

Střední průmyslová škola strojní   
a elektrotechnická a Vyšší odborná škola, Liberec 1, Masarykova 3

Možnosti stylování v React

Maturitní práce

Autor **Štěpán Klein**

Obor **Technické lyceum**

Vedoucí práce **Ing. Tomáš Kazda, DiS.**

Školní rok **2021/2022**

Anotace

Práce se zabývá tvorbou dokumentace procesu tvorby uživatelského rozhraní v aplikaci na platformě React. Zaměřuje se na porovnání nejčastějších způsobů stylování. Tyto přístupy jsou prezentovány na čtyřech variantách té samé ukázkové aplikace.

Summary

The work deals with the creation of documentation of the user interface creation process in the application on the React platform. It focuses on comparing the most common ways of styling. These approaches are presented on four variants of the same sample application.

Čestné prohlášení

Prohlašuji, že jsem předkládanou maturitní práci vypracoval sám a uvedl jsem veškerou použitou literaturu a bibliografické citace.

V Liberci dne

Štěpán Klein

Obsah

[Úvod 1](#_Toc97755825)

[1 Frameworky a knihovny pro vývoj single-page webových aplikací 2](#_Toc97755826)

[1.1 React 2](#_Toc97755827)

[1.1.1 Komponenty 2](#_Toc97755828)

[1.1.2 Virtuální DOM 3](#_Toc97755829)

[1.1.3 Hooks 3](#_Toc97755830)

[1.2 Angular 3](#_Toc97755831)

[1.3 Vue.js 3](#_Toc97755832)

[2 Stylování v React 4](#_Toc97755833)

[2.1 Externí css 4](#_Toc97755834)

[2.1.1 Styl psaní 4](#_Toc97755835)

[2.2 CSS Modules 5](#_Toc97755836)

[2.2.1 Styl psaní 5](#_Toc97755837)

[2.3 CSS-in-JS 5](#_Toc97755838)

[2.3.1 Styled Components 6](#_Toc97755839)

[2.3.2 Ostatní knihovny 7](#_Toc97755840)

[2.4 Bootstrap 7](#_Toc97755841)

[3 Tvorba demo aplikace 8](#_Toc97755842)

[3.1 Návrh 8](#_Toc97755843)

[3.2 Funkční část 9](#_Toc97755844)

[3.3 Stylování jednotlivých verzí 10](#_Toc97755845)

[3.3.1 Externí css 11](#_Toc97755846)

[3.3.2 Css modules 12](#_Toc97755847)

[3.3.3 Styled components 13](#_Toc97755848)

[3.3.4 Bootstrap 14](#_Toc97755849)

[4 Dokumentace 16](#_Toc97755850)

[4.1 Struktura dokumentace 16](#_Toc97755851)

[4.2 Tvorba stránky 16](#_Toc97755852)

[Závěr 18](#_Toc97755853)

[Seznam zkratek a odborných výrazů 19](#_Toc97755854)

[Seznam obrázků 20](#_Toc97755855)

[Použité zdroje 21](#_Toc97755856)

[A. Seznam přiložených souborů I](#_Toc97755857)

Úvod

V rámci maturitní práce se rozhodl zdokumentovat a předvést nejznámější způsoby stylování webových aplikací vytvořené na platformě React. Osobně jsem se rozhodl pro tuto práci hlavně abych si rozšířil mé znalosti o dané problematice a Reactu samotném. Vybral jsem si čtyři konkrétní zpusoby jak nastylovat webovou aplikaci. Každý z těchto zpusobů jsem si nastudoval a vytvořil jsem verzi vybrané demoaplikace abych získal i zkušenosti z praktického použití. Mnou nově nabité zkušenosti jsem se dále rozhodl sdílet s ostatními studenty Střední průmyslové školy strojní a elektrotechnické a Vyšší odborné školy v Liberci prostřednictvím dokumentace. V této práci se věnuji jak teorii spojené se stylováním v React, tak i popisu tvorby samotné dokumentace a demo aplikací.

# Frameworky a knihovny pro vývoj single-page webových aplikací

Webové aplikace často musejí interagovat s uživatelem a serverem. Cílem Single-page přístupu k webové aplikaci je interagovat s uživatelem rychleji a přirozeněji. To se děje tak že se stránka v prohlížeči neobnovuje, ale načítá pouze nová a potřebná data ze serveru pomocí jiných metod (nejčastěji JSON data.), nebo je generuje lokálně.

Při vývoji webové aplikace je téměř nezbitné usnadnit si práci pomocí frameworku, který pomáhá implementaci chování popsané výše. I přesto že celá práce se věnuje pouze jedné knihovně konkrétně knihovně React, některé z přístupů stylování popsaných dále by se dali použít i v rámci jiných frameworků a knihoven podobného typu.

## React

React je JavaScriptová knihovna pro tvorbu uživatelského rozhraní webových aplikací vyvíjená firmou Meta (dříve Facebook). React mění strukturu psaní webových stránek tím že HTML a JavaScript (nebo TypeScript) spojuje v JSX (TSX) a samotný web následně rozděluje do komponent. React je pouze základním stavebním blokem pro vývoj webové aplikace a častokrát je nutnost doplnit o knihovny třetích stran.

### Komponenty

Celá webová aplikace vytvořená v React se skládá z komponent. V rámci každé komponenty se řeší jak její logika, tak i její HTML struktura. Tato struktura se v rámci JSX tvoří kódem podobnému HTML, ale místo tagů můžeme vkládat i jiné komponenty. Existují dva hlavní typy React komponent. První je definována pomocí třídy, což využívá nativní objektové orientace jazyka JavaScript. Druhý rozšířenější způsob je komponenta definovaná pomocí funkce. Tato funkce má jeden parametr „props“ a vrací výše zmíněnou strukturu. Z té pak vychází HTML kód, který se bude renderovat uživateli. Pomocí parametru props můžeme předávat data z rodičovských komponentů na potomky ve formě podobající se HTML atributů. Ve zbytku dokumentu budu referovat hlavně na komponenty definované funkcemi, ale principy spojené se stylováním jsou téměř stejné v obou případech.

const Komponenta = (props) => {

//JS kód

return (

//Struktura (HTML + Komponenty)

);

}

#### Životní cyklus komponent

Každá komponenta má třífázový životní cyklus, a to vytvoření komponenty (Mount), změny stavu nebo props komponenty (Update) a odebrání komponenty (Unmount). Pokud je potřeba navázat nějakou logiku na životní cyklus komponenty využívá se hook *useEffect*.

#### Stav komponenty

Stavem komponenty referujeme na určitý typ proměnných v komponentě. Pokud se některé z těchto proměnných změní hodnota, ergo změní se stav komponenty, komponenta se znovu vygeneruje. Pokud chceme vytvořit tento typ proměnné ve komponentě využijeme React hook funkci *useState*. Hook *useState* přímá jako parametr vychozí hodnotu stavu a vrací pole se dvěma prvky. První prvek nese současnou hodnotu stavu a druhý je funkce, která slouží ke změně stavu. Tato funkce přímá pouze jeden parametr, a to novou hodnotu funkce.

const [variable, setVariable] = useState(0);

### Hooks

React hooks jsou funkce pomocí, kterých můžeme navazovat na různé vlastnosti Reactu, zejména měnit stav komponent a navazovat na jejich životní cyklus nebo kontext. React hooks jsou určeny pouze pro komponenty založené pomocí funkce (komponenty definované pomocí třídy fungují jinak). A dají se použít pouze v top-levelu dané komponenty. Pokud je potřeba použít kod založený na hooks ve více komponentech je možné vytvořit si vlastní „custom hook“.

const Komponenta = (props) => {

//Zde mohu použít hooks

if(true){

//Zde už ne

}

return (

);

}

### Virtuální DOM

JavaScript používá k referování na HTML, DOM. DOM je reprezentace HTML struktury webové stránky. V případě že je obsah na stránce aktualizován je nutné aby byl aktualizován i DOM, to se však s narůstající velikostí a komplikovaností HTML struktury stránky stává velmi náročný úkol, který muže trvat relativně dlouho zpracovat. Aby se zajistila plynulost chodu webové aplikace, která neustále mění obsah na stránce je potřeba tento problém vyřešit.

Virtuální DOM je řešení Reactu na tento problém. Jedná se o virtuální reprezentaci DOM uloženou v paměti. Když se má obsah na stránce změnit nejprve se aktualizuje virtuální DOM, jehož změny nejsou tak náročné na zpracování. Aktualizovaný Virtuální DOM se následně porovná se předchozí verzí a pouze nalezené změny se aplikují na reálný DOM.

### Kontext

Kontextem v React referujeme na další způsob, jak v rámci komponent předávat data. Pokud je potřeba předat data z rodičovského komponentu na potomka obyčejně se využívají props. Takto lze velmi jednoduše předat data přímému potomkovi, problém však nastává, pokud chceme zpřístupnit ta samá data velkému množství komponentů i hluboko ve struktuře aplikace. V těchto případech lze využít React kontext, který nám umožní sdílet data a zpřístupnit kód nehledě na strukturu komponent. Vytvořený kontext můžeme v komponentě zpřístupnit pomocí hooku *useContext*.

### Tvorba React projektu

Prvním krokem při vývoji React aplikace je vytvořit samotné prostředí do kterého se již budou vkládat jednotlivé komponenty. Vývojové prostředí pro React lze vytvořit pomocí závislosti Create-React-App, díky které se vše nastaví pomocí jediného příkazu *npx create-react-app Nazev-aplikace*. Tento příkaz získá a aplikuje potřebné knihovny, vytvoří všechny potřebné soubory, a dokonce aplikuje i git. Celý projekt je následně připravený k vývoji, testování aplikace a samotné sestavení aplikace pro produkci.

## Angular

Angular je framework určený pro TypeScript vyvíjený firmou Google. Vychází ze starší, ale stále využívané JavaScriptové verze Angularu AngularJS. Angular Je založen stejně jako React na komponentech a však práce s nimi je už odlišná.

## Vue.js

Vue.js je taktéž komponentově založený framework, který čerpá mnoho funkcí od Reactu i od Angularu. I přesto že za vue.js nestojí žádná velká společnost, dokáže ostatním zmíněným platformám velmi dobře konkurovat a používá mnoho pokročilých technologii jako virtuální DOM

# Stylování v React

Pro React neexistuje jednotný způsob, jak přistoupit k stylování. Existuje však mnoho oddělených proudů, které tuto problematiku řeší zcela odlišnými způsoby. Nejprve jsem si vybral čtyři velmi oblíbené přístupy ke stylování React aplikace. A to konkrétně:

* Externí CSS s pomocí BEM
* CSS Modules
* CSS in JS (Styled Components)
* Externí knihovna stylů (Bootstrap)

Pro tyto jednotlivé způsoby jsem si musel nastudovat jak fungují, principy a konvence, které se používají při jejich aplikování.

## Externí CSS

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.

### Styl psaní a 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“*.

## CSS Modules

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. Samotné styly se píší jako klasické CSS, ale do atributu classname v JSX souborech odkazujeme na blok stylu pomocí názvu referující na modul, tečky a názvu atributu class. tento zápis se ve finálním html interpretuje jako „*NazevSouboru\_nazevAtributuClass\_\_hash*“.

### Styl psaní a 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.

## CSS-in-JS

Trochu jiný přístup ke stylování React aplikací je takzvané „CSS-in-JS“, které spojuje CSS se zbytkem React. Tento způsob není nativně podporovaný JavaScript 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.

### Styled Components

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 JavaScript. To nám dává možnost CSS zkombinovat s JavaScript 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.

Styled Components přinášejí nový způsob zápisu. Nejprve se vytvoří nová konstanta, která bude držet komponentu. Při tvorbě samotné komponenty 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 JavaScript.

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 kódu anebo klidně vlastní knihovnu/katalog komponentů, které můžeme využívat ve více projektech.

#### Styl psaní a 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 pro každý 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 napsaný 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 odlišily od React komponentů.

### Emotion

Knihovna Emotion není pouhá React dependence, ale lze požít i samostatně. V rámci Reactu podporuje dvě metody používaní. Ta první je velmi podobná knihovně Styled Componets, má téměř stejný syntax i způsob použití. Druhá metoda vytváří nastylovanou komponentu pomocí CSS propu velmi podobně jako inline styl, ale s rozšířenou syntaxí a přehlednějším kódem.

### JSS

Podobné jako Emotion lze použít i mimo React. V rámci Reactu funguje trochu odlišně jak předchozí dvě zmíněné metody CSS-in-JS knihovny. V rámci stylované komponenty se vytvoří konstanta *useStyles* pomocí funkce *createUseStyles*. Tato funkce přímá jako parametr JSON objekt obsahující jednotlivé styly. Tyto styly se skrz zmíněnou konstantu následně vkládají do JSX html pomocí atributu classnames.

## Bootstrap

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 a mnoho dalších. Samotný Bootstrap se na web aplikuje pomocí atributu class. Pro použití Bootstrapu v Reactu je možné využít knihovny Reactstrap, která staví na Bootstrapu již předem vytvořené nastylované React komponenty. Ty dále můžeme nastavit pomocí atributů a nebo využít Bootstrap napřímo pomocí atributu classname. Při práci s Reactstrapem je důležité nezapomenout naimportovat i samotný bootstrap.

K práci s knihovnami Reactstrap a Bootstrap je vhodné využít jejich oficiální dokumentaci.

# Tvorba demo aplikace

Pro tvorbu demo aplikací jsem si zvolil čerpat z již hotové aplikace, která je vyžívaná při výuce na škole <https://github.com/TomasKazda/react-sandwich-feast>. Tato aplikace byla již nastylována pomocí knihovny Bootstrap. Vzhledem k tomu že aplikace již byla nastylovana jedním ze způsobů, který v práci pokrývám a že její rozsah mi nedovoloval použít všechno co jsem potřeboval při stylování vyzkoušet, rozhodl jsem se k úpravám nad rámec stylování. A to konkrétně jsem nahradil komponent Navigation.jsx komponenty Leaderboard.jsx a Placement.jsx dále jsem upravil ostatní komponenty tak aby byli v souladu s návrhem.

## Návrh

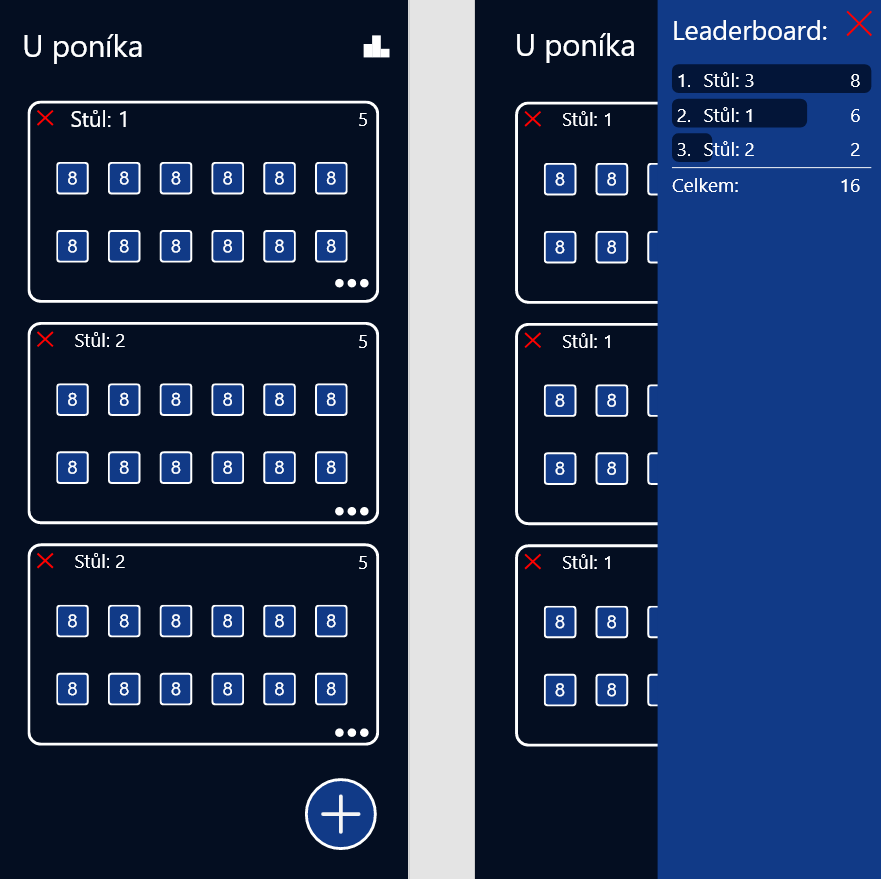
Návrh jsem tvořil v programu Adobe XD. Při tvorbě návrhu bylo důležité myslet na to, aby se dalo využít co nejvíce různých vlastností CSS. Protože bylo potřeba abych si udělal představu, jak vhodně půjdou použít jednotlivé přístupy stylování v různých situacích.

Abych využil mediaqueiris rozhodl jsem se vytvořit responzivní web, pro který budu psát styl pomocí metodiky mobile-first. Pro potřeby tvorby tohoto typu webu jsem si navrhl dvě verze, a to verzi pro telefon a pro desktop. Finální design pro ostatní velikosti a formáty vycházejí právě také z těchto dvou návrhů. Můj návrh obsahuje i vlastní ikony ve formátu SVG, které jsou ve finální aplikaci obohaceny animacemi.

Při návrhu byl zvolen převážně modrý barevný profil, design se zaoblenými hrany a netradičně tlustými linkami.



Obrázek Návrh designu pro desktop



Obrázek návrh designu pro smartphone

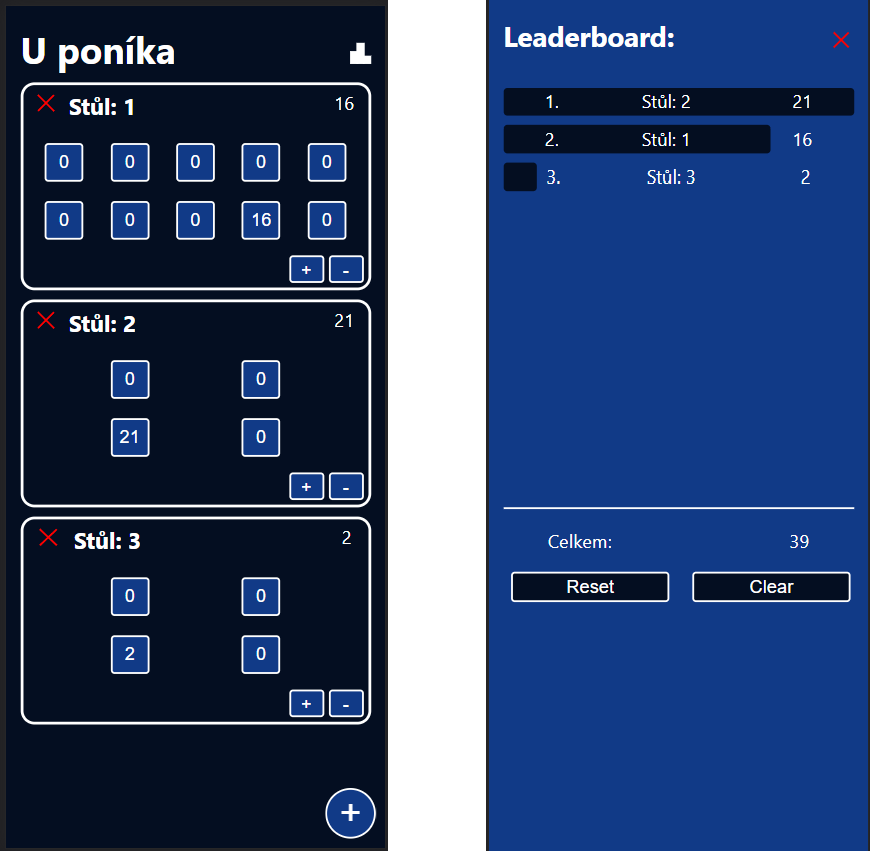
## Funkční část

Aplikace React-sandwich-feast je jednoduché počítadlo zkonzumovaných produktů (sendvičů, piv nebo cokoli jiného) u stolů v rámci jednoho podniku. Celá aplikace manipuluje s daty pomocí jednoho 2d pole. Každý stůl je reprezentován polem číselných hodnot udávající počet zkonzumovaných produktů na jednom místě v rámci stolu. Všechny stoly jsou elementy dalšího pole. Číslo stolu je jeho index v rámci pole plus jedna (stul číslo jedna je elementem nula v poli). Manipulace a distribuce dat v rámci tohoto pole skrz aplikaci je zprostředkovaná pomocí kontextu. Samotná finální aplikace se skládá z pěti funkčních React komponent:

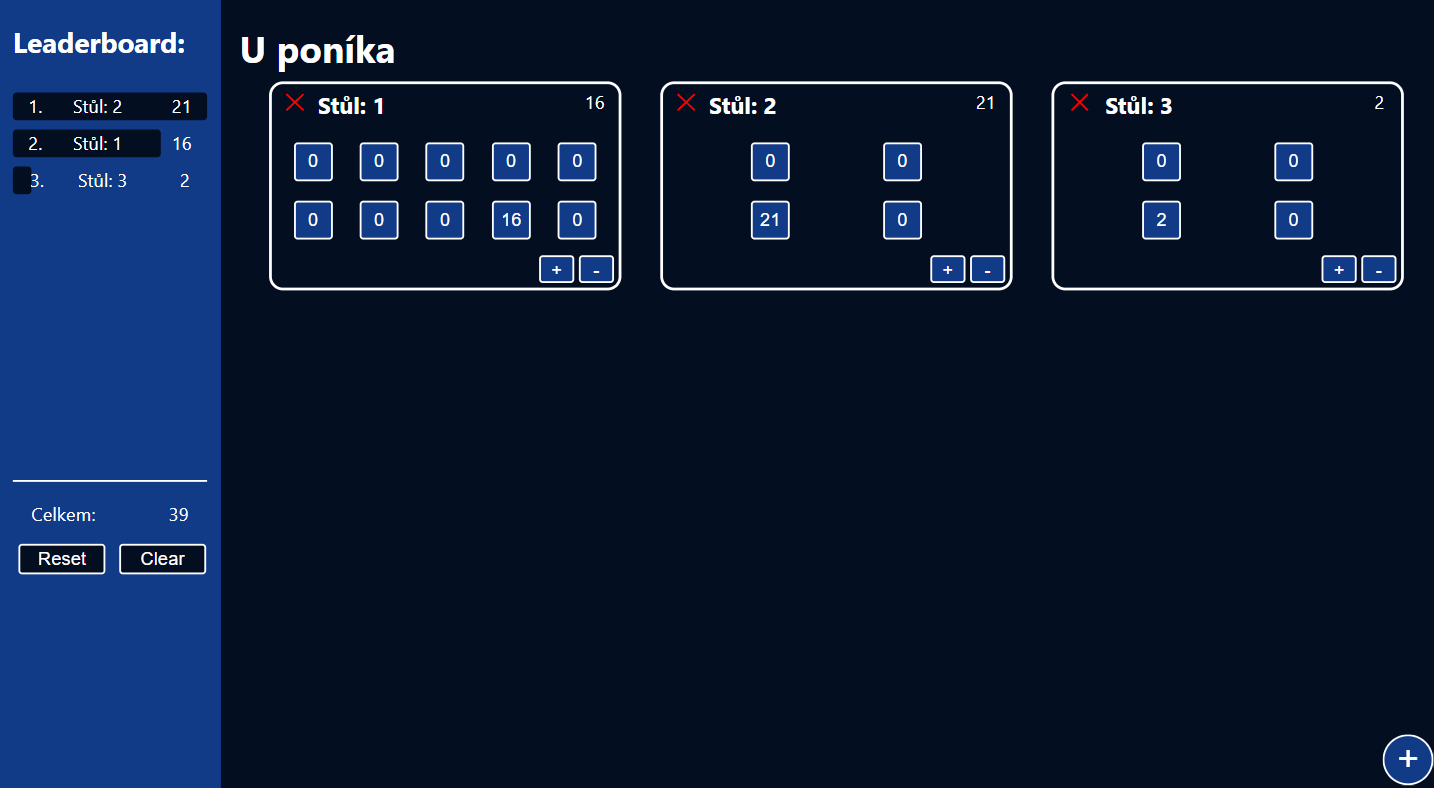
* **Seat** – Drží hodnotu zkonzumovaného produktu jedním účastníkem. Při kliknutí se hodnota zvýší o jedna.
* **Table** – Jedná se o kartu reprezentující jeden stůl. Tato komponenta se vždy skládá z několika komponentů Seat, které zde můžeme libovolně přidávat a ubírat. Dále zde vidíme počet zkonzumovaných produktů na stůl a můžeme i daný komponent Table odebrat pomocí křížku.
* **Tables** – Tato komponenta se nachází v aplikaci pouze jednou a drží všechny Table komponenty. Taká lze pomocí tohoto komponentu přidávat nové stoly.
* **Placement** – Každá komponenta Placement reprezentuje každý stul v rámci leaderboardu aplikace. Tento komponent upravuje svoje pozdí v závislosti počtu zkonzumovaných produktů v poměru k nejlepšímu pro vizuální znázornění.
* **Leaderboard** –Drží komponenty Placement a stará se o jejich správné zařazení dle zkonzumovaných produktů. Dále obsahuje tlačítka pro reset a vynulování celé aplikace

## Stylování jednotlivých verzí

Poté jsem se postupně věnoval stylování jednotlivých verzí demo aplikace. Při stylování všech verzí jsem se řídil s konvencemi popsanými v druhé kapitole tohoto dokumentu a postupoval dle metodiky Mobile-first. V rámci metodiky mobile-first se nejprve styluje forma pro telefony a následně se tento styl upravuj v závislosti na velikosti obrazovky na postupně větší verze. Tato logika se aplikuje v CSS pomocí media queris. To znamená že aplikace z hlediska Reactu funguje stejně na všech typech zařízení a jediné co se mění je styl.



Obrázek finální vzhled aplikace na smartphonu



Obrázek finální vzhled na desktopu

### Knihovna Classnames

Knihovnu Classnames používám jak v Externí CSS tak v CSS modules verzích demo aplikace. Tato knihovna umožnuje jednouší a čistší zápis podmíněně aplikovaných hodnot v rámci atributu class. V tomto případě se běžně používá ternární operátor (? :). Právě pomocí knihovny Classnames můžeme nahradit ternární operátory zjednodušenou syntaxí. Toto je ve výsledném kódu přehlednější hlavně v případě využití více podmíněně aplikovaných hodnot atributu class.

//Řešení pomocí ternárního operátoru

<div className={`nepodmíněně\_aplikovaná\_třída ${hodnotaBool ? 'podmíněně\_aplikovaná\_třída' : null}`}>

​

//Řešení pomocí knihovny Classnames ve verzi External CSS

<div className={classNames("nepodmíněně\_aplikovaná\_třída", {podmíněně\_aplikovaná\_třída: hodnotaBool})}>

### Externí CSS

Jako první jsem styloval aplikaci jako klasický web tedy pomocí externího CSS souboru s dodržením konvence BEM.

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. Ukázka kódu:

return (

<div className='tables'>

<h1 className='tables\_\_h1'>{state.name}</h1>

<div className='tables\_\_grp'>

{tablesGrp}

</div>

<button className='tables\_\_btn' onClick={e =>

dispatch({type: ADD\_TABLE})

}>+</button>

</div>

);

A CSS

.table{

position: relative;

width: 375px;

height: 220px;

border: 3px solid rgb(255, 255, 255);

border-radius: 15px;

color: rgb(255, 255, 255);

margin: 10px auto;

}

.table\_\_head{

display: flex;

height: 32px;

justify-content: space-evenly;

margin-bottom: 20px;

margin-top: 5px;

}

### CSS modules

Druhou verzi jsem styloval pomoci CSS modules, které řeší mnoho problémů vzniklé 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. Ukázka kódu:

return (

<div className={styles.tables}>

<h1 className={styles.h1}>{state.name}</h1>

<div className={styles.grp}>

{tablesGrp}

</div>

<button className={styles.btn} onClick={e =>

dispatch({type: ADD\_TABLE})

}>+</button>

</div>

);

.tables{

margin: 0px;

margin-bottom: 80px;

margin-top: 20px;

}

.grp{

width: 100%;

}

### Styled Components

Styled Components je velmi zajímaví přístup k řešení celého problému stylování. Jedná se o nejpokročilejší metodu s úplně jinou filozofii a přístupem k problému. Pokud se člověk chce věnovat tvorbě aplikací v React myslím, že se vyplatí tuto metodu naučit. Styled Components celkově navazuje na filozofii React a rozšiřuje ji i do CSS což se velmi hodí hlavně pokud chcete mít velmi interaktivní styly a nebo při práci na rozsáhlých projektech. Ukázka kódu:

const Rectangle2 = styled.rect`

width: 9.076px;

height:23px;

${(props) => {

if(props.cross){

return css`

transform: translate(388.882px, 20px) ;

fill: rgba(0,0,0,0);

`;

}

else{

return css`

transform: translate(388.882px, 40px);

fill: #fff;

`;

}

}

}`;

### Bootstrap

Při stylování této verze nešlo docílit stejného designu jako u ostatních verzí. Tento způsob je 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 kompletní kontrolu nad jejím výsledkem. Samozřejmě v některých případech je potřeba použít inline styl nebo trochu improvizovat, aby člověk docílil požadovaného vzhledu. Ukázka kódu:

return (

<Container >

<h1 >{state.name}</h1>

<CardGroup>

{tablesGrp}

</CardGroup>

<Button onClick={e =>

dispatch({type: ADD\_TABLE})

}>+</Button>

</Container>

);

# Dokumentace

Po dokončení jednotlivých verzí demo aplikace bylo na řadě zdokumentovat mnou nově získané zkušenosti a publikovat je prostřednictvím SharePoint Pages. Výhodou této dokumentace nad většinou podobných zdrojů je přístup ke zdrojovým kódům jedné aplikace nastylovanou všemi čtyřmi způsoby ze, kterých dokumentace vychází. Díky tomu muže kdokoli porovnat jednotlivé přístupy na rozsahu celého projektu, nebo si ujasnit detaily, které ani nemohou být v dokumentaci popsány.

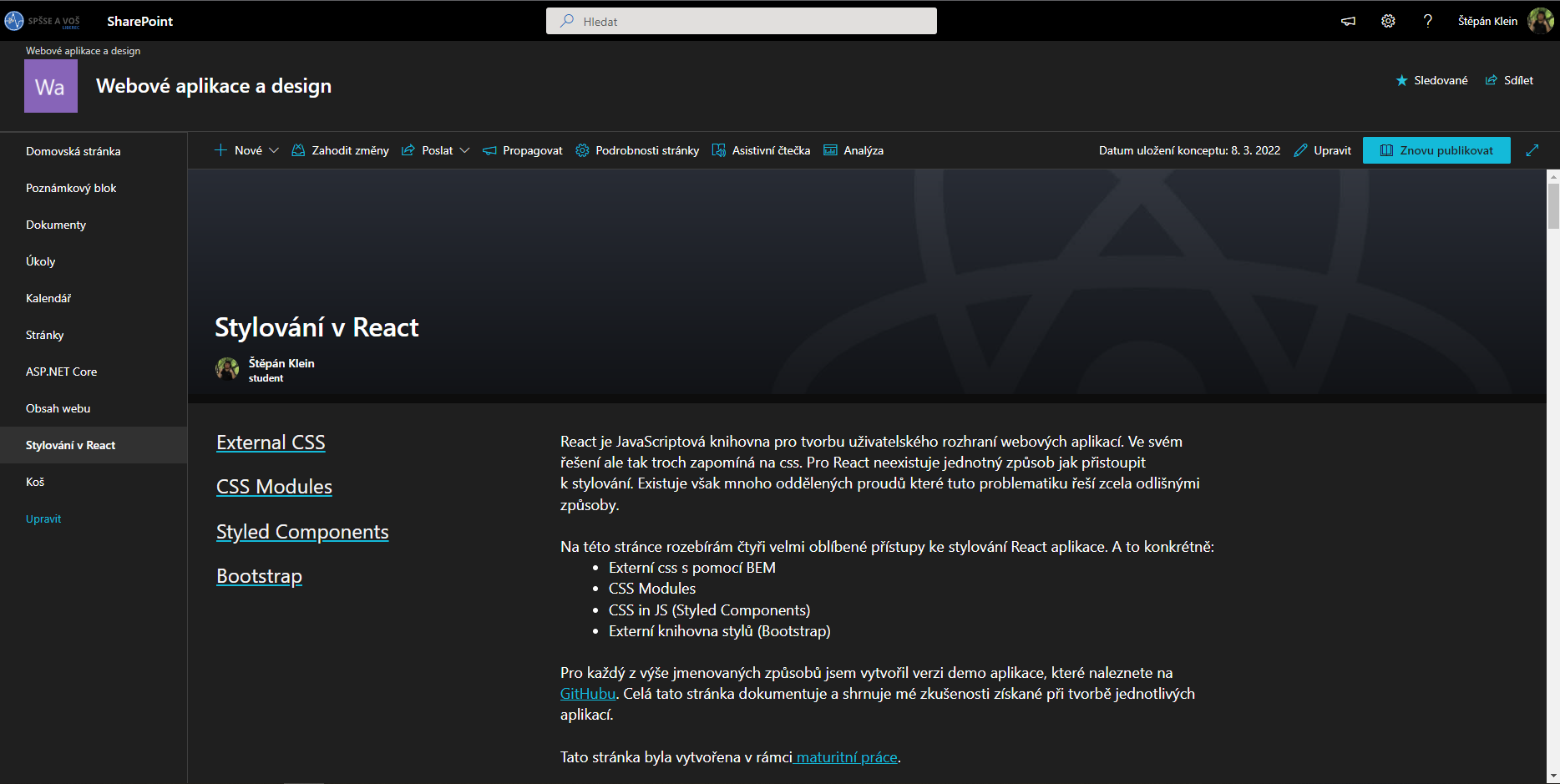
## Struktura dokumentace

Vytvořená SharePoint stránka slouží pouze jako úvod do problematiky stylování React aplikací. Popisuje a porovnává oblíbené technologie a přístupy k této problematice. Dokumentace předpokládá předchozí zkušenosti s vývojem webových stránek a základní znalosti React na straně čtenáře. V případě potřeby odkazuje na oficiální dokumentaci knihoven a zdrojové kódy demo aplikací

Po krátkém úvodu do problematiky se v dokumentaci jednotlivě věnuji každému z vybraných způsobů stylování aplikace. U každého způsobu uvádím použité knihovny, základy psaní stylu, konvence a styl psaní. Zde vycházím z druhé kapitoly (Stylování v React) tohoto dokumentu. Dále v závěru již prezentuji můj osobní názor na způsob stylování a uvádím výhody a nevýhody každého přístupu ke stylování React aplikace. V celé dokumentaci se nachází ukázky kódu, na, kterých je prezentovaná přesná syntaxe každé z metod.

## Tvorba stránky

Tvorba stránky prostřednictvím SharePoint pages je velmi jednoduchá a intuitivní. Celý SharePoint web se tvoří pomocí funkčních bloků. Tyto bloky už drží samotný obsah stránek ať už jde o text, ukázky kódu, obrázky a mnoho dalšího. Stránky jsou k nalezení pro všechny studenty na školním SharePoint ve stránkách Studijních materiálu pro webové aplikace a design. V rámci těchto stránek existuje již několik jiných kurzů, abych zachoval jednoty vzhled rozhodl jsem se využít stejné šablony.



Obrázek – Finální vzhled SharePoint stránek dokumentace.

Závěr

V rámci této se jsem zdokumentoval stylování v React a vytvořil ukázky správného postupu, které jsou k dispozici na GitHubu. Zaměřuji se hlavně na čtyři odlišné a široce používané metody. Sám jsem chtěl prohloubit své znalosti v práci s reactem. To se povedlo a díky dokumentaci ve, které své poznatky prezentuji a poskytuji jako zdroj informací pro lidi kteří již mají nějaké základní zkušenosti s tvorbou webů. Hlavní výhodou této dokumentace je přístup ke zdrojovým kódům jedné aplikace nastylovanou všemi čtyřmi způsoby, což umožnuje lidem čerpajícím z tohoto zdroje prohlédnout si prezentovaná řešení aplikované v praxi.

Seznam zkratek a odborných výrazů

HTML

HyperText Markup Language – značkovací jazyk používaný pro tvorbu webových stránek.

CSS

HyperText Markup Language – značkovací jazyk používaný pro tvorbu webových stránek.

JS

HyperText Markup Language – značkovací jazyk používaný pro tvorbu webových stránek.

JSON

HyperText Markup Language – značkovací jazyk používaný pro tvorbu webových stránek.

OOCSS

HyperText Markup Language – značkovací jazyk používaný pro tvorbu webových stránek.

SVG

HyperText Markup Language – značkovací jazyk používaný pro tvorbu webových stránek.

JSX

HyperText Markup Language – značkovací jazyk používaný pro tvorbu webových stránek.

DOM

HyperText Markup Language – značkovací jazyk používaný pro tvorbu webových stránek.

Seznam obrázků

[Obrázek 1 Návrh designu pro desktop 7](#_Toc97667144)

[Obrázek 2 návrh designu pro smartphone 8](#_Toc97667145)

[Obrázek 3 finální vzhled aplikace na smartphonu 9](#_Toc97667146)

[Obrázek 4 finální vzhled na desktopu 10](#_Toc97667147)

[Obrázek 5 – Finální vzhled SharePoint stránek dokumentace. 16](#_Toc97667148)

Použité zdroje

1. **Meta Platforms, Inc.** *React.* [Online] [Citace: 20. Únor 2022.] https://reactjs.org/.

2. **Wikimedia Foundation, Inc.,.** React (JavaScript library). *Wikipedia.* [Online] [Citace: 20. Únor 2022.] https://en.wikipedia.org/wiki/React\_(JavaScript\_library).

3. **Michálek, Martin.** BEM: Pojmenovávací konvence pro třídy v CSS. *Vzhůru dolů.* [Online] [Citace: 22. Únor 2022.] https://www.vzhurudolu.cz/prirucka/bem.

1. Seznam přiložených souborů