Stylování v React

React je JavaScriptová knihovna pro tvorbu uživatelského rozhraní webových aplikací. Ve svém řešení ale tak troch zapomíná na css. Pro React neexistuje jednotný způsob jak přistoupit k stylování. Existuje však mnoho oddělených proudů které tuto problematiku řeší zcela odlišnými způsoby.

# External css

Využité knihovny:

* Classnames (není nutné)

Prvním způsobem, jak stylovat webovou aplikaci napsanou v React je vytvořit jeden nebo více externích css souboru které se naimportují přímo do app.js. A to takto“ import './App.css';“ Samotná aplikace se pak stylujete jako běžná webová stránka. Na každý html tag se odkazuje převážně pomocí atributu class (v React classname) nejlépe s pomocí nějaké stylovací konvence jako například BEM (Block-Element-Modifier). Styl je tedy celý oddělený od samotné aplikace.

[ukázka kodu]

## Konvence

Při tomto způsobu stylování React aplikací se finální css soubory stávají velkými a často i nepřehlednými. Protože se veškeré styly aplikují globálně muže docházet k neočekávanému chování a chybám které je těžké najít. Proto je dobré při pojmenovávat třídy pomocí některé z široce používaných konvenčních metod jako je například bem.

### BEM v React

Bem staví na OOCSS tedy odděluje styl od struktury, a proto by se mělo dbát na nízkou specifičnost selektorů. Rozděluje webovou stránku na bloky a elementy ze kterých se skládají. Syntaxe pojmenování tříd vypadá takto pro blok „.nazev\_bloku“ a takto pro element „.nazev\_bloku\_\_nazev\_elementu“.Při pojmenování elementu tedy vždy přidáme i název jeho rodičovského bloku oddělený dvěma podtržítky. Jak blok, tak element mohou mít i vlastní modifikátory které mění jejich chování. Název modifikátoru se přidává oddělený dvěma spojovníky. Například tedy „.nazev\_bloku\_\_nazev\_elementu—nazev\_modifikatoru“.

## Zhodnocení

I přestože demo aplikace, kterou jsem styloval byla velmi jednoduchá ke konci bylo někdy těžké se v celém css souboru vyznat. Bem se pro React velmi hodí a funguje v něm celkem intuitivně vzhledem k tomu, že se pracuje už tak se stránkou rozdělenou na komponenty. Interaktivní prvky se daly krásně zvládnout pomocí knihovny Classnames, ale někdy jsem byl nucen napsat i inline styl. Osobně si myslím že tento přístup je nejvhodnější při práci na velmi malých projektech vzhledem k tomu, že se nemusíte učit nic nového a dají se tak dobře zvládnout media queries. I tak bych doporučil alespoň CSS modules.

### Výhody

* Práce media queries
* Nemusíte se učit nic nového
* BEM funguje velmi dobře s filozofii React

### Nevýhody

* Nefunguje přirozeně s js
* Velmi špatná škálovatelnost
  + Všechno je globálně aplikované
  + Ve velkých souborech je těžké se vyznat

# CSS Modules

Využité knihovny:

* Classnames (není nutné)

Další nativní možností, jak stylovat React aplikaci je pomocí css modules. Css modules nám umožnuje aplikovat styl pouze lokálně na komponenty v souboru na který je importován. React css modules se importují do každého souboru komponenty, na kterou se styl má aplikovat pomocí import, from statementu takto „import styles from "../styles/NazevSouboru.module.css";“. samotné styly se píší jako klasické css, ale do atributu classname v reactu odkazujeme na blok stylu pomocí „className={styles.nazevAtributuClass}“. tento zápis se ve finálním html interpretuje jako „NazevSouboru\_nazevAtributuClass\_\_hash“.

## Konvence

Abychom dokázali využít všech výhod které css modules nabízejí, je potřeba dodržovat pár základních pravidel. Soubory css, které jsou určeny požít jako modules se běžně pojmenovávají nazev.modules.css, aby bylo jasné že se nejedná o klasické css určené k aplikování globálně. Dále je vhodné využít komponentové struktury reactu a ke každému komponentu přiřadit právě jedem css module styl. Tak je potom jasné, jaký styl patří, k jakému komponentu. Při pojmenovávání class není potřeba dodržovat BEM vzhledem k tomu, že pro každý komponent máme vlastní soubor a lokální scope. problémy, které bem řeší nikdy nenastanou a je proto redundantní. Naopak při pojmenovávání nedoporučuji používat podtržítko pro čistotu zápisu ve konečné interpretaci css modulu.

## Zhodnocení

Css modules řeší mnoho problémů které vznikali se stylováním pomocí externího css. Aplikují styl pouze na komponent, do kterého se importují a pracuje se především s menšími soubory. Obecně má tento přístup lepší škálovatelnost a nebál bych se ho použít i ve větších projektech. Vzhledem k tomu že se stále píše klasické css je velmi jednoduché si tento způsob stylování osvojit a pokud nechcete věnovat čas učit se knihovnu Styled Components jedná se určitě o způsob, jak stylovat React Aplikaci.

### Výhody

* Jednoduchá práce s malými soubory, ve kterých se dá velmi dobře orientovat.
* nedochází k neočekávanému chování a chybám kvůli duplicitním názvům.
* Bem není potřeba.
* Výborná škálovatelnost

### Nevýhody

* Nefunguje přirozeně s js
* U menších projektů horší práce z media queries

# Styled Components

Využité knihovny:

* Styled Components

Trochu jiný přístup ke stylování React aplikací je takzvané „css in js“ které spojuje css se zbytkem React. Tento způsob není nativně podporovaný js ani reactem jako předchozí dva způsoby proto je potřeba využít externí knihovny která pomůže styl implementovat do aplikace tímto způsobem. Existuje mnoho různých knihoven. Já jsem vybral tu nejoblíbenější, a to Styled Components, které plně podporují veškeré funkce css.

Ve styled component místo vytváření klasických css souborů a aplikování stylu pomocí atributu classname, vytváříme úplně novou komponentu na základě jednoho html tagu. Této nové komponentě definujeme styl již v js. To nám dává možnost css zkombinovat s js a vytvořit tak dynamické chování stylu přímo v css kódu. Tak komponenta muže měnit styly na základě atributů a jejich hodnot, které se komponentě přiřadí. tímto způsobem se nahradí všechny tagy ke kterým chceme přiřadit styl.

[ukázka kódu]

Styled components přinášejí nový způsob zápisu. Nejprve se vytvoří nová konstanta, která bude držet komponentu. Samotná komponenta se tvoří pomocí styled.[HTML tag]`[klasické css deklarace]`;. Kde si určíme na základě jakého tagu chceme komponentu stavět a string s již klasickými css deklaracemi které se na komponentu aplikují. Pomocí ${} můžeme do tohoto stringu psát další js. Atributy zpřístupňujeme klasicky pomocí parametru props Css z funkcí se vrací pomocí „return css `[klasické css deklarace]`;“.

styled components můžeme exportovat importovat pracovat s nimi jako z běžnými react komponenty. To nám dává možnost vytvořit si Styled components které následně použijeme na více místech v kodu.

## Konvence

Styled components přinášejí mnoho řešení na problémy spojené se stylováním React aplikace ale také mnoho způsobů jak dělat věci špatně. Existuje pár obecně uznávaných příkladů jak na to.

Jedním z prvních pravidel je vyhnout se psaní selektorů do styled components a prokaždý element který chceme stylovat napsat vlastní styl. Tyto nastylované komponenty poté píšeme do stejného souboru jako React komponent který stylujeme. Pokud budeme chtít styled component použít ve více souborech(React komponentech) vytvoří se soubor styles.jsx ve kterém budou napsany pouze všechny tyto styly. Když se tyto styly importují, lze použít klasické import { nazevkomponenty } from './styles'; avšak obecně se preferuje import \* as S from './styles'; aby se styled components lépe odlyšily od React komponentů.

## Zhodnocení

Styled components je velmi zajímaví přístup k řešení celého problému stylování. Jedná se o nejpokročilejší metodu s uplně jinou filozofii a přístupem k problemu. Pokud se člověk chce věnovat tvorbě aplikací v reactu myslím že se vyplatí tuto metodu naučit. Sc celkově navazuje na filozofii reactu a rozšiřuje ji i do css což se velmi hodí hlavně pokud chcete mít velmi interaktivní styly a nebo při práci na rozsáhlých projektech

### Výhody

* Spojuje css sReactem v jeden celek
* Řeší duplicitu kodu
* Výborná škálovatelnost

### Nevýhody

* Zavádí nový přístup ke stylování a musí se mu věnovat trochu času aby se s tím člověk naučil pracovat

# Bootstrap

Využité knihovny:

* Bootstrap
* Reactstrap

Bootstrap je knihovna stylů a komponent k vytvoření webových stránek bez nutnosti psaní vlastního css. Tato knihovna obsahuje vše od nastylovaných tlačítek, navbarů, tabulek, karet atd… . Samotný bootstrap se na web aplikuje pomocí tříd. Pro použití bootstrapu v reactu je možné využít knihovny Reactstrap, která staví na bootstrapu již před vytvořené nastylované React komponenty. Ty dále můžeme nastavit pomocí atributů a nebo využít bootstrap napřímo pomocí classnames.

Ukázka kódu

Všechny reactstrap komponenty a jejich použití naleznete na webové stránce <https://reactstrap.github.io/?path=/story/home-installation--page>

a dokumentaci k bootstrapu zde

<https://getbootstrap.com/docs/5.1/getting-started/introduction/>

## zhodnocení

Velmi vhodný pro tvorbu jednoduchých prototypů, nebo pro aplikace kde nepotřebujete originální stylování. Jedna se o knihovnu komponent to znamená že u výsledné aplikace člověk nemá kontrolu nad jejím výsledkem. Samozřejmě v některých případech je potřeba použít inline styl nebo trochu improvizovat aby člověk docílil požadovaného vzhledu.

### Výhody:

* Jednoduché rychlé vytvoření základně nastylované stranky.

### Nevýhody:

* Někdy je nutné použít inline styl
* Nemáte plně pod kontrolou vysledny vzhled, a aplikace bude vypadat genericky.