4.2 - 1. HTML struktura pro galerii.

```
<div class="container">
  <div class="photo photo1"></div>
  <div class="photo photo2"></div>
  <!-- another photos -->
  <div class="photo photo19"></div>
  <div class="photo photo20"></div>
</div>
```

Kontejner v sobě bude obsahovat pole z čtverců. Bude mít 4 řádky, každá bude mít 13 čtverců, což vytvoří plochu z 52 buněk, které mezi sebou budou oddělené mezerou o velikosti 5 pixelů. A to všechno je umístěno uprostřed stránky.

Zdrojový kód č. 4.2 - 2. Definice sloupců a řádků pro kontejner.

```
.container {
  display: grid;
  grid: repeat(4, 75px) / repeat(13, 75px);
  grid-gap: 5px;
  justify-content: center;
}
```

Dále je každému obrázku třeba nastavit jednotlivé umístění na ploše. Toho se dá docílit pomocí vlastnosti *grid-column-start*, *grid-column-end*, *grid-row-start* a *grid-row-end*. Ale z důvodu rychlosti a jednoduchosti je obvykle preferováno raději využívat zkratky *grid-column* a *grid-row*.

Zdrojový kód č. 4.2 - 3. Definice polohy pro element.

```
.photo1 {
 grid-column: 1 / span 2;
 grid-row: 1 / span 2;
}
```

Galerie obsahuje obrázky o třech různých velikostech (1x1, 2x2, 4x4). První číslo uvádí, u které čáry element začíná, druhá hodnota - u které končí. Hodnoty mezi sebou musejí být odděleny lomítkem. Pro zjednodušeni lze použit klíčové slovo *span,* které přesně říká, kolik buněk bude obsahovat element. Jinými slovy, těmi prvními hodnotami je nastavená pozice pro pravý horní roh. Zkušený vývojář může použít i vlastnost *grid-area*, která ještě víc zkrátí definici.

Zdrojový kód č. 4.2 - 4. Definice polohy pro element. Druhá verze.

```
.photo1 {
 grid-area: 1 / 1 / span 2 / span 2;
}
```

Výsledek bude úplně stejný. Podobným způsobem je nutné nastavit pozici pro každý další element.

Zdrojový kód č. 4.2 - 5. Definice polohy pro element.

```
.photo2 {
 grid-area: 1 / 3;
}
```

Zde byla definována pozice pouze pro pravý horní roh, ale chybí pozice pro levý dolní.V takovém případě bude obrázek obsahovat pouze jednu buňku. Až po určení všech poloh je možné nastavit *div* elementům jednotlivý obrázek na pozadí, který si zachová poměr stran. Typicky to vede k tomu, že část pozadí není vidět. Ale obrázky používané v tomto příkladu, mají poměr stran 1:1, proto žádná část nebude odříznutá.

Zdrojový kód č. 4.2 - 6. Nastavení pozadí.

```
.photo {
 background-size: cover;
}

.photo1 {
 background-image: url(img/1.jpg);
}
```

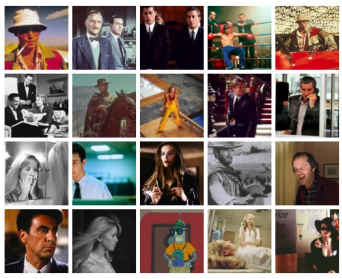
Stejnou operaci s definicí poloh je třeba provést ještě dvakrát pro různé rozšíření. Bohužel v podmínce v media query nelze použít funkci *calc*. A proto by tato hodnota měla být spočítána

ručně. Jako alternativu lze využít proměnné v jakémkoliv preprocessoru nebo *custom property* (Out-of-the-box funkcionalita v CSS), které zatím podporuje kolem 80% prohlížečů. [27] Hodnota *max-width* se počítá následovně: suma šířky buňky a šířky okraje krát počet sloupců (všechny hodnoty jsou z předcházejícího breakpointu). V posledním *media query* všechny obrázky budou umístěny do jedné buňky, proto by třída *photo* měla mít výchozí hodnotu vlastnosti *grid-area*.

Zdrojový kód č. 4.2 - 7. Tabletová a mobilní verze pro kontejner.

```
@media (max-width: 1030px) {
  .container {
    grid: repeat(5, 75px) / repeat(10, 75px);
  }

  /* Define positions of all photos due to new grid  */
}

@media (max-width: 800px) {
  .container {
    grid: repeat(4, 75px) / repeat(5, 75px);
  }

  .photo {
    grid-area: auto / auto;
  }
}
```

Obrázek 4.2 - 2. Galerie obrázků. Tabletová verze.
Zdroj: vlastní screenshot



Obrázek 4.2 - 3. Galerie obrázků. Mobilní verze.
Zdroj: vlastní screenshot

4.3. Navigace

Navigace je jedna z nejdůležitějších částí webové prezentace a je základním nástrojem pro pohyb mezi stránkami. K implementaci navigačního menu jsou použity značky *ul* a *li* pro jednotlivé odkazy. Celý seznam pro sémantiku je zabalen do značky *nav*, čím je zvýšena

přístupnost webu pro vyhledávací roboty a hlasové čtečky. Navíc je přidáno místo pro logo nebo název webu, které bude umístěno vlevo. Ostatní položky seznamu budou přitisknuty k pravé straně.V dnešní době se navigace samozřejmě nemůže obejít bez přizpůsobeni pro menší rozlišení. Proto, v případě, že se odkazy nevejdou do jedné řady, dojde k jejich přesunutí pod logo doprostřed. Tam se přesune i samotné logo.

Zdrojový kód č. 4.3 - 1. HTML struktura pro navigaci.

```
<nav class="menu">
  <ul class="menu__container">
    <li class="menu__logo"></li>
    <li class="menu__arrow">↓</li>
    <li class="menu__item"><a href="#" class="menu__link"></a></li>
    <!-- another links -->
    <li class="menu__item"><a href="#" class="menu__link"></a></li>
  </ul>
</nav>
```

Zpravidla navigace zůstává na stejném místě v okně, i při rolovaní stránky, aby k ní uživatel měl vždy snadný přístup. Zároveň jsou odkazy schované pod logo proto, aby navigace nezabírala většinu volného místa na mobilech a tabletech. Opakované zobrazení seznamu bude probíhat kliknutím na šipku, která je další položkou umístěnou přímo pod logem. Tato operace bude probíhat pomocí JavaScriptu, který skryje nebo opět zobrazí ostatní položky.

Zdrojový kód č. 4.3 - 2. Nastavení pozice pro navigaci.

```
.menu {
  position: fixed;
  width: 100%;
}
```

Jak už bylo zmíněno, flexbox velice usnadňuje práci. V daném případě je pro vytvoření popisované navigace třeba použít pouze tři vlastnosti: *display*, *align-items* a *flex-direction*.

Zdrojový kód č. 4.3 - 3. Uspořádání položek seznamu.

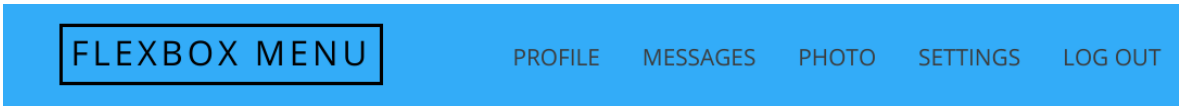
```
.menu__container {
  display: flex;
  align-items: baseline;
}

@media screen and (max-width: 790px) {
  .menu__container {
    flex-direction: column;
    align-items: center;
  }
}
```

Přitlačení navigace je zajištěno pomocí nastavení automatického pravého vnějšího okraje u loga. To je možné díky mechanismu distribuce volného místa u flex prvků.

Zdrojový kód č. 4.3 - 4. Centrování loga.

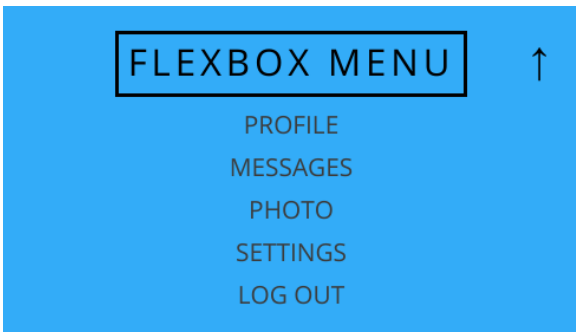
```
.menu__logo {
  margin-right: auto;
}
```

Nyní je již možné říci, že navigace je z pohledu rozložení layoutu z velké části dokončena. Na obrázku níže byly provedeny pouze kosmetické úpravy, jako vytvoření pozadí, okraje, barvy, fonty a tak dále.



Obrázek 4.3 - 1. Navigace.
Zdroj: vlastní screenshot

Obrázek 4.3 - 2. Navigace. Mobilní verze.
Zdroj: vlastní screenshot



Obrázek 4.3 - 3. Navigace. Mobilní verze.
Zdroj: vlastní screenshot

4.4. Podpora prohlížečů

4.5. Flexbox

Podpora prohlížečů pro flexbox je vynikající a většina prohlížečů v této chvíli nepotřebuje žádný prefix. Pokud ignorujete IE10 a nižší verze, což je doporučeno společností Microsoft [28], můžete bez obavy používat verzi bez prefixu. Většina problémů s flexboxem se týká změn ve specifikaci, které byly provedeny během vývoje. Pokud se snažíte zajistit zpětnou kompatibilitu se staršími verzemi prohlížečů, zejména s IE10 a IE11, stránka Flexbugs (github.com/philipwalton/flexbugs) je pro vás užitečným nástrojem. Zde si je možné všimnout, že mnoho z uvedených chyb se vztahuje výlučně na staré verze prohlížečů, tudíž

jsou v jejich aktuálních verzích již opraveny. Každý z bugů má vypsané řešení, což vám může ušetřit mnoho hodin práce. Chcete-li zajistit podporu flexboxu ve velmi starých prohlížečích, můžete navíc přidat vlastnosti s prefixem do CSS. Ale nezapomeňte, že vlastnost bez prefixu by měla byt umístěná jako poslední. Prohlížeče podporují také verzi bez prefixu pro správné zobrazení stránek, které byly vyvíjeny před tím, než byl nový standard oficiálně přijat. A proto je nutné, aby prohlížeče přepsaly experimentální implementace s prefixy a použily optimální variantu. Tohle to se týká jakýchkoliv vlastností, nejen flexboxu. Ale i tento problém se stává méně a méně aktuální. Hlavním důvodem je to, že prohlížeče se samy automaticky aktualizují (*evergreen browsers*), aniž by nutily uživatele stáhnout poslední verze softwaru, anebo se ptaly se na povolení aktualizací. Dobrou pomůckou může být Autoprefixer Online (autoprefixer.github.io). Je to užitečná služba pomocí níž je možné zjistit, které předpony se doporučuje užívat pro jednotlivé typy a verze prohlížečů. Další možností je nainstalovat tento plugin do textového editoru, který automaticky přidává prefixy po uložení souboru.

4.6. Grid layout

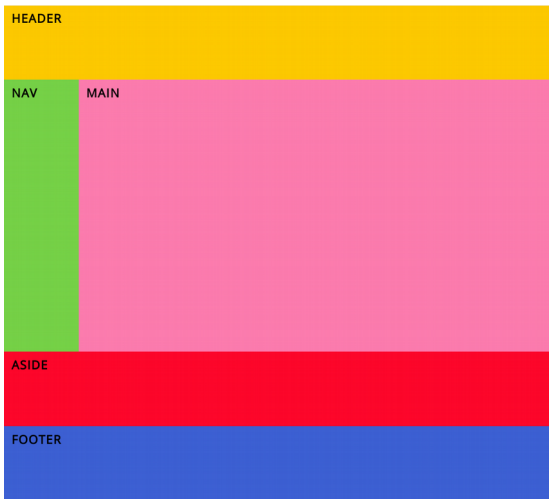
Je třeba říci, že původně byl CSS Grid Layout implementován v aplikaci Internet Explorer 10, a do verzi Edge 15 je nutné používat příponu -ms. Ale během času se tato specifikace významně změnila. Velmi náročné a občas není možné zároveň podporovat obě verze najednou. Jak ukazuje praxe, vývojáři webových prohlížečů se budou v budoucnosti vyhýbat používání různých přípon. Chrome, Firefox, Safari, Opera vždy podporovaly verzi bez prefixu. Ovšem s jednou podmínkou, uživatel musel v nastavení zapnout experimentální režim, aby prohlížeč začal rozumět novým vlastnostem. Až po testovaní v roce 2017 každý velký prohlížeč přesunul podporu do OOTB režimu. Podpora všech vlastností a hodnot podrobně popsaných v dokumentaci je v prohlížečích interoperabilní. To znamená, že pokud napíšete kód ve Firefoxu, měl by v Chromu zobrazovat stejným obsah a pracovat stejným způsobem.

4.7. Současná podpora

Každý frontend vývojář musí znát cílové publikum webu, který vytváří. Na základě toho by se měl rozhodnout o tom, které technologie může používat během vývoje. Hlavní problém je v tom, že většina v minulosti prodávaných počítačů měla již předinstalovaný IE a převážná část koncových uživatelů je s ním spokojená. V souvislosti s tím, v dalším příkladu je cílem navrhnut takový layout, který bude podporovat každý prohlížeč. Jakmile se podíl uživatelů,

kteří používají IE, sníží na úroveň kolem 5ti procent (nebo na jakékoliv podobně nízké procento), je možné smazat část kódu, která odpovídá za implementaci layoutu u starých prohlížečů a nechat tu, která je založená na grid layoutu. Výhoda takového řešení je v tom, že není nutné v budoucnosti přepisovat kód a web bude připraven na okamžité ukončení podpory starého softwaru. Návrh takového řešení umožňuje pravidlo *@support*, které nabízí mechanismus pro selektivní použití kaskádových stylů na základě podporovaných vlastností. Pravidla budou v rámci bloku aplikována, pouze v případě, že prohlížeč podporuje pravidlo uvedené v závorkách. V opačném případě bude tento blok prostě ignorován. Pro další ukázku byl vybrán vám již dobře známý Svatý Grál. Pro zvýšení zajímavosti a komplikovanosti byl navíc upraven přidáním jednoho breakpointu pro tablety. Zbytek layoutu zůstává vizuálně absolutně stejný, jako ve dvou předchozích ukázkách.

```
<div class="container">
  <header class="header">Header</header>
  <div class="content">
    <nav class="nav">Nav</nav>
    <main class="main">Main</main>
    <aside class="aside">Aside</aside>
  </div>
  <footer class="footer">Footer</footer>
</div>
```



Obrázek 4.7 - 1. Svatý Grál. Současná podpora. Tabletová verze.
Zdroj: vlastní screenshot

Nejprve je nutné zajistit podporu starých prohlížečů za pomoc plovoucích prvků. Přitlačeným elementům uvnitř třídy *content* doleva a jsou nastaveny šířky, které dohromady dávají 100%. U breakpointu je šířka přepočítána tak, aby se na tabletech *aside* element přesunul dolů a *main* obsadil uvolněné místo, přičemž na mobilních telefonech bude každý ze zmíněných elementů na novém řádku.

Zdrojový kód č. 4.7 - 2. Nastavení struktury a velikosti elementů.

```
.nav, .main, .aside {
  float: left;
}

.nav {
  width: 100px;
```

```
}

.main {
  width: calc(100% - 100px - 200px);
}

.aside {
  width: 200px;
}
```

Pro grafické znázornění jsou elementům nastaveny výšky tak, aby obsadily veškeré prázdné místo ve vertikálním směru, tím se zajišťuje přitlačení patičky na spodek obrazovky. Z důvodu úspory místa v této bakalářské práci je možné tu část kódu najít v příloze. Nelze zapomenout na zastavení obtékání u centrálního kontejneru, aby se elementy pod ním zobrazovaly správně. K tomu je používán stejný clearfix, který byl uveden v teoretické části. Dalším krokem je napsaní kódu, který vytvoří layout stránky v nových prohlížečích. V tomto případě je veškerý zmíněný kód umístěn v bloku *@support,* pro zjednodušení tento blok je zde uveden pouze jednou. V neposlední řadě je nutné změnit způsob zobrazení prvků a nadefinovat šířku sloupcům.

Zdrojový kód č. 4.7 - 3. Blok *support*.

```
@supports (display: grid) {
  .container {
    display: grid;
    min-height: 100vh;
    grid-template-rows: 100px 1fr 100px;
  }
}
```

Neboť před tím byla nastavena šířka a výška, když vytvářeli rozvržení pomocí vlastnosti *float*, nyní je nutné nastavit jejich hodnoty na *auto*, aby grid mechanismus bral tuto hodnotu z vlastnosti *grid-template-columns* a *grid-template-rows*. Ve zdrojovém kódu CSS je část pravidel označená komentářem, který upozorňuje na to, že daná část by měla být vymazaná při ukončení podpory starších prohlížečů. Nelze sice předpokládat, že se tak stane v nejbližší budoucnosti, ale záměrem tohoto komentáře je ušetřit čas člověku, který budu kód upravovat po vás.

Zdrojový kód č. 4.7 - 4. Zrušení nastavení velikosti.

```
/* delete after finishing float support */
.nav, .main, .aside {
  width: auto;
```

```
  height: auto;
}
```

Do produkce se takto zanechané zprávy přesto nedostanou, protože stránka projde optimalizací pro zvýšení rychlosti načítání. V rámci té optimalizátor vymaže nepotřebné mezery, převody na nové řádky, komentáře a vymění hodnoty a vlastnosti tak, aby jejich zápis byl co nejkratší. Na následujícím příkladu je možné vidět vstup a výstup po provedení takové operace.

Zdrojový kód č. 4.7 - 5. Příklad optimalizace CSS kódu.

```
.foo { color: #ff0000; }
/* Color for logo */
.bar { color: rgba(255, 0, 0, 1); }
```

```
.bar,.foo{color:red}
```

Například po optimalizaci celého CSS souboru u tohoto příkladu pomocí nástroje CSSO bylo ušetřeno 45% místa. Současně jsou vymazány pseudo-elementy, které byly použity u clearfixu.

Zdrojový kód č. 4.7 - 6. Smazání pseudo-elementů.

```
.content::after,
.content::before {
  display: none;
}
```

Pro centrální kontejner byly vytvořeny dva breakpointy. U každého je pomocí vlastnosti *grid-template-areas* nadefinovaná šablona umístění vnitřních elementů. Pro tablety je potřeba uvést šířku a výšku pro řádky a sloupce. Poté jsou nadefinovaným oblastem přiřazené jednotlivé prvky pomocí *grid-area*. U druhého breakpointu je nastavená velikost na *none*, protože jinak, by vlastnosti grid-template-columns a grid-template-rows dědily hodnoty z těch nastavených pro tablety.

Zdrojový kód č. 4.7 - 7. Tabletová a mobilní verze.

```
@media (max-width: 768px) {
  .content {
    grid-template-areas:
      "nav main"
      "aside aside";
    grid-template-columns: 100px 1fr;
    grid-template-rows: 1fr 100px;
  }
}

.nav   { grid-area: nav; }
.main  { grid-area: main; }
```

```
@media (max-width: 576px) {
  .content {
    grid-template-areas:
      "nav"
      "main"
      "aside";
    grid-template-columns: none;
    grid-template-rows: none;
  }
}
```

`.aside { grid-area: aside; }`	

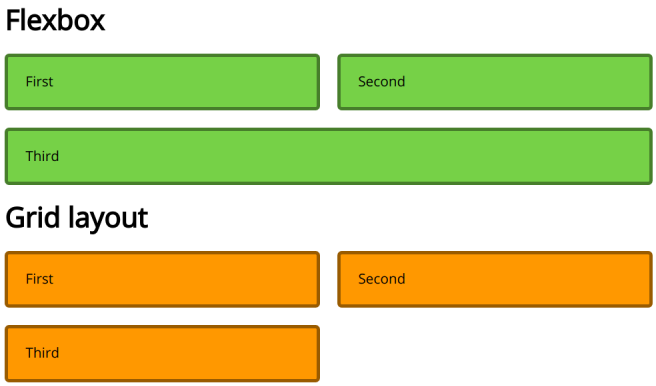
Poslední věc, kterou je třeba vymezit, je šablona pro hlavní kontejner, který v sobě obsahuje v záhlaví, centrální kontejner a patičku. Z důvodu toho, že třída *container* obsahuje pouze tři podřízené prvky první úrovně, a každý z nich je na novém řádku, není nutné vytvářet pojmenované oblasti.

Zdrojový kód č. 4.7 - 8. Definice pro centrální kontejner.

```
.container {
  display: grid;
  min-height: 100vh;
  grid-template-rows: 100px 1fr 100px;
}
```

5. Výsledky a diskuse

Na základě výsledků dosažených při návrhu předchozích příkladů, lze vyvodit jednoduchý závěr o tom, v jakých případech, a kterou techniku je nejlépe použít. Z důvodu flexibility flex prvků je velmi snadné přizpůsobit jejich chování při přenosu řádků nebo při neznámém počtu prvků. Pokud například existuje sada prvků a je nutné je rovnoměrně rozložit do kontejneru. Flexbox povolí prvkům, aby rozhodly samy, kolik místa bude každá položka zabírat. Pokud se elementy dostanou na nový řádek, zaberou si prostor pro sebe v závislosti na jejich velikosti a volném prostoru, který je v tomto řádku k dispozici. Na obrázku níže můžete vidět znázornění tohoto chování.



Obrázek 5 - 1. Rozdíl mezi fleboxem a grid layoutem.
Zdroj: vlastní screenshot

Flexbox je ideální pro zarovnání obsahu uvnitř elementu. Použijte ho k umístění malých detailů projektu jako navigaci. Ideálně se dá použít flexbox pro zarovnání elementů s neznámou šířkou a výškou na střed. Hlavní rozdíl mezi flexboxem a grid layoutem je v tom, že flexbox je v podstatě určen pro umístění položek v jedné dimenzi, v řádku NEBO ve sloupci. Grid layout slouží k rozvržení položek ve dvoudimenzionálním prostoru – řádcích A sloupcích. Většina problémů s flexboxem si způsobují vývojáři sami, protože se pokoušejí vytvořit rozložení pomocí flexboxu ve dvou dimenzích. Díky možnosti vytvoření pojmenované oblasti grid layout skvěle vyhovuje tvorbě asymetrického layoutu. Také je vhodný pro umístění základních elementů stránky, jako je hlavička, patička a sidebary, protože kvůli možnosti vytváření pojmenované oblasti se dá jednodušeji měnit pozice elementů na různých zařízeních. Pravdou však zůstává, že neexistuje jednoznačně lepší systém, jak flexbox, tak grid jsou dobré v různých věcech a měly by být používány společně,

ne jako alternativy k sobě navzájem.

6. Závěr

V této bakalářské práci se autor snažil, co nejsrozumitelněji popsat principy fungování těchto dvou nových modulů a popsat rozdíly a jejich podobnosti. V neposlední řadě se také věnuje ukázání jejich možného uplatnění na reálných příkladech. Z čeho vyplývá, že flexbox je vhodný při práci s menšími detaily v projektu. Optimální využití grid layout je při tvorbě rozložení stránky a asymetrického umístění jednotlivých prvků v kontejneru. Což vede k tomu, že by tyto moduly měly být používány současně a vzájemně se doplňovat.

V oblasti IT se ale naneštěstí není možné jednou provždy něco naučit a aplikovat získané znalosti v průběhu celého života v praxi. Pokrok v této oblasti je velmi rychlý a musí být neustále sledován. Možná vás to trochu překvapí, ale po mnoha letech se tyto moduly stále ještě nepoužívají. Hlavním problémem je to, že vlastníci stránek nechtěli ztratit část návštěvníků, kteří stále používali staré prohlížeče. Ale každým dnem podíl těchto návštěvníků klesá. Dalším faktorem je i to, že vývojáři jsou natolik zvyklí používat plovoucí prvky pro vytvoření layoutu, že se nechtějí tohoto způsobu vzdát. Cílem mé práce bylo seznámit vývojáře s touto technologií a přesvědčit je, že nastal čas znovu se vzdělávat a aplikovat v praxi flexbox a grid layout. Nové standardy mohou nejen zjednodušit jejich vývoj a šetřit čas, ale také poskytují příležitosti, které neexistovaly předtím. Proto by na to frontend vývojáři měli být připraveni. Důkazem tohoto trendu je i fakt, že základní CSS frameworky již opustily vytváření mřížky pomocí plovoucích prvků.

Hlavním cílem práce bylo přesvědčit čtenáře, že už dnes je třeba začít studovat a uplatňovat alespoň částečně nové standardy v praxi. Tím lze zvýšit kvalifikaci a hodnotu na trhu práce. A největší důvod spočívá v tom, že si vývojář sám sobě ulehčí práci. Protože jakmile je ovládne, vývoj pro něj bude jednodušší a příjemnější, než kdy jindy. Čas strávený na školení se mnohonásobně vyplatí.

Z praktických ukázek lze odvodit to, že kombinace flexboxu a grid layoutu dokáže vyřešit všechny existující problémy CSS, jako stejně vysoké sloupce, zarovnaní na střed bloku s neurčitou výškou, tvorbu asymetrického layoutu, tvorbu layoutu s neznámým počtem prvků. Z čehož vyplývá, že paralelní použití těchto dvou modulů přináší velký počet možností, které před jejich vznikem nebyly vývojářům dostupné.

7. Seznam použitých zdrojů

7.1. Seznam literatury

1. A Complete Guide to Flexbox | CSS-Tricks. CSS-Tricks [online]. Dostupné z: https://css-tricks.com/snippets/css/a-guide-to-flexbox/
2. David Sawyer McFarland. CSS: The Missing Manual, 4th Edition. O'Reilly Media, 2015. ISBN-13: 978-1491918050
3. Eric A. Meyer. CSS Floating: Floats and Float Shapes. O'Reilly Media, 2016. ISBN-13: 978-1491929643
4. Eric A. Meyer. Grid Layout in CSS: Interface Layout for the Web. O'Reilly Media, 2016. ISBN-13: 978-1491930212
5. Estelle Weyl. Flexible Boxes in CSS: Free Yourself with Flexbox. O'Reilly Media, 2017. ISBN-13: 978-1491930045
6. World Wide Web Consortium (W3C). World Wide Web Consortium (W3C) [online]. Dostupné z: https://www.w3.org/
7. Layout | Firemnislovnik [online] [cit. 04.03.2018]. Dostupné z: http://www.firemnislovnik.cz/l/layout/
8. Co je Layout? | Adaptic [online] [cit. 04.03.2018]. Dostupné z: http://www.adaptic.cz/znalosti/slovnicek/layout/
9. Co je to Layout? | IT Slovník [online] [cit. 04.03.2018]. Dostupné z: https://www.it-slovnik.cz/pojem/layout
10. Responzivní adaptivní webdesign | Webdesign studio Chabera. [online] [cit. 04.03.2018]. Dostupné z: http://www.chabera.cz/responzivni-webdesign-15
11. Jak předělat web na responsivní | Zdroják. [online] [cit. 04.03.2018]. Dostupné z: https://www.zdrojak.cz/clanky/jak-predelat-web-na-responsivni/
12. Jak vytvořit responzivní design webu | WhiteHat. [online] [cit. 04.03.2018]. Dostupné z: https://www.whitehat.cz/jak-vytvorit-responzivni-design/
13. Float v CSS | Je čas. [online] [cit. 04.03.2018]. Dostupné z: http://jecas.cz/float
14. Float | ITnetwork. [online] [cit. 04.03.2018]. Dostupné z: https://www.itnetwork.cz/html-css/css-manual/pozicovani/float-css-3-vlastnost-cesky-manual/

15. Obtékání | Jak psát web. [online] [cit. 04.03.2018]. Dostupné z: http://www.jakpsatweb.cz/css/obtekani.html
16. Float css vlastnost | HelpMark. [online] [cit. 04.03.2018]. Dostupné z: https://www.helpmark.cz/navody/html-css/css-vlastnosti/315-float
17. Visual formatting model details | W3C. [online] [cit. 04.03.2018]. Dostupné z: https://www.w3.org/TR/CSS2/visudet.html#float-width
18. Alignment, font styles, and horizontal rules | W3C. [online] [cit. 04.03.2018]. Dostupné z: https://www.w3.org/TR/html401/present/graphics.html#adef-align
19. The Clearfix: Force an Element To Self-Clear its Children | CSS-Tricks. [online] [cit. 04.03.2018]. Dostupné z: https://css-tricks.com/snippets/css/clear-fix/
20. Basic concepts of flexbox | MDN. [online] [cit. 05.03.2018]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout/Basic_Concepts_of_Flexbox
21. CSS Flexible Box Layout Module | W3C. [online] [cit. 05.03.2018]. Dostupné z: https://www.w3.org/TR/css-flexbox/
22. CSS Grid Layout: Introduction | Alligator. [online] [cit. 05.03.2018]. Dostupné z: https://alligator.io/css/css-grid-layout-intro
23. Google Developers. [online] [cit. 05.03.2018]. Dostupné z: https://developers.google.com/web/updates/2017/01/css-grid
24. Grid, aneb revoluce tabulkového layoutu | CSS3 Tipy a Triky. [online] [cit. 05.03.2018]. Dostupné z: http://css.chobits.ch/grid/
25. Line-based placement with CSS Grid | MDN. [online] [cit. 05.03.2018]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout/Line-based_Placement_with_CSS_Grid
26. An Introduction to the `fr` CSS unit | CSS-Tricks. [online] [cit. 05.03.2018]. Dostupné z: https://css-tricks.com/introduction-fr-css-unit/
27. Can I use | Custom property [online]. Dostupné z: https://caniuse.com/#search=custom%20properties
28. Internet Explorer End of Support for Older Versions | Microsoft Corporation [online] [cit. 13.03.2018]. Dostupné z: https://www.microsoft.com/en-us/WindowsForBusiness/End-of-IE-support

7.2. Seznam obrázků

7.3. Seznam zdrojových kódů

8. Přílohy

- **Holy Grail**
    - Flexbox: flexbox.html, styles.css
    - Grid layout: grid.html, styles.css
    - Fallback (grid layout + float): fallback.html, styles.css, styles.css.min
- **Menu**: menu.html, styles.css
- **Gallery**: gallery.html, styles.css
    - **img**: 1-20.jpg
- **Difference**: difference.html, styles.css