# INFORMATIKA

# Moderní layout webových stránek

*MARTIN TRNEČKA*

Přírodovědecká fakulta, Univerzita Palackého, Olomouc

## Layout webové stránky

Tvorba webových stránek byla vždy specifickou oblastí informatiky, kterou významně ovlivnilo posledních 30 let dramatického vývoje Internetu a technologií s ním spojených. Typickým rysem je např. neustálý vývoj nových webových specifikací (standardů), které se snaží, byť omezeně, reagovat na potřeby v oblasti webových technologií a tvorby webových stránek. Znalosti tvorby webových stránek proto velice rychle zastarávají.

Ačkoliv jsou původní technologie a principy do jisté míry stále platné a reálně použitelné, nepoužívání nových metod dramaticky degraduje možnosti tvorby webových stránek. Dalším aspektem je, že znalost pouze základních technologií je velice vzdálená reálné praxi, která často moderní technologie integruje již v jejich raných fázích vývoje. V dnešní době se naprosto běžně používají webové specifikace, které nejsou dokončené.[1)]

Při běžné výuce na středních školách jsou nové webové technologie často opomíjeny. Toto opomíjení má celou řadu důvodů. Nové technologie jsou obvykle velice složité a přesahují to, co bývá označováno jako „základy tvorby webových stránek“. K tomu přidejme absenci kvalitní české literatury, fakt, že staré technologie pořád fungují, a výsledkem je, že se často učí věci, které se v praxi již dlouho nepoužívají.

Jednou z takových oblastí je tvorba layoutu, neboli základního rozvržení webové stránky. V této oblasti se dlouho nedělo nic nového. První změny

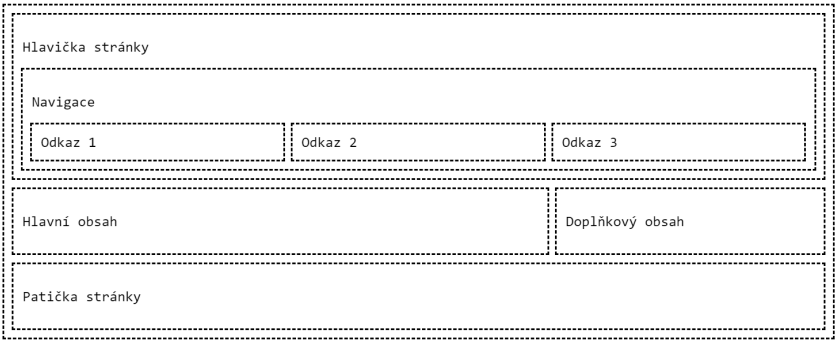
[1)] Webové specifikace procházejí dlouhým procesem vývoje. Například, vývoj specifikace jazyka HTML 5 trval 10 let. Aby webové prohlížeče držely krok s dobou, běžně implementují rozpracované verze webových specifikací, jež jsou dostupné na stránkách World Wide Web konsorcia (W3C), které webové specifikace vyvíjí.

přišly s potřebou responzivního zobrazování[2)], ale jednalo se o změny spíše kosmetické. Vývoj šel dál a zastaralé technologie nedokáží plnit požadavky na vzhled soudobých webových stránek. Jednu z největších změn v této oblasti způsobily specifikace CSS FlexBox [3] a CSS Grid [4], na které se zaměříme.[3)] Obě specifikace jsou považovány za revoluci v moderním web designu. Jsou velice komplexní a jejich celkové možnosti dalekosáhle přesahují to, co zde budeme demonstrovat. Byť jsou relativně nové (CSS Flexbox specifikace byla dokončena v listopadu 2018, CSS Grid specifikace v prosinci 2017), dnešní reálná praxe je obě považuje za základní technologie a běžně je používá.

Tento článek není referenční příručkou, ani detailním popisem uvedených specifikací; ukážeme ale, jak mohou usnadnit tvorbu layoutu stránky a odstranit problémy jen obtížně řešitelné pomocí starších technologií. Obě specifikace použijeme naprosto minimalistickým způsobem. Hlavním cílem článku je tak motivovat k zařazení, byť pouze v míře zde uvedené, těchto specifikací do výuky základů tvorby webových stránek. Vše budeme demonstrovat na tzv. dvousloupcovém layoutu, který aktuálně patří mezi nejpoužívanější layouty webových stránek.

### Webová stránka před pár lety

Než začneme, vytvoříme si „postaru“ základní webovou stránku, na které budeme demonstrovat použití nových specifikací. Webová stránka bude vypadat následovně.




[2)] Responzivní rozhraní se přizpůsobuje zařízení, na kterém je aktuálně webová stránka zobrazena. Prvopočátky sahají do roku 2007, kdy byl představen první iPhone.

[3)] Názvy specifikací zde uváděné jsou slangové. Oficiální název specifikací je „CSS Flexible Box Layout Module Level 1“ a „CSS Grid Layout Module Level 1“.

Vizuální podoba stránky (veškeré čáry, okraje a popisky) nebude pro naše účely důležitá, důležité bude pouze rozmístění hlavních prvků stránky, tedy hlavičky stránky, navigace, hlavního a doplňkového obsahu a patičky stránky. Dále si uvedeme HTML kód (pro jednoduchost pouze obsah `body` elementu)

```
<body>
  <div>
    <header>
      <p>Hlavička stránky</p>
      <nav>
        <p>Navigace</p>
        <ul>
          <li>Odkaz 1</li>
          <li>Odkaz 2</li>
          <li>Odkaz 3</li>
        </ul>
      </nav>
    </header>

    <main>
      <p>Hlavní obsah</p>
    </main>

    <aside>
      <p>Doplňkový obsah</p>
    </aside>

    <footer>
      <p>Patička stránky</p>
    </footer>
  </div>
</body>
```

a CSS kód, použitý pro vytvoření této stránky.

```
header, main, aside, footer, nav, body > div {
  outline: 2px dashed black;
  box-sizing: border-box;
  padding: 10px;
}

```

```
7  body > div {
8    font-family: monospace;
9    font-size: 1.2rem;
10   color: black;
11   width: 900px;
12   margin: auto;
13 }
14
15 main, aside {
16   margin: 12px 0;
17 }
18
19 main {
20   float: left;
21   width: 580px;
22 }
23
24 aside {
25   float: right;
26   width: 290px;
27 }
28
29 footer {
30   clear: both;
31 }
32
33 nav ul {
34   padding-left: 0px;
35   margin-bottom: 0px;
36 }
37
38 nav ul::after {
39   content: '';
40   clear: both;
41   display: block;
42 }
43
44 nav ul li {
45   float: left;
46   box-sizing: border-box;
47   outline: 2px dashed black;
```

```
48    padding: 10px;
49    list-style-type: none;
50    width: 273px;
51  }
52
53  nav ul li:nth-child(2) {
54    margin-left: 10px;
55    margin-right: 10px;
56  }
```

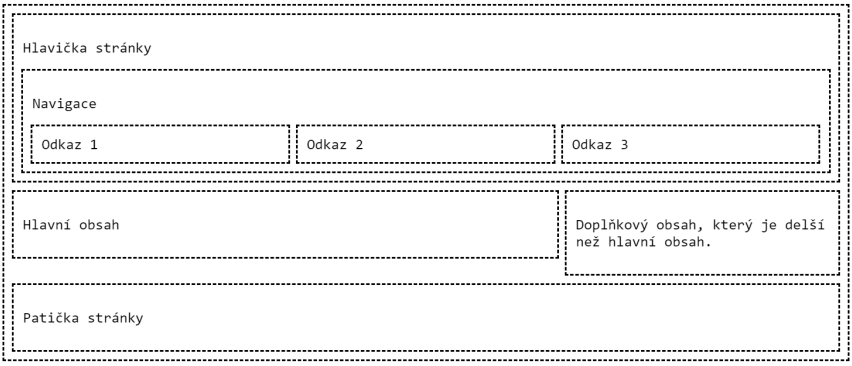
Celý kód je k dispozici na stránce https://codepen.io/trnecka/pen/eXEGBa. Nyní se na zdrojový kód stránky podíváme podrobněji. V případě HTML kódu (a tudíž ani v případě CSS) nejsou použity atributy `id` a `class` pro identifikaci elementu. Ačkoliv by v tomto případě bylo možné je použít, typickou začátečnickou chybou je jejich nadužívání. To vede zejména u rozsáhlejších projektů ke špatnému CSS kódu, který se obtížně udržuje.[4)]

Uvedený CSS kód je velice jednoduchý a jedná se o standardní a v minulosti běžně používané řešení. Jedinou výjimkou by mohlo být modernější použití pseudo-elementu `::after` (CSS pravidlo na řádcích 38–42) pro ukončení obtékání. Základní layout stránky je vytvořen pomocí obtékání prvků (vlastnost `float`). Pomocí stejné vlastnosti je rozmístěna i navigace stránky. Pro jednoduchost je stránka navržena na fixní šířku `900px`. Ačkoliv to na první pohled nemusí být zcela zřejmé, toto řešení má několik úskalí. Prvním úskalím je nastavení vlastnosti `width` na hodnotu `273px` na řádku 50. Pokud spočítáme šířku všech rodičovských elementů, což je velice jednoduché, jelikož jsme si vše usnadnili použitím `box-sizing: border-box;`, zjistíme, že celková šířka je `899px`, což je o jeden pixel méně. Pokud se podíváme na výslednou stránku na zařízení s vyšším rozlišením, například na retina displeji, je tento chybějící pixel[5)] znatelně vidět (pravý okraj u elementu obsahujícího text `Odkaz 3` je větší než ostatní okraje). Problémem je, že tato situace se nedá jednoduše vyřešit. Pokud navýšíme šířku elementů na hodnotu `274px`, elementy se již nevejdou na řádek a navigace se zalomí na dva řádky. Jediným řešením je ústupek z šířky `900px`.

[4)] Definice špatného CSS kódu neexistuje. Obecně je za špatný CSS kód považován kód, který je redundantní a špatně se udržuje. Naopak správný CSS kód je popsán celou řadou CSS metodiky (například BEM metodika).

[5)] Je zapotřebí si uvědomit, že se jedná o jeden tzv. CSS pixel, který se ve skutečnosti zobrazí v závislosti na použitém zařízení jako více hardwarových pixelů.

Dalším problémem tohoto řešení je nevyváženost elementů obsahujících hlavní a doplňkový obsah stránky. Velikost těchto elementů je nyní určena jejich obsahem (nyní obsahují pouze element `p` s textem). Pokud jednomu z nich změníme (zvětšíme) obsah, bude tento element vyšší než druhý element. Ačkoliv to nemusí být patrné, jedná se o zcela zásadní problém. Pokud bychom chtěli, aby tyto elementy měly nějaké pozadí, a chtěli bychom layout využívat univerzálně, tedy například obsah těchto elementů by se načítal dynamicky z databáze, tak tyto elementy nebudou nikdy stejně vysoké. V takovém případě by se na stránce objevilo bílé místo pod elementem s menší výškou tak, jak je to ukázáno na následujícím obrázku.




Tento problém se opět nedá jednoduše vyřešit. Nastavení minimální výšky elementů pomocí `min-height` nelze použít, pokud by byl obsah jednou hodně dlouhý a jindy zase velice krátký. Jediným způsobem je vytvoření pomocného rodičovského elementu, který bude obsahovat námi požadované pozadí ve formě grafiky (například jednopixelový pruh, který se bude opakovat).

V neposlední řadě je celkové řešení pomocí obtékání obtížné pro nezkušené tvůrce webových stránek. Špatné ukončení obtékání často způsobuje v lepším případě deformaci celého layoutu, která se projeví okamžitě, nebo chyby v CSS, které nejsou vidět okamžitě, ale projeví se až v pozdější fázi, kdy je velice těžké je odhalit. Jednu z těch záludnějších chyb bychom mohli vytvořit změnou CSS selektoru `nav ul::after` na `nav::after`. Tím element `ul` získá výšku `0px`, jeho obsah je ale stále viditelný. Dodejme, že chyby při použití obtékání jsou jedny z nejčastějších chyb v CSS obecně.

### CSS FlexBox

Nyní se podíváme, jak nám mohou nové CSS specifikace usnadnit tvorbu layoutu a vyřešit výše popsané problémy. Nejprve si ukážeme řešení chybějícího pixelu v navigaci. K tomuto účelu nám poslouží CSS specifikace FlexBox. Jak její název napovídá, jedná se o způsob vytváření elementů s flexibilními rozměry. Rozměry za nás automaticky počítá webový prohlížeč, nemusíme se o ně proto starat. V našem případě upravíme následující dvě CSS pravidla, zbylá CSS pravidla zůstávají beze změny:

```
nav ul {
  padding-left: 0px;
  margin-bottom: 0px;
  display: flex;
}

nav ul li {
  float: left;
  box-sizing: border-box;
  outline: 2px dashed black;
  padding: 10px;
  list-style-type: none;
  flex-grow: 1;
}
```

Změna spočívá v přidání `display: flex;`. Nastavením této vlastnosti se říká, že daný element bude obsahovat elementy, jejichž rozměry budou počítány automaticky. V terminologii CSS FlexBox specifikace se jedná o tzv. kontejner. Jelikož v našem případě jsou všechny položky tohoto kontejneru stejně velké, určíme, že mají zabírat stejnou část pomocí vlastnosti `flex-grow: 1;`. Vlastnost `flex-grow` udává, o kolik se má zvětšit daný element oproti dalším elementům, které jsou spolu s ním rozmístěny v kontejneru. Pokud tedy máme 3 elementy s `flex-grow: 1;`, bude mít každý z nich nastavenu stejnou šířku. Pokud bychom například elementu obsahujícímu text `Odkaz 2` nastavili `flex-grow: 2;`, bude jeho šířka stejná, jako součet šířek zbylých dvou elementů.

Výsledná stránka je téměř identická s naším ukázkovým příklad. Rozdílem je to, že chybějící pixel není vůbec patrný. Výhodou tohoto řešení je značná univerzálnost. Pokud bychom se rozhodli změnit jakýkoliv rozměr, všechny zbylé rozměry budou automaticky přepočítány. V původním řešení by bylo zapotřebí tyto rozměry přepočítat manuálně.

Tímto ale CSS FlexBox zdaleka nekončí. Umožňuje například změnu pořadí elementů, různá zarovnání, a to včetně horizontálního zarovnání na střed, které bez této specifikace nelze udělat, a další. Přehled všech možností je k dipozici např. v [1]. Pro úplnost dodejme, že po výše popsané úpravě CSS pravidlo

```
nav ul::after {
  content: '';
  clear: both;
  display: block;
}
```

již nemá význam a můžeme ho smazat.

FlexBox nám může pomoci vyřešit i druhý problém (nevyváženost elementů) v původním kódu. Toto řešení uvedeme pouze pro zajímavost. Později uvidíme, že FlexBox se pro tento účel příliš nehodí.

Nejdříve je zapotřebí přidat do HTML kódu nový element, který bude představovat kontejner a bude obsahovat elementy `main`, `aside` a `footer`. Pro tento účel použijeme element `div`, který umístíme bezprostředně za element `header`. Následně upravíme CSS pravidla takto:

```
main {
  flex-basis: 580px;
}

aside {
  flex-basis: 290px;
}

footer {
  flex-grow: 1;
}

header + div { /* nový element */
  display: flex;
  flex-wrap: wrap;
  justify-content: space-between;
}
```

Nový element nám poslouží jako kontejner, nastavíme mu tedy vlastnost `display: flex;`. Elementům `main` a `aside` určíme jejich velikost pomocí vlastnosti `flex-basis`, která udává šířku elementů. Díky této

vlastnosti nebude šířka elementu vypočítána automaticky. Kontejneru nastavíme vlastnost `justify-content: space-between;`, která udává, že první položka kontejneru nebude mít levý okraj a poslední položka kontejneru nebude mít pravý okraj. Velikost zbylých mezer bude vypočítána automaticky. Elementy `main` a `aside` jsou nyní vyvážené a jejich výška se automaticky počítá podle většího z nich. Stejně jako v předchozím případě je patička stránky přes její celou šířku, čehož je docíleno pomocí `flex-grow: 1;`. Patička stránky je umístěna pod elementy `main` a `aside`, jelikož tyto elementy vyplňují celou šířku stránky (mají nastavenou hodnotu `flex-basis`). K zalomení obsahu kontejneru dojde automaticky.[6] Kompletní ukázka použití CSS FlexBox specifikace je k dispozici na stránce https://codepen.io/trnecka/pen/RXjZZp.

### CSS Grid

Obecně je FlexBox výhodné použít vždy, když potřebujeme automaticky určit velikost nebo rozmístění elementů na webové stránce. V případě nevyváženosti elementů `main` a `aside` je použití FlexBoxu poněkud násilné. Tyto elementy mají pevnou šířku a automaticky je počítána pouze mezera mezi těmito elementy a šířka elementu `footer`, která by ze své podstaty být počítána nemusela.

Mnohem lepší řešení nám nabízí specifikace CSS Grid, která umožňuje vytvoření layoutu webové stránky pomocí rozmístění prvků do dvoudimenzionální mřížky. V dřívějších dobách se často pro vytvoření layoutu webové stránky používala tabulka (element `table`), do jejichž políček se vkládaly jednotlivé prvky stránky. Tento způsob tvorby[7] je již dlouhou dobu překonaný a navíc v dnešní době nesprávný. Nicméně myšlenka rozmístit elementy do políček mřížky je elegantní a především v klasické grafice a typografii. Bez nadsázky je CSS Grid specifikace považována za jednu z nejlepších CSS specifikací vůbec, protože přináší něco, co bylo vždy potřeba, něco co skutečně funguje.

Základem práce s CSS Grid je vytvoření dvourozměrné mřížky a následné rozmístění prvků do této mřížky. Způsobů, jak vytvořit mřížku,

[6] Pro řízení zalomení obsahu kontejneru slouží vlastnost `flex-wrap`, která má implicitně nastavenu hodnotu `wrap`. Proto tuto vlastnost nemusíme v CSS kódu uvádět.

[7] Dodejme, že použití tabulky pro layout stránky bylo motivováno potřebou vytvářet komplikované layouty. K tomuto účelu neměl být element `table` nikdy použit. Přestože to specifikace jazyka HTML nedoporučovala, stal se tento způsob jedním z nejpopulárnějších vůbec.

nabízí specifikace opravdu velké množství a to včetně jejího automatického generování. Náš ukázkový příklad bychom mohli upravit následovně:

```
body > div {
  width: 900px;
  margin: auto;
  font-size: 1.2rem;
  display: grid;
  grid-template-columns: 580px 290px;
  grid-template-rows: auto;
  grid-column-gap: 10px;
}
```

Elementu `body > div` nastavíme vlastnost `display: grid;`, čímž určíme, že tento element bude obsahovat mřížku. Mřížku následně určíme pomocí `grid-template-columns: 580px 290px;` a `grid-template-rows: auto;`. Tímto vytvoříme mřížku, která bude mít dvě buňky na řádku o příslušných velikostech a tolik řádků kolik bude zapotřebí (hodnota `auto`). Dále určíme mezery mezi jednotlivými políčky `grid-column-gap: 10px;`. Tím máme mřížku vytvořenou a můžeme ji použít.

Nyní můžeme rozmístit do mřížky jednotlivé elementy. Například následující CSS kód říká, že element `header` bude zabírat celý řádek naší mřížky:

```
header {
  grid-column-start: 1;
  grid-column-end: 3;
  grid-row-start: 1;
  grid-row-end: 2;
}
```

V tomto případě se odkazujeme na jednotlivé čáry, které mřížku tvoří. Ty jsou automaticky číslovány zleva doprava (svislé čáry) a shora dolů (vodorovné čáry). Navíc je možné použít i zpětné indexování. Například `grid-column-end: -1;` odkazuje na svislou čáru, která je nejvíce vpravo. Totéž platí pro vodorovné čáry. Zápis si můžeme usnadnit pomocí

```
header {
  grid-column: 1 / 3;
  grid-row: 1 / 2;
}
```

nebo

```
header {
  grid-area: 1 / 1 / 2 / 3;
}
```

Všechny tři způsoby jsou ekvivalentní.

CSS specifikace umožňuje jednotlivé čáry pojmenovat a v zápisu používat jejich jména, což je praktické především u komplikovaných layoutů. Navíc je možné pojmenovávat jednotlivé oblasti a odkazovat se na ně. Tento způsob použití CSS Grid patří mezi nejpoužívanější. Zkusíme jej tedy použít v našem případě. CSS kód upravíme takto.

```
body > div {
  width: 900px;
  margin: auto;
  font-size: 1.2rem;
  display: grid;
  grid-template-columns: 580px 290px;
  grid-template-rows: auto;
  grid-column-gap: 10px;
  grid-template-areas: "hlavicka␣hlavicka"
                       "hlavni␣doplnkovy"
                       "paticka␣paticka";
}
```

`grid-template-areas` umožňuje pojmenovat jednotlivá políčka mřížky. Stejné názvy pak určují, že políčka tvoří jednu větší oblast. Na tyto oblasti je možné se odkázat pomocí vlastnosti `grid-area`. Jednotlivé prvky rozmístíme následovně:

```
main {
  grid-area: hlavni;
}

aside {
  grid-area: doplnkovy;
}

footer {
  grid-area: paticka;
}
```

Elementy `main` a `aside` jsou vyvážené, jelikož tvoří řádek mřížky, jehož výška je určena větším z nich. Ukázka použití CSS Grid na našem příkladu je k dispozici na stránce https://codepen.io/trnecka/pen/aeVVbY.

Celkově je rozmístění elementů do mřížky velice jednoduché. Ta nejzajímavější část CSS Grid specifikace se týká vytváření mřížky. Úplný přehled o možnostech lze nalézt v [2].

### Pár praktických poznámek

Ukázali jsme, že CSS FlexBox bylo možné použít pro rozmístění všech elementů na stránce. Stejně tak je možné použít CSS Grid a rozmístit všechny prvky. V našem případě by ale toto řešení bylo obtížné, jelikož šířka položek v navigaci není shodná s šířkou hlavního ani doplňkového obsahu. Není tedy možné je umístit do pravidelné mřížky. To je ale naprosto v pořádku. Specifikace CSS FlexBox a CSS Grid nejsou určeny k tomu, aby se pomocí nich vytvářel celý layout stránky a standardně se používají v symbióze. Zatímco CSS Grid je výhodné použít pro rozmístění hlavních částí stránky, CSS FlexBox je výhodné používat na obsah těchto hlavních částí. Ideální řešení našeho příkladu využívá CSS FlexBox pro navigaci stránky a CSS Grid pro celkový layout stránky.

Přirozenou otázkou je, zda má v dnešní době stále své uplatnění starý způsob používající vlastnost `float`. Pravdou je, že pro vytváření layoutu se tento způsob používá stále méně a během pár let pravděpodobně zcela vymizí. Na druhou stranu je však umístění prvku pomocí `float` nenahraditelné a CSS FlexBox nebo CSS Grid jej nedokáží zastoupit. Typickým příkladem jsou situace, kdy chceme umístit plovoucí obrázek do textu.

Jako poslední zmiňme použitelnost specifikací. CSS Flexbox i CSS Grid jsou dokončené specifikace, které mají dobrou podporu[8)] v moderních webových prohlížečích. Jediným zásadnějším problémem je podpora těchto specifikací v prohlížeči Internet Explorer 11, který patří stále k používaným prohlížečům. Ačkoliv tento prohlížeč obě specifikace podporuje, implementace CSS FlexBox obsahuje řadu chyb a v případě CSS Grid je podporována pouze starší verze specifikace. Dodejme, že všechny ukázky prezentované v článku především z důvodů jejich jednoduchosti, jsou funkční i v tomto prohlížeči.

Na závěr ještě uveďme, že v přípravě je již druhá verze specifikace CSS Grid [5], která bude mimo jiné umožňovat vnořit jednu mřížku do druhé.

[8)] Aktuální stav je k dipozici na https://caniuse.com/.

## Závěr

Ukázali jsme, jak jednoduchým a minimalistickým způsobem použít dvě nové specifikace při návrhu základního layoutu webové stránky. Vše bylo demonstrováno na univerzálním a v praxi často používaném příkladu tak, aby bylo možné snadno zařadit specifikace CSS FlexBox a CSS Grid do výuky. Pro úplnost dodejme, že soudobý web design je běžně používá. Z tohoto důvodu si obě specifikace zaslouží zařadit do výuky základů tvorby webových stránek, byť nebudou plnohodnotně popsány všechny jejich rozsáhlé možnosti.

## Literatura


[1] https://css-tricks.com/snippets/css/a-guide-to-flexbox/

[2] https://css-tricks.com/snippets/css/complete-guide-grid/

[3] https://www.w3.org/TR/css-flexbox-1/

[4] https://www.w3.org/TR/css-grid-1/

[5] https://www.w3.org/TR/css-grid-2/