Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Tutoriál pro tvorbu webové stránky prostřednictvím knihovny React.js

Bakalářská práce

2024 Petr Gajdoš

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2022/2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Petr Gajdoš**
Osobní číslo:	**I20087**
Studijní program:	**B0688A140009 Informační technologie**
Téma práce:	**Tutoriál pro tvorbu webové stránky prostřednictvím knihovny React.js**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování

Bakalářská práce bude zaměřena na vytvoření webové stránky s využitím knihovny React.js. Práce bude obsahovat jednotlivé kurzy. Každý tento kurz se bude zabývat jednotlivými tématy. Prostřednictvím každého kurzu bude čtenářovi vysvětlena určitá látka. U každého kurzu bude zároveň daná látka demonstrovaná na příkladu. Kromě jednotlivých kurzů, bude v práci zahrnuta i odborná část, tedy důvody, proč využívat knihovnu React.js, její výhody atd. Výsledkem bakalářské práce bude webová stránka, ve které budou uplatněny jednotlivé lekce z bakalářské práce.

Rozsah pracovní zprávy: **min. 30 stran**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

WIERUCH, Robin. *The Road to React: Your journey to master plain yet pragmatic React.js*. 2021. Berlin: Independently published, 2021. ISBN 9781720043997.

LNC, Men a Emma WILLLIAM. *React js: Learning React js Library From Scratch*. 1. Independently Published, 2020. ISBN 9798698196686.

Vedoucí bakalářské práce: **Ing. Jan Panuš, Ph.D.**
Katedra informačních technologií

Datum zadání bakalářské práce: **16. prosince 2022**
Termín odevzdání bakalářské práce: **12. května 2023**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

Ing. Jan Panuš, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 28. února 2023

Prohlašuji:

Práci s názvem Tutoriál pro tvorbu webové stránky prostřednictvím knihovny React.js jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 7/2019 Pravidla pro odevzdávání, zveřejňování a formální úpravu závěrečných prací, ve znění pozdějších dodatků, bude práce zveřejněna prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 06. 05. 2024

Petr Gajdoš

PODĚKOVÁNÍ

Tímto bych rád poděkoval Ing. Janu Panušovi, Ph.D., za vedení mé bakalářské práce a poskytnutí cenných rad pro vytvoření této práce.

ANOTACE

Tato bakalářská práce se zabývá seznámením s knihovnou React, která slouží pro tvorbu webových stránek. Výsledkem této práce je webová stránka, která představuje nástroj správy rybářské organizace. V teoretické části jsou postupně přestaveny základní nástroje knihovny React společně s příkladem využití.

KLÍČOVÁ SLOVA

React, HTML, webová stránka, komponenta, JSX

TITLE

Tutorial for creating websites using the React.js library

ANNOTATION

This bachelor's thesis deals with introduction to the React library which is used for creating websites. The result of this of this work is a website that can be use as a tool for fishing organizations managing. In the theoretical part, the basic tools of the the React library are gradually rebuilt together with an example of use.

KEYWORDS

React, HTML, website, component, JSX

OBSAH

SEZNAM ILUSTRACÍ A TABULEK

SEZNAM ZKRATEK A ZNAČEK

HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
JSX	JavaScript Syntax eXtension
NPM	Node Packge Manager
JSON	JavaScript Object Notation
DOM	Document Object Model
SPA	Single Page Application

ÚVOD

Existuje mnoho způsobů a nástrojů pro tvorbu webových stránek. Neustále dochází k jejich vývoji a snaze zlepšit postup samotné tvorby za pomoci nových metod. Jedním z těchto nástrojů je i knihovna React.js, která představuje odlišný pohled na samotnou tvorbu stránek.

Cílem této bakalářské práce je seznámení čtenáře s tvorbou webových stránek pomocí knihovny React. V práci je postupně vysvětlována látka, u které dochází k uvedení příkladů aplikace látky. Tyto příklady jsou buď možností práce přímo s Reactem nebo demonstrační příklady definované pomocí obrázků. Samotné demonstrační příklady jsou obsáhlejší, zatímco příklady pomocí obrázku jsou uvedeny u takových částí látky, které jsou jednoduché. Příklady, které jsou uvedeny jako součásti demonstračního projektu umožní čtenáři vyzkoušet si práci v prostředí React a seznámit se s tvorbou složitějších komponent. Následně bude představena webová stránka, která vznikla na základě znalostí z předchozích kapitol samotného tutoriálu. Tato webová stránka bude sloužit pro správu rybářské organizace, kde bude možné získat informace o organizaci a nahlédnutí do rybářského řádu. Dále zde bude možné zobrazit si seznam lovných revírů a následně tvorba docházky člena organizace.

1 Předpoklady pro práci s knihovnou React

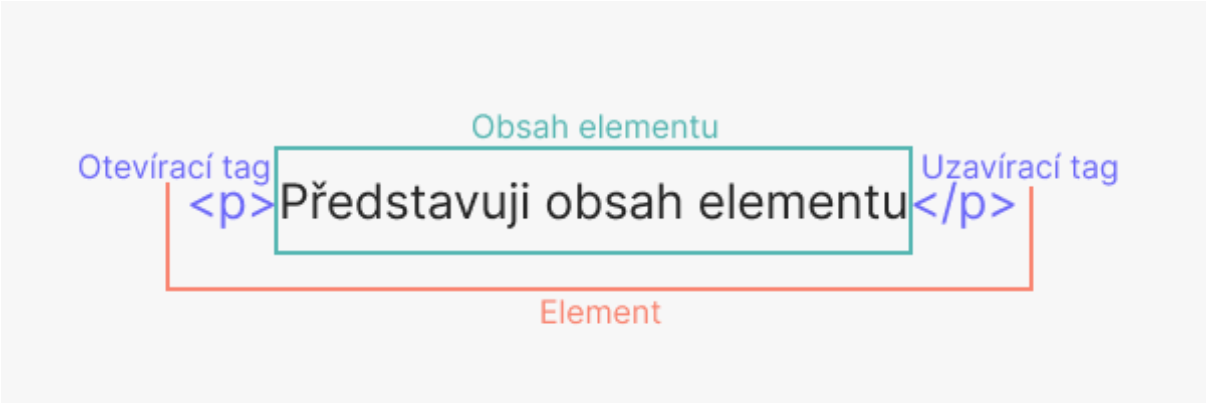
1.1 HTML

Jedná se o značkovací jazyk, který se používá pro definování struktury obsahu na webové stránce. Pro vytvoření struktury je možné využít z řady mnoha elementů. Pomocí elementů se určuje, jakým způsobem bude daný obsah zobrazen. [1]

Mezi dva základní kameny HTML se řadí již zmíněný element a následně tag. Samotný element se v základní podobě skládá z obsahu elementu a příslušného tagu. Pojem tag představuje způsob, jak prohlížeči říct, jakým způsobem má obsah zobrazit pro uživatele. [1; 2]

Element je možné následně doplnit o atributy. Ty se používají pro dodatečnou specifikaci elementu. Může se jednat například o atribut class nebo id. [1]

Pro získání představy, jak může základní element vypadat, je zde k dispozici demonstrační obrázek. [Obrázek 1] Na tomto obrázku je možné vidět element demonstrující odstavec textu. Skládá se z otevíracího a uzavíracího tagu. Tyto tagy slouží jako způsob pro vymezení hranice elementu. Poslední částí je už pouze samotný obsah elementu, který bude zobrazen uživateli.



Obrázek 1 - Ukázka základního HTML elementu (zdroj vlastní)

1.2 CSS

Kaskádové styly představují jazyk, pomocí kterého je možné snadné stylování elementů napsaných prostřednictvím značkovacích jazyků. Nejčastěji jsou spojovány s jazykem HTML, a to právě z důvodu absence přímého stylování uvnitř HTML souboru. Před vznikem kaskádových stylů disponovalo HTML pouze speciálním tagem font, který sloužil pro jednoduché stylování. Ve chvíli, kdy ovšem docházelo k rozsáhlejšímu stylování elementu, stala se tato možnost velmi nepřehlednou. Z tohoto důvodu vzniklo CSS. Samotný zápis kaskádových stylů je možné provést třemi způsoby. Tyto způsoby se nazývají Internal, External a Inline. [3]

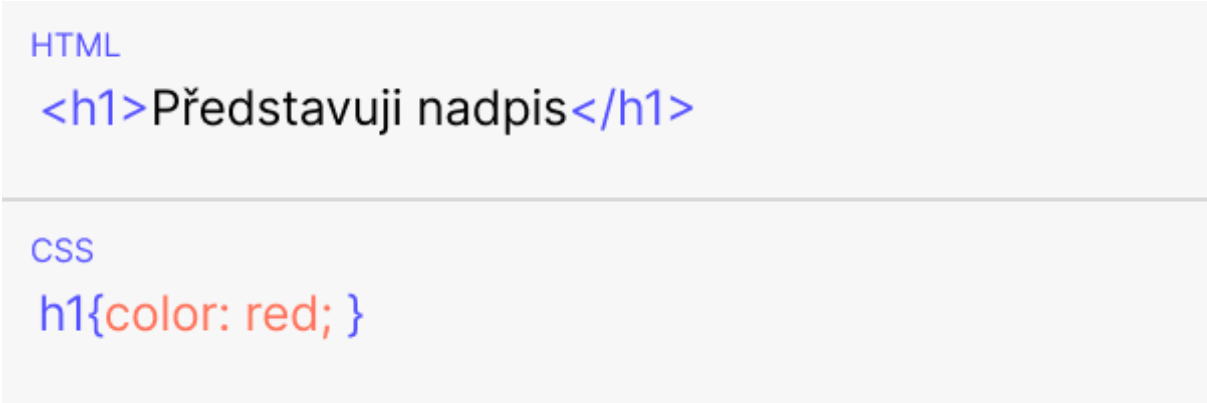
Prvním způsobem je External, tedy vytvoření samostatného souboru, který obsahuje všechna definovaná stylování. Po vytvoření tohoto souboru je nutné jeho přidání k příslušnému HTML souboru, kterému stylovaní náleží. [3]

Druhý způsob se nazývá Internal. Zapsání tohoto stylu je uvnitř samotného HTML souboru. K tomu je potřeba použít párový tag style, který se pro zachování přehlednosti nejčastěji přidává do hlavničky HTML souboru. Nevýhodou tohoto stylování je skutečnost, že ve chvíli

kdy dojde k obnovení stránky, dojde k opětovnému načtení stylů, to může způsobit delší dobu načítání samotné stránky. [3]

Poslední možností je Inline stylování, kde se využívá atribut style u elementu, kterému bude stylování náležet. Nevýhodou této možnosti je nutnost stylování každého elementu samostatně. [3]

Na obrázku [Obrázek 2] je možné vidět jednoduchý způsob uplatnění stylování na element, kdy je samotná struktura v souboru HTML a styl se nachází v souboru typu css. V ukázkovém případě bude při každém využití tagu h1 následně konečný výstup červené barvy.



Obrázek 2 - Ukázka stylováni pomocí CSS (zdroj vlastní)

Pomocí kaskádových stylů je možné měnit barvu elementu, styl písma, velikost elementů, ale také vytváření animací, efektů a mnohem více. [3] S rozvojem moderních zařízení je také nutné přizpůsobovat webové stránky tak, aby vždy došlo ke korektnímu zobrazení na všech zařízeních. K tomu je v kaskádových stylech možné využít media queries. Díky jejich pomoci lze velmi jednoduchým způsobem přizpůsobit webovou stránku pro zobrazení na více zařízeních tak, aby byly stránky vždy zobrazeny korektně.

1.3 JavaScript

Jedná se o programovací jazyk, který se používá pro vytváření dynamických webových stránek. Tvůrcem tohoto jazyka je Brendan Eich, který ho vytvořil v roce 1995. Oproti jiným programovacím jazykům se výhradně využívá v prohlížeči uživatele, a ne na straně serveru, nicméně samotné použití na straně serveru je také možné. Společně v kombinaci s jazyky HTML a CSS představují nejpoužívanější jazyky na internetu.[4]

Oproti ostatním programovacím jazykům je JavaScript spustitelný přímo uvnitř prohlížeče a před jeho samotným spuštěním dochází k převodu na jednoduchý strojový kód. Dále podporuje dynamické psaní, to znamená uživatel nemusí uvádět, jaký datový typ určitá proměnná představuje. Kompilátor si tuto skutečnost sám odvodí a díky tomu dochází k rychlejšímu zápisu kódu. [4]

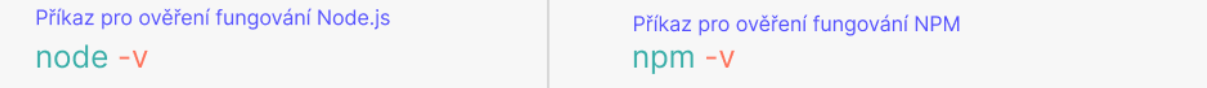
Pro práci je možné si vytvořit samostatný soubor, který bude obsahovat například deklarování proměnných nebo funkcí. Následně se tento soubor připojí k danému souboru HTML. Druhou možností, která se primárně využívá u psaní kratších jednodušších kódů je zápis JavaSriptu přímo uvnitř souboru HTML. K tomu je potřeba použít příslušný tag script. [4]

1.4 Node.js a NPM

Node.js představuje prostředí, prostřednictvím kterého je možné spouštět kód vytvořený pomocí JavaScriptu mimo webový prohlížeč. Je možné ho využít pro tvorbu webových aplikací. [5]

Pod pojmem NPM se skrývá správce pro balíčky typu JavaScript. Za pomoci tohoto správce lze jednoduchým způsobem instalovat balíčky a následně je i udržovat. Samotný NPM správce je volně k dispozici a jeho instalace probíhá současně s instalací Node.js. React využívá NPM pro instalaci dodatečných knihoven. [5]

Samotná instalace je provedena společně při instalaci prostředí Node.js. Pro ověření, zda je NPM a Node.js k dispozici, lze použít příkazy z obrázku [Obrázek 3] například v terminálu editoru.



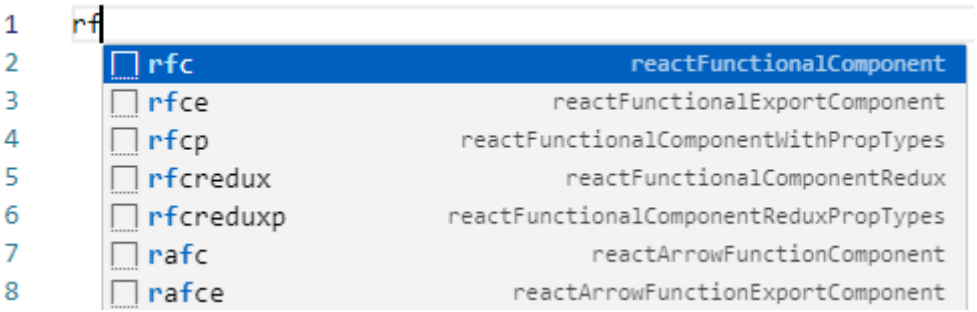
Obrázek 3 - Příkazy pro ověření instalace Node.js a NPM (zdroj vlastní)

1.5 Vývojové prostředí

Ačkoli nutnost vývojového prostředí není nezbytná, protože je možné psát samotný kód v běžném textovém editoru, je díky správnému vývojovému prostředí psaní kódu mnohem efektivnější a mnohem jednodušší na práci. Existuje velké množství těchto prostředí, mezi nejznámější lze uvést Microsoft Visual Studio Code, WebStorm nebo NetBeans. [6]

Microsoft Visual Studio Code je jeden z nejvíce oblíbených volně dostupných textových editorů. Jeho správu má na starosti společnost Microsoft. Umožňuje práci na operačním systému Windows, Linux i macOS. Mezi jeho přednosti lze uvést jednoduchou instalaci a následně poměrně malou velikost samotného editoru. Další vlastností editoru je podpora rozšíření. Rozšíření představuje dodatečné funkce třetích stran, které mohou být do editoru dodatečně přidány. [7]

Jedním z velmi užitečných rozšíření při práci s Reactem je ES7+ React/Redux/React-Native snippets. Toto rozšíření se používá pro zrychlení samotného postupu při tvorbě jednotlivých částí aplikace. Jedná se o zkratky, pomocí kterých lze při zapsání pouze několika písmen říct samotnému editoru, co má vytvořit. [8] Například pokud dojde k zápisu rf je možné si nechat vygenerovat functional komponentu. Příklad použití je možné vidět na obrázku [Obrázek 4]



Obrázek 4 - Ukázka práce s rozšířením ES7+ React/Redux/React-Native snippets (zdroj vlastní)

2 Knihovna React

React představuje JavaScriptovou knihovnu, určenou pro tvorbu uživatelských rozhraní nebo jejich částí nazývané jako komponenty. Byl vytvořen společností Facebook (v dnešní době Meta) a celkově s příchodem této knihovny změnil samotný pohled na tvorbu webových aplikací. Vymyslel ho tehdejší zaměstnanec Facebooku Jordan Walke. Sám React představuje jeden z nejoblíbenějších nástrojů pro tvorbu moderních webových aplikací. Pro samotnou tvorbu je možné postupně samotnou React aplikaci rozšiřovat o samostatné knihovny, které ulehčují vývojářům práci. Například knihovna nesoucí název React Router umožňuje pohodlné směrování uvnitř samotné aplikace. React je složen z jednotlivých komponent, kde každá komponenta představuje určitou část celku a má na starosti vždy jednu konkrétní činnost. Další odlišností od ostatních nástrojů je možné nalézt ve využití Virtual DOM, která umožňuje rychlejší a zároveň efektivnější práci při tvorbě aplikací. Poslední výhodou je samotná práce s Reactem, který pro zápis využívá jazyk JSX, který představuje kombinaci jazyka HTML a JavaScript. To umožňuje jednoduchým způsobem vytvářet aplikace s dynamickým obsahem a aktivním reagováním na uživatelské požadavky. [9]

2.1 Virtual DOM

Nejprve je nutné představit pojem DOM. Ten je možné nalézt na každé webové stránce a představuje samotnou strukturu jednotlivých částí HTML, které se na stránce nachází. Tyto části jsou sestaveny do hierarchie, kde každá představuje konkrétní uzel. Oproti tomu představuje Virtual DOM unikátní koncept, který obsahuje kopii skutečné DOM. Tato kopie je uložena v paměti a jejím cílem je zrychlit proces promítání změn. Tento proces probíhá následujícím způsobem. React nejprve provede příslušnou změnu v samotné Virtual DOM a teprve až následně provede změnu ve skutečné DOM. Přesněji řečeno, ve skutečné DOM nedochází ke kompletní aktualizaci, ale pouze jen takových částí, které nejsou stejné s Virtual DOM. Z důvodu, že je Virtual DOM uložena v paměti, je aktualizace mnohem rychlejší, než kdyby docházelo k opětovné aktualizaci celé skutečné DOM. Tento proces lze demonstrovat na příkladu, ve kterém dojde k přidání nového elementu do samotného projektu. V tu chvíli nastane aktualizace celé Virtual DOM, do které je přidán nový kořenový uzel s příslušným elementem. React má následně za úkol porovnat virtuální a skutečnou DOM a v místech kde se nebudou shodovat, provede aktualizaci ve skutečné DOM. [6]

2.2 JSX

Pokud vývojář začíná s tvorbou webových stránek, jako první se setká se způsobem definování samotné struktury stránky pomocí jazyka HTML. Následně po vytvoření struktury je možné využít kaskádové styly pro vytvoření vzhledu pro dané elementy. V poslední řadě je možné umožnit webu reagovat na interakce nebo manipulaci s daty. K tomu lze využít například programovací jazyk JavaScript. Všechny tyto zmíněné nástroje jsou pro větší přehlednost od sebe oddělené ve svých vlastních souborech. V Reactu se ovšem pro zjednodušení a větší přehlednost využívá kombinace jazyků HTML a JavaScript. Tato kombinace nese název JSX a představuje rozšíření jazyka JavaScript, který umožňuje zápis HTML uvnitř JavaScript souboru. [10]

Jak samotná práce s JSX probíhá, je možné demonstrovat na následujícím příkladu, ve kterém bude cílem vytvořit patičku webové stránky, která je graficky znázorněna na obrázku [Obrázek 5].

Obrázek 5 - Demonstrační patička webové stránky (zdroj vlastní)

Samotný zdrojový kód této komponenty je následně popsán na obrázku [Obrázek 6]. Postup pro vytvoření patičky začíná definováním komponenty. Dále je zde definování stylu samotné komponenty, který je ukládán do proměnné s názvem footerStyle. Tento styl je následně přiřazen příslušnému elementu. Potom zde dochází k využití funkce pro získání aktuálního datumu. Následuje samotný element, který bude výstupem komponenty.

```
import React from 'react'

function Footer() {
  //styl komponenty
    const footerStyle ={
        width: "100%",
        display: "flex",
        justifyContent: "center",
        backgroundColor: "var(--box-color-dark)",
        color: "white",
        marginTop: "auto"
    }
    //JavaScript proměnná pro získání aktuálního roku
    const currentYear = new Date().getFullYear();
  return (
  <footer style={footerStyle}>
    {/*HTML výstup doplněný s hodnotou currentYear*/}
    <p>© {currentYear} RYBARSKY ELEKTRONICKY SYSTEM</p>
  </footer>
  )
}

export default Footer
```

Obrázek 6 - Postup práce pro vytvoření patičky (zdroj vlastní)

2.3 Pravidla pro práci s JSX

Práce s JSX se může zdát jednoduchá, existují však určitá pravidla, která je potřeba dodržovat. Pokud by vývojář bez vědomí těchto pravidel do nově vytvořené komponenty, pouze vložil čisté HTML, setkal by se následně s mnoha chybami. Níže je možné vidět základní pravidla pro práci s JSX.

2.3.1 Návratový pouze jeden element

V případě vytváření obsáhlejších komponent, které budou tvořeny z více elementů, je nutné tyto elementy obalit jedním hlavním elementem. Pokud by toto pravidlo nebylo dodrženo, nebylo by možné komponentu vytvořit. Hlavním důvodem, proč nelze vracet více než jeden element, je správná funkčnost již dříve zmíněné Virtual DOM. Ve chvíli, kdy komponenta vrací více než jeden element, není možné přidat do stromové struktury hlavní kořenový uzel, protože není možné rozpoznat, který z elementů má tento uzel reprezentovat. [11; 10]

Pro obalení více elementů je možné zvolit více způsobů. Jedním z nejběžnějších je provést samotné obalení pomocí tagu div. Dále je možné využít mnoho další tagů, kterými disponuje HTML. Mezi další příklady můžeme zařadit tag header, footer, section.

V některých případech může nastat situace, kdy už dodatečné obalení do dalšího elementu by bylo zcela zbytečné. V takovém případě React disponuje další možností pro obalení více elementů. Tato možnost se nazývá React Fragment. Ten představuje možnost vrácení více elementů současně, bez nutnosti vytváření dodatečného prvku v samotné DOM. Fragment oproti tagu div efektivně šetří paměť, protože neobsahuje žádné další dodatečné vlastnosti a také vytváří přehlednější zápis samotného kódu. [12]

Obalení pomocí Fragmentu

```
return(
 <>
 <h1>Nadpis</h1>
 <p>Vnitřní text</p>
 </>
)
```

Obalení pomocí tagu Div

```
return(
 <div>
 <h1>Nadpis</h1>
 <p>Vnitřní text</p>
 </div>
)
```

Chybný zápis

```
return(

 <h1>Nadpis</h1>
 <p>Vnitřní text</p>

)
```

Obrázek 7 - Ukázka možností obalení více elementů (zdroj vlastní)

2.3.2 Využití className místo atributu class

Běžně se v HTML využívá atribut class při specifikování třídy pro konkrétní element. Díky tomu, lze například k danému element jednoduše přiřadit jeho stylování. Problém nastane ve chvíli, kdy je slovo class použito uvnitř JSX. JavaScript má totiž class jako rezervované klíčové slovo, a protože je JSX rozšířením JavaScriptu, používá se v Reactu className místo class. [11; 10] Na přiloženém obrázku [Obrázek 8] je demonstrován způsob správného použití className a následně chybného použití class.

Správný zápis pomocí className

```
<div className="name-of-class">Obsah elementu</div>
```

Chybný zápis pomocí class

```
<div class="name-of-class">Obsah elementu</div>
```

Obrázek 8 - Ukázka správného a nesprávného zápisu className (zdroj vlastní)

2.3.3 Uzavírání tagů

V jazyce HTML existují tagy, které není potřeba na jejich konci uzavírat. Typickým případem je tag input. V JSX je nutné vždy tagy uzavírat a nejjednodušším způsobem je zakončit tag lomítkem. Druhou a velmi užitečnou vlastností je uzavírání tagu, pokud neobsahuje žádný obsah. Pokud budeme používat element, který se skládá z otvíracího a zavíracího tagu, ale nebude mít žádný obsah. Není nutné tento element uzavírat klasickým uzavíracím tagem. Místo toho stačí element zakončit kombinací znaků lomítka a znaku většítka. [10; 11]

Příklad zápisu je možné vidět na obrázku [Obrázek 9].

Správný zápis elementu bez obsahu

```
<img src="image.jpg" alt="Image" />
```

Obrázek 9 - Ukázka uzavírání tagů (zdroj vlastní)

2.3.4 Zápis pomocí metody camelCase

Tato metoda představuje způsob zápisu pro víceslovné fráze. JSX využívá tuto metodu na pojmenovávání všeho. Základní využití nalezneme u pojmenovávání atributů, názvu proměnných nebo pro názvy obslužných funkcí. Příklad správného pojmenovávání nalezneme už u samotného atributu className, kde je tato metoda uplatněna. [10; 11]

2.3.5 Využití složených závorek pro zápis JavaScriptu

Ve chvíli, kdy je potřeba kromě statického obsahu zobrazovat i obsah dynamický, je vhodný důvod pro využití JavaSciptu, přesněji o určitý výraz. Tento výraz může být využit v mnoha případech. Ať už se jedná o podmíněné zobrazení obsahu, interakci s uživatelem nebo pro výpočetní operace. Aby toto bylo možné je potřeba v místě, kde se bude samotný výraz používat obalit ho do složených závorek. Pokud by se výraz nenacházel uvnitř složených závorek, došlo by k běžnému výpisu obsahu elementu. [10] Příklad je možné nalézt na obrázku [Obrázek 10].

Správný zápis s použitím {}

```
const nameFromVariable = "Alex";
<h1>{nameFromVariable}</h1>
```

Chybný zápis bez použití {}

```
<h1>nameFromVariable</h1>
```

Obrázek 10 - Ukázka zápisu JavaScriptu uvnitř JSX (zdroj vlastní)

3 Tvorba a práce s React aplikací

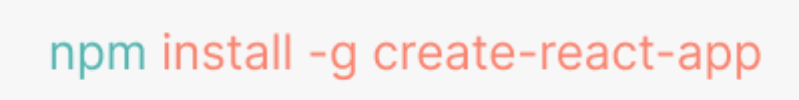
Před vytvořením nové aplikace v prostředí React je nutné mít na svém zařízení nainstalovaný Node.js společně s NPM. Dále vývojové prostředí, ve kterém bude samotná aplikace postupně vytvářená. [13] Všechny potřebné postupy pro instalaci zmíněných nástrojů je možné nalézt v příslušných kapitolách.

Vytvoření nového React projektu může být provedeno mnoha metodami. Jednou z nich je i možnost, která nese název Create React App. Jedná se oficiální a zároveň podporovanou možnost pro tvorbu nové aplikace. Jedním z hlavních aspektů, proč je tato možnost velmi používána, je schopnost vytvoření projektu bez nutnosti konfigurace. [14] K vytvoření nové aplikace se používá příkaz, který je k dispozici na obrázku [Obrázek 12].

Druhou velmi často používanou možností je tvoraba aplikace pomocí násrtoje Vite. Oproti předchozí možnosti je zde nutná vlastní konfigurace, která může být pro začátečníky složitá.

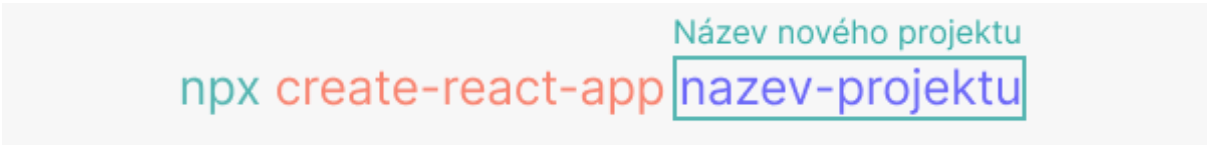
Na druhé straně se z této varianty stala velmi oblíbená záležitost oproti Create React App, hlavně kvůli rychlejšímu vytvoření nového projektu a její větší flexibilitě. [14]

V této práci bude pro svou jednoduchost využita možnost tvorby nové React aplikace pomocí metody Create React App. Pro možnost vytvářet aplikace pomocí této metody je nedříve nutná její instalace. Aby isntalace proběhla globálně a následně bylo možné vytvářet nové projekty v kterémkoliv adresáři, musí být pomocí terminálu editoru spuštěn příkaz z obrázku [Obrázek 11]. [13]



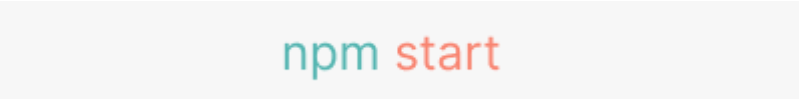
Obrázek 11 - Příkaz pro instalaci Create React App (zdroj vlastní)

Po úspěšné instalaci je možné vytvořit nový projekt. Nejprve je nutné vytvořit nový adresář, ve kterém se bude samotný projekt nacházet. Následně po úspěšném vytvoření adresáře se v otevřeném editoru pomocí terminálu je nutné přemístit do zmíněného nového adresáře. V příslušeném adresáři stačí už následně pouze spustit pomocí terminálu příkaz z obrázku [Obrázek 12] pro tvorbu React aplikace. Tato operace obvykle zabere delší chvíli. [13]



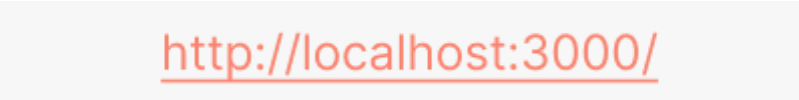
Obrázek 12 - Příkaz pro vytvoření nového projektu

V okamžiku kdy dojde k úspěšnému vytvoření nového projektu, je už pouze nutné spustit vývojový server. K tomu je potřeba se opět přesunout do dalšího adresáře. Tentokrát se jedná o samotný adresář nově vytvořeného projektu. V tomto adresáři je už pouze potřeba pomocí terminálu spustit příkaz pro spuštění serveru. Pro spuštění slouží příkaz uvedený na obrázku [Obrázek 13]. [13]



Obrázek 13 - Příkaz pro spuštění vývojového serveru (zdroj vlastní)

Po úspěšném spuštění samotného serveru dojde k automatickému otevření nové záložky uvnitř webového prohlížeče. Tato záložka bude obsahovat náhled samotné aplikace. [13] Pokud by k tomuto nedošlo, je možné nalézt spuštěnou aplikaci na adrese, která je vidět na obrázku [Obrázek 14].



Obrázek 14 - Ukázka adresy spuštěné aplikace (zdroj vlastní)

3.1 Seznámení se strukturou projektu

Po vytvoření nové aplikace pomocí metody Create React App, obsahuje samotný projekt už větší množství vytvořených souborů a adresářů. Znát doslovný význam každého není nutné, ale

je vhodné vědět základní informace, co představují. Prvním adresářem je node_modules, ve kterém se nachází všechny potřebné závislosti, které samotná aplikace po svém vytvoření potřebuje. Dále je zde soubor .gitcore. Tento souboru říká Gitu, které soubory mají být z pohledu Gitu ignorované. Následuje souboru package.json, který je velmi důležitý. Je možné v něm nalézt informace o tom, která rozšíření projekt využívá a pomáhá při nastavení projektu při přenosu na jiné zařízení. Zároveň se jedná o soubor, který je možné využít pro ověření, zda byla samotná rozšíření přidána úspěšně. Dále soubor README.md obsahuje pokyny, jak se samotnou aplikací pracovat a které základní příkazy využívat. [15]

Důležitou roli má adresář s názvem public. Ten obsahuje všechny veřejné soubory jako samotnou favi ikonu aplikace. Poté hlavní soubor index.html, který je podle potřeby naplňován komponentami. Poslední je soubor manifest.json, který slouží pro definici samotné aplikace. [15]

Veškerá práce je zaměřena na adresář s názvem src, ve kterém se nachází a následně jsou do tohoto adresáře přidávané postupně vytvářené soubory, ze které se aplikace skládá. Ve chvíli samotného vytvoření se už v tomto adresáři nachází určité soubory. Nejdůležitější je index.js, který představuje vstup do celého projektu. Poté App.js je první jednoduchá komponenta, která představuje samotný kořen celé aplikace. Tento soubor je možné nahradit vlastním, ale v mnoha případech se ponechává. V tomto souboru dochází následně například k definování cest pro směrování. Dále jsou zde soubory typu css, které obsahují stylování a soubor App.test.js, který je používán pro jednoduché testování. [15]

React sám o sobě neurčuje podle jakého standardu by se měla struktura adresářů vytvářet. Ve chvíli, kdy dojde k vytvoření nového projektu se vývojář převážně času nachází v adresáři src. Do této složky bude postupně přidávat své vlastní komponenty, zdroje a mnoho dalšího. Samotný adresář je ovšem v základu velmi primitivní a ve chvíli kdy bude obsahovat více souborů, stane se také velmi nepřehledným. Existuje mnoho odlišných návodů na správnou podobu přehledné struktury souborů. Záleží však na každém vývojáři, který bude upřednostňovat. Hlavní důvod tvorby této struktury je především přehlednost pro samotného vývojáře. Existují ovšem adresáře, které je možné nalézt skoro ve všech aplikacích. [16]

V každém projektu je možné se setkat s adresářem obsahující obrázky a samotné stylování pro jednotlivé komponenty. Tento adresář nese název assets a je možné ho dále rozdělit pro přehlednost na adresáře pro obrázky a styly. Poté adresář, ve kterém se budou nacházet všechny jednotlivé komponenty. Tento adresář nese jednoduchý název components. Poslední je adresář pro samotné stránky, které budou složeny z komponent. Tento adresář ponese název pages. [16]

4 Komponenta

Pojem komponenta představuje určitou část kódu, která definuje konkrétní část uživatelského rozhraní. Díky komponentám je možné rozdělit webovou stránku na jednotlivé menší části. To pomáhá tomu, že jsou jednotlivé prvky stránky oddělené od ostatních a je možné je samostatně vyvíjet, upravovat či testovat. Velkou předností komponent je možnost jednoduché znuvupoužitelnosti v odlišných částech webové aplikace a následná rychlá odezva při úpravách. [17]

V celku jednoduchým způsobem je možné demonstrovat komponentu na příkladu z reálného světa. Běžným příkladem je jakákoliv dětská stavebnice. Pokud z této stavebnice chceme

postavit určitý objekt, bereme postupně jednotlivé dílky stavebnice a dáváme je dohromady. Aby byl koncept této stavby co možná nejvíce efektivní, je potřeba rozdělit jej na jednotlivé menší časti. Tyto části budou následně sestaveny zvlášť a nezávisle na ostatních. Ve chvíli, kdy budou všechny časti hotové, je možné je spojit dohromady a utvořit určitý celek. Pokud by po určité době kterákoliv část stavby nesplňovala nutné požadavky a bylo by nutné ji upravit nebo dokonce úplně odebrat, bude to velmi snadné a rychlé. [18]

Komponenta může mít na webové stránce mnoho podob, počínaje jednoduchým tlačítkem nebo formulářem. Může také být mnohem obsáhlejším a složitějším prvkem jako hlavička webové stránky nebo samotná sekce. [18]

Ná obrázku [Obrázek 15] je možné vidět grafický návrh úvodní sekce webové aplikace. V návrhu jsou zvýrazněné základní komponenty, které daná sekce obsahuje. Při pohledu na zmíněný obrázek je možné vidět již výhodu přednosti komponent a tou je znovupoužitelnost. Samotná navigační komponenta je vytvořena pouze jednou a následně je do dalších částí webové aplikace pouze přidávána. Pokud by v budoucnu mělo dojít na jakoukoliv změnu této komponenty, bude pouze stačit úprava hlavního souboru této komponenty a následné změny se promítnou do všech míst, kde je tato komponenta použita.

Obrázek 15 - Demonstrace komponent na webové stránce (zdroj vlastní)

V Reactu je možné vytvářet dva druhy komponent. Tyto komponenty se nazývají functional a class. Každá z nich má své určité využití a přednosti, které budou dále představeny.[18]

4.1.1 Functional komponenta

Tato varianta představuje nejjednodušší způsob pro definování komponent v Reactu. Zápis této komponenty připomíná JavaScriptovou funkci. Díky jednoduchému zápisu je lze velmi snadno číst. Oproti class komponentě nedokáže například pracovat se svým stavem nebo životním cyklem. K tomu, aby tyto schopnosti byli dostupné, je nutné komponentu doplnit o příslušné rozšíření v podobě React Hooks. Pro svoji jednoduchost a díky přidání React Hooks je tento typ komponenty nejpoužívanější při tvorbě aplikací. [18]

Obrázek [Obrázek 16] demonstruje dva způsoby zápisu functional komponenty.

Běžný zápis functional komponenty

```
function Contact(){
return(
<h1>Obsah</h1>
);
}
export default Contact
```

Zkrácený zápis pomocí

```
const Contact = () =>{
return(
<h1>Obsah</h1>
)
}
export default Contact
```

Obrázek 16 - Ukázka zápisu Functional komponenty (zdroj vlastní)

4.1.2 Class komponenta

Před přidáním React Hooks byla komponenta typu class jediným způsobem, jak uchovávat a následně pracovat se stavem komponenty. Oproti functional komponentám mají více podrobný svůj samotný zápis. V dnešní době se díky jednoduchosti zápisu komponent typu functional používá class komponenta pouze ve velmi komplexních případech. [18]Způsob zápisu class komponenty je možné vidět na obrázku [Obrázek 17].

Zápis Class komponenty

```
import React, { Component } from 'react'
export class Contact extends Component{
  render(){
    return(
     <h1>Obsah</h1>
    )
   }
}
export default Contact
```

Obrázek 17 - Ukázka zápisu Class komponenty (zdroj vlastní)

4.2 Hierarchie komponent

V Reactu je samotná hierarchie komponent reprezentována pomocí stromové struktury. Tato struktura je organizována do podoby rodič a potomek. Datová struktura typu strom se skládá z uzlů, kde platí že nejvyšší uzel je nazýván kořenový uzel. Tento uzel je vstupním bodem do celé struktury. Každý uzel může mít svého potomka a každý potomek musí mít pouze jednoho svého rodiče. Výjimku zde představuje kořenový uzel, ten ovšem rodiče mít nemůže. V Reactu jsou tyto uzly reprezentovány komponentami. Schopnost umět si představit samotnou strukturu, může být pro vývojáře velkou výhodou. Hlavně v případech, kdy je potřeba přemýšlet o předávání props nebo vyhledání potencionálních nedostatků a chyb. Je důležité si uvědomit, že ve chvíli, kdy dojde k přidání nové komponenty do aplikace, stává se tato komponenta automaticky novým uzlem ve stromové struktuře. [19]

Jak taková hierarchie vzniká uvnitř kódu je možné vidět na obrázku [Obrázek 18]. Při vytvoření rodičovské komponenty byl následně pomocí import zavolán potomek. Tím se zajistilo, že uvnitř rodičovské komponenty, je možné s dceřinou komponentou pracovat. Zároveň uvnitř stromové struktury došlo k přidání dvou nových uzlů, kde ParentKomponent je rodičem ChildComponent.

Rodičovská komponenta

```
import React from 'react'
import './ChildComponent'
function ParentComponent(){
return(
<ChildComponent/>
);
}
export default ParentComponent
```

Komponenta potomka

```
import React from 'react'
function ChildComponent(){
return(
<h1>Hello from child</h1>
);
}
export default ChildComponent
```

Obrázek 18 - Ukázka hierarchie komponent (zdroj vlastní)

4.3 Práce s komponentou

Při vytváření nové komponenty platí pravidlo pojmenování. Toto pravidlo udává, že každá komponenta začíná velkým písmenem. Pro vytvoření nové komponenty je nutné vytvoření nového souboru, který ponese stejný název, jako samotná komponenta. Následuje přípona, kterou bude daný soubor disponovat. Existuje mnoho přípon, které zde mohou být uplatněny. Protože je součástí této práce JSX, budou komponenty mít vždy koncovku .jsx. V situaci, kdy je soubor vytvořen, je potřeba definovat samotnou komponentu. Je nutné si tedy vybrat, zda se bude pracovat s functional nebo class komponentou.

Ve chvíli, kdy je samotná komponenta vytvořena a nejedná se o kořenovou komponentu je potřeba přidání této komponenty do příslušné rodičovské komponenty. K provedení takového kroku je nutné v rodičovské komponentě využít import pro určitou komponentu. Aby mohl být samotný import komponenty úspěšný, musí samotná komponenta mít definovaný export.

Tvorbu komponenty je možné demonstrovat vytvořením demonstrační aplikace. V adresáři sloužícím pro uchování komponent vytvořit nový soubor s názvem InfoBox.jsx. V tomto souboru je následně potřeba definovat samotnou komponentu. Poté pro zobrazení komponenty na hlavní stránce aplikace, její přidání do souboru App.js. Výsledný kód by se měl podobat obrázku [Obrázek 19].

InfoBox.jsx U

demo-app > src > components > InfoBox.jsx > ...

```
1   import React from 'react'
2
3   function InfoBox() {
4     return (
5       <div className='infoBox'>
6           <p>Text komponenty</p>
7       </div>
8     )
9   }
10
11  export default InfoBox
12
13
```

App.js M

demo-app > src > App.js > ...

```
1   import './App.css';
2   import InfoBox from './components/InfoBox';
3   function App() {
4     return (
5     <div className="App">
6      <InfoBox/>
7      <InfoBox/>
8      <InfoBox/>
9     </div>
10    );
11  }
12
13  export default App;
```

Obrázek 19 - Kód pro vytvoření komponenty (zdroj vlastní)

5 Komponenta a její data

5.1.1 Props

Props představují určitou hodnotu, kterou je možné předávat z rodičovské komponenty jejímu potomkovi. Velmi často jsou přirovnávány ke vstupním parametrům u metod v programovacích jazycích. V češtině jsou často nazývané jako vlastnosti. Při práci s vlastnostmi je možné pouze čtení, a ne následné úpravy v komponentě potomka. Díky vlastnostem lze velmi jednoduchým způsobem opětovně používat jeden typ komponenty. [20]

Nejjednodušším způsobem pro práci s vlastnostmi, je přidat v komponentě potomka parametr props. Následně v potomkovi postupně dochází ke specifikaci vlastností, které se budou používat. K tomu se využívá tečková anotace. V místě kde se bude s danou vlastností pracovat, je nutné použít props.navezVlastnosti. V případě kdy jsou vlastnosti využívány uvnitř class komponent, přistupuje se k nim pomocí anotace this.props. V situacích, kdy jsou vlastnosti používány pro výpis uvnitř elementů, je nutné takové vlastnosti obalit složenými závorkami. [20]

Práci s vlastnostmi je možné vyzkoušet v samotné demonstrační aplikaci. Pro umožnění pracovat s vlastnostmi je nutné přidat jako vstupní parametr komponenty klíčové slovo props. Následně změnit zobrazovaný text na určitou vlastnost. V demonstračním příkladu je využíváno spojení props.text. V poslední řade je nutné vytvořit samotný obsah vlastností. Tento krok se provádí pomocí definování samotného obsahu vlastnosti v deklaraci rodičovské komponenty. V souboru App.js je tedy nutné přidat tento obsah. Výsledný kód je možné vidět na obrázku [Obrázek 20].

InfoBox.jsx U

demo-app > src > components > InfoBox.jsx > ...

```jsx
import React from 'react'

function InfoBox(props) {
  return (
    <div className='infoBox'>
        <p>{props.text}</p>
    </div>
  )
}

export default InfoBox
```

App.js M

demo-app > src > App.js > ...

```jsx
import InfoBox from './components/InfoBox';
function App() {
  return (
  <div className="App">
   <InfoBox text={["seznam", <br></br>, "revírů"]}/>
   <InfoBox text={["členská", <br></br>, "správa"]}/>
   <InfoBox text={["rybářský", <br></br>, "řád"]}/>
  </div>
  );
}

export default App;
```

Obrázek 20 - Kód pro práci s props (zdroj vlastní)

5.1.2 State

Jedná se o určitý stav, který je možný vytvořit pro komponentu a následně se s ním může pracovat. Stejně jako u vlastností, je možné stav číst, ale je možná i jeho následná modifikace. Ve chvíli, kdy dojde ke změně stavu komponenty, dojde k znovunačtení komponenty. Při práci s class komponentou je možné vytvořit stav pomocí anotace this.state. Pokud je potřeba pracovat se stavem uvnitř functional komponenty, je potřeba využít usaState Hook. [20]

5.2 Props Drilling

Jedná se o proces, ve kterém dochází k předávání vlastností mezi komponentami. Přesněji řečeno, jedná se o předávání skrze více vrstev komponent, a to i v případě, že určitá vrstva

danou vlastnost nevyužívá. Z běžného pohledu se může tento způsob zdát velmi jednoduchý a efektivní. Jsou zde ovšem negativa, která například mohou nastat ve chvíli, kdy dojde k odebrání jedné z prostředních vrstev. Následně může také nastat problém ve chvíli, kdy se projekt stane obsáhlejším a samotnou vlastnost bude nutno předávat skrze mnoho vrstev. Způsobů, jak předejít tomuto procesu, je mnoho. Každý má svoje určité klady i zápory. Velmi efektivní způsob ovšem představuje useContext hook, který umožňuje přistupovat k určitým datům z kterékoliv části projektu. [21]

6 Stylování komponent

Způsob stylování uvnitř Reactu může mít mnoho podob. Nejčastěji se využívá Kaskádových stylů. Hlavním důvodem je převážně schopnost použití atributu className na vytvořené elementy. V samotném Reactu je možné pracovat s Kaskádovými styly třemi různými variantami. Každá z těchto variant má svůj určitý způsob zápisu a následně i své klady a zápory. Kromě stylování pomocí Kaskádových stylů je zde ještě možnost využít už předpřipravených komponent z volně dostupných knihoven. Komponenty z takových knihoven už převážně mají své stylování hotové, ale ve většině případů je možné doplnit stylování pomocí Kaskádových stylů. [22]

6.1 CSS Soubor

Jedná se o jednoduchý způsob stylování s využitím klasického soubor css. Aby mohla komponenta se stylem pracovat, je nutné u dané komponenty, ke které se styl vztahuje přidat pomocí import cestu k souboru se stylem. Při použití tohoto způsobu je nutné přidat elementům jejich názvy. Těmto názvům bude následně v css souboru přiřazeno jejich stylování. Výhodou této metody je možnost použití souboru se stylováním na více místech. Nevýhodou je skutečnost, že může dojít k problému, kdy více elementů má stejný název. [23]

Pro vyzkoušení práce s css souborem lze demonstrační aplikaci rozšířit o nový styl. Bude se jednat o styl sekce, která představuje určitou část aplikace. Je nutné vytvořit v adresáři pro uchování stylů nový soubor typu css. V tomto souboru dojde k definování stylu pro tag section. Následně je potřeba tento styl přidat do hlavního souboru aplikace tedy App.js a změnit tag div na section. Na obrázku [Obrázek 21] je možné vidět, jak by měl vypadat konečný kód.

App.js M

src > App.js > ...

```
1  import InfoBox from './components/InfoBox';
2  import './styles/SectionStyle.css'
3  function App() {
4    return (
5    <div className="App" >
6    <section style={heroSectionStyle}>
7     <InfoBox text={["seznam", <br></br>, "revírů"]}/>
8     <InfoBox text={["členská", <br></br>, "správa"]}/>
9     <InfoBox text={["rybářský", <br></br>, "řád"]}/>
10   </section>
11   </div>
12   );
13 }
14
15 export default App;
```

SectionStyle.css U

src > styles > SectionStyle.css > ...

```
1  section{
2      width: 100%;
3      min-height: 100vh;
4      background-color: darkslategray;
5      display: flex;
6      flex-direction: row;
7      justify-content: center;
8      align-items: center;
9      gap: 2rem;
10 }
11
12
13
14
15
```

Obrázek 21 - Kód pro stylování s CSS souborem (zdroj vlastní)

6.2 Inline

Tento způsob využívá přímého uplatnění daného stylu na konkrétní element. Není zde potřeba vytváření nové souboru, ve kterém by se definovaný styl nacházel. Výhodou tohoto způsobu je skutečnost, že definovaný styl se nachází přímo u komponenty, ke které je následně přiřazen. To pomáhá zlepšit přehlednost a také se podílí na úspoře celkové velikosti projektu. Oproti běžnému zápisu pravidel, kdy se používá oddělení více slov pomocí pomlčky, je zde využita metoda camelCase. [24]

Mezi nevýhody je možné zařadit absenci využití pseudo-tříd, tedy není možné definovat chování například pomocí: hover, která se používá na reakci elementu ve chvíli, kdy na ni uživatel přesune kurzor. Není také možné zde definovat media queries. Zápis lze provést dvěma způsoby. První představuje zápis stylu přímo do atributu style v daném elementu, druhý způsob představuje definování stylu pomocí JavaScriptu, kde styl je přidán do příslušné proměnné a následně přiřazen elementu. [24]

Je možné vyzkoušet si stylování Inline způsobem na demonstrační aplikaci, doplněním nové sekce Contact. Tato sekce bude mít stylování určená pomocí atributu přímo u elementu section. V poslední řadě je nutné pouze tuto komponentu přidat do hlavního souboru. Výsledek je možné vidět na obrázku [Obrázek 22].

App.js M

src > App.js > ...

```
import InfoBox from './components/InfoBox';
import './styles/SectionStyle.css'
import Contact from './pages/sections/Contact';
function App() {
  const heroSectionStyle = {
    backgroundColor: "brown"
  }
  return (
  <div className="App" >
  <section style={heroSectionStyle}>
   <InfoBox text={["seznam", <br></br>, "revírů"]}/>
   <InfoBox text={["členská", <br></br>, "správa"]}/>
   <InfoBox text={["rybářský", <br></br>, "řád"]}/>
  </section>
  <Contact/>
  </div>
  );
}

export default App;
```

Contact.jsx U

src > pages > sections > Contact.jsx > ...

```
import React from 'react'

function Contact() {
  return (
    <section id='contact' style={{backgroundColor: 'yellow'}}>
        <h1>Hello from contact</h1>
    </section>
  )
}

export default Contact
```

Obrázek 22 - Kód pro stylování pomocí Inline metody (zdroj vlastní)

6.3 CSS Module

Stylování s využitím modulů využívá běžný soubor typu css, ve kterém jsou styly definovány. Zvláštností tohoto způsobu je definování samotného stylu pro konkrétní elementy a následně přiřazení stylu ke každému elementu samostatně. Samotný identifikátor, který bude v css souboru definován, bude následně automaticky doplněn o další jedinečný název a díky tomu bude možné předejít problémům, kdyby náhodou byly použity dvě různé komponenty se stejným názvem. [24; 23]

Samotné stylování s pomocí metody module je možné si vyzkoušet v demonstrační aplikaci. Výsledek je možné vidět na obrázku [Obrázek 23].

Obrázek 23 - Kód pro stylování pomocí metody Module (zdroj vlastní)

6.4 Existující knihovny

Není vždy nutné vytvářet vlastní stylování komponent. Existuje početné množství volně dostupných předpřipravených knihoven. Ty lze do projektů velmi jednoduchým způsobem přidat a následně s nimi pracovat. Práce s nimi je jednoduchá a v mnoha případech lze prvky knihoven dodatečně stylovat podle vlastní potřeby.

6.4.1 MUI

Jedná se o open-source knihovnu pro React komponenty. Obsahuje kolekci předpřipravených komponent, které je možné okamžitě použít po jejich přidání do projektu. Komponenty lze dodatečně velmi jednoduchým způsobem stylovat podle vlastních potřeb. Ke stylování je možné použít vlastní soubor s kaskádovými styly nebo samotné komponenty lze stylovat pomocí vlastností s názvem sx. [25]

```
npm install @mui/material @emotion/react @emotion/styled
```

Obrázek 24 - Příkaz pro přidání knihovny MUI (zdroj vlastní)

Pro přidání MUI knihovny do vlastního projektu je nutné spustit v terminálu příkaz z obrázku [Obrázek 24]. Ve chvíli, kdy bude příkaz úspěšně proveden je možné využívat všechny dostupné nástroje knihovny. Následuje zvolení určité komponenty, která má být přidána do projektu. Každá komponenta, kterou MUI disponuje má svou vlastní stránku, na které je detailně popsána funkčnost a význam jednotlivých částí, ze kterých se komponenta skládá.

6.4.2 Knihovna Lucide

Představuje volně dostupnou knihovnu ikon, které je možné použít v React aplikaci. Knihovna je zaměřená na jednoduché poskytnutí materiálů pro vývojáře, kteří mohou tyto materiály využívat. Samotné ikony fungují v prostředí Reactu jako komponenty. To umožňuje, pomocí vlastností, upravovat je přímo v aplikaci. Mezi základní props patří size pro velikost, color pro barvu. [26]

Pro přidání knihovny je nutné spuštění příkazu z obrázku [Obrázek 25]. Následně si na webové stránce Lucide vybrat potřebné ikony. Po vybrání ikony se u každé nachází popis i s ukázkou, jak zvolenou ikonu přidat do vlastního projektu.

```
npm install lucide-react
```

Obrázek 25 - Obrázek 25 - Příkaz pro přidání knihovny Lucide (zdroj vlastní)

7 HOOKS

Představují možnost pracovat se stavem komponenty a mnoho dalšího, bez nutnosti použít class komponentu. Do příchodu Reactu verze 16.8 bylo možné pracovat se stavem nebo životním cyklem komponenty pouze s využitím komponenty typu class. Právě s jejich příchodem nastala možnost využívat i tyto funkce uvnitř functional komponent a umožnit tak vývojáři využívat výhody Reactu. Důležitým pravidlem je používání Hooks pouze s functional komponentou. Při práci s každým typek Hooks je nejdříve nutný jeho samotný import na začátku komponenty. Samotných Hooks je možné využít spousty a každá má své využití. V následující části budou přestaveny nejčastěji používané.[27]

U každého příkladu je také uveden příklad pro jednoduchou demonstraci. Pokud je potřeba, je možné si tyto příklady vyzkoušet uvnitř demonstrační aplikace.

7.1 useId

Používá se pro generování unikátních identifikátorů, které je následně možné použít pro samotné HTML elementy a zamezit tak možnosti vzniku konfliktu, kdy by více elementů mělo stejný identifikátor. Je důležité si pamatovat, že pokud dochází k vygenerování identifikátoru pomocí useId v jedné komponentě, bude tento identifikátor po celou dobu totožný. Jedním z možných řešení je generovat několik identifikátorů pro více elementů, ale mnohem více se používá metoda, kdy dojde k vygenerování pouze jednoho identifikátoru, který je následně doplněn o dodatečný popis. Pokud ovšem dojde k opětovnému použití dané komponenty, ve které je využíván useId, identifikátory se shodovat nebudou. [28]

```
Práce s useId
import React, { useId } from 'react'
function Contact (){
const id = useId()
return(
<h1>{id}</h1>
)
}
export default Contact
```

Obrázek 26 - Příklad práce s useId (zdroj vlastní)

7.2 useState

Umožňuje vytváření a následnou kontrolu nad stavem uvnitř functional komponenty. Při vytváření dochází k nastavení počáteční hodnoty na určitou výchozí hodnotu. Tato hodnota může být i prázdný řetězec. Výstupy jsou následně dva. Jedná se o samotnou hodnotu stavu, která je v určitou dobu nastavena a následně druhý výstup představuje funkci pro změnu stavu. Tato funkce je volána ve chvíli, kdy má nastat změna stavu. [27; 29]

```
Práce s useState
import React, { useState } from 'react'
function Counter (){
const [count, setCount] = useState(0);
return(
<button onClick={() ⇒ setCount()count + 1}>{count}</button>
)
}
export default Counter
```

Obrázek 27 - Příklad práce s useState (zdroj vlastní)

7.3 useEffect

Používá se pro provádění vedlejších efektů, jako například načítání dat, pro práci se samotnou DOM nebo pro vyčištění nepoužívaných komponent. Samotné použití useEffect nastává ve chvíli, kdy komponenta provede dokončení svého vykreslení. Na konci se uvádí závislosti, na kterých samotných efekt záleží. Pokud dojde ke změně závislosti nastane znovuspuštění efektu. [29]

```
Práce s useEffect
import React, { useEffect, useState } from 'react'
const [fact, setFact] = useState("");
useEffect(() ⇒{
fetch("https://catfact.ninja/fact")
.then(response ⇒ response.json())
.then(data ⇒ setFact(data));
});
console.log(fact)
```

Obrázek 28 - Příklad práce s useEffect (zdroj vlastní)

7.4 useMemo

Umožňuje zapamatovat si složitější výpočet a jeho hodnotu, který bude následně uchován pro znovupoužití a nebude nutné ho provádět při každém načtení. Toto ovšem platí do chvíle, dokud nedojde ke změně například vstupních parametrů. Pomáhá pro zlepšení optimalizace a výkonu komponenty při práci se složitějšími výpočty. [27; 29]

```
Práce s useMemo
import React, { useMemo } from 'react'

const result = useMemo(() ⇒ {
return computeComplexValue(a, b);
}, [a, b]);
```

Obrázek 29 - Příklad práce s useMemo (zdroj vlastní)

7.5 useRef

Používá se pro vytvoření objektů, které mohou měnit svůj stav. Na rozdíl od toho, kdy je stav například kontrolován pomocí useState, vytváří useRef referenci na daný objekt a ve chvíli, kdy dojde k opětovnému vykreslení komponenty, hodnota objektu bude stejná. [27]

```
Práce s useRef
import React, { useRef } from 'react'
const someRef = useRef();
const refValue = someRef.current;
console.log(refValue)

<p ref={someRef}>Obsah</p>
```

Obrázek 30 - Příklad práce s useRef (zdroj vlastní)

7.6 useContext

Umožnuje přístup k datům, které se nachází mimo danou komponentu. Velmi jednoduchým a efektivním způsobem lze s jeho využitím předejít zbytečnému předávání dat přes více vrstev komponent a také je to efektivní způsob, jak se vyvarovat Props Drilling. [29]

Postup práce s useContext je následující. Nejdříve je nutné definovat samotný context s hodnotou, která má být předávána. Následně je nutné obalit rodičovskou komponentu v části, kde má být context použit. Nejčastěji dochází k obalení komponenty App, aby byl přístup možný odkudkoliv. Posledním krokem je zavolání samotného contextu uvnitř komponenty, kde se s ním má pracovat. [29]

```
Práce s useContext
import React, { createContext } from 'react'
export const Mycontext = createContext("");
const data = "admin";
<Mycontext.Provider value= {data}>
<App/>
<MyContext.Provider>

import React, { useContext } from 'react'
import { MyContext } from '../App'
const dataFromContext = useContext(MyContext);
alert(dataFromContext)
```

Obrázek 31 - Příklad práce s useContext (zdroj vlastní)

8 Práce s formulářem a zpracování událostí

8.1 Zpracování událostí

Při vytváření moderních webových aplikací je v mnoha případech nutné umožnit uživateli možnost interagovat se samotnou aplikací. Díky tomu je možné od uživatele získat potřebná data, pomocí kterých může být následně samotná aplikace doplněna nebo jen včasně reagovat na jeho požadavky. Události je možné v Reactu vyvolat například kliknutím na tlačítko či samotnou komponentu nebo při odesílání formuláře. K tomu, aby mohla být určitá událost zpracována, je potřeba vytvořit samotnou funkci, která bude definována podle zadaných požadavků, které mají být vykonány. React se proti ostatním odlišuje tím, že při samotném vyvolání události dojde k vytváření speciálního objektu, který v sobě obsahuje informace o dané události. V tomto objektu je možné nalézt, o jaký typ události se jedná nebo který element událost vyvolal. Tento objekt je následně předán samotné funkci a může se s ním dále pracovat. [30]

Stejně jako při práci s událostmi v jazyce HTML je nutné k příslušnému elementu, který má vyvolat danou událost, přiřadit, o jakou událost se jedná. Na rozdíl od HTML se v Reactu nepředává název funkce jako řetězec, ale je předán samotný název funkce. Dále je nutné si

pamatovat, že se v Reactu udává pouze název funkce, a to bez závorek. Pokud by se tak stalo, tak by došlo k vyvolání samotné funkce. [31]

Zápis Eventu v HTML

```
<button onclick="handleClick()">Obsah</button>
```

Zápis Eventu v Reactu

```
<button onClick={handleClick}>Obsah</button>
```

Obrázek 32 - Ukázka správného a nesprávného zápisu události (zdroj vlastní)

Následující samotné definice funkce možné zapsat dvěma způsoby. V prvním je použito klíčové slovo function, následně doplněno názvem funkce a samotným tělem. Druhý způsob využívá zkrácený zápis, který šetří místo a je jednodušší na zápis. [30] Oba zápisy je možné vidět na obrázku [Obrázek 33].

Běžný zápis funkce

```
function doSomething(){
alert("Hello");
}
```

Zkrácený zápis funkce

```
const doSomething = () => {
alert("Hello");
}
```

Obrázek 33 - Ukázka zápisu funkce (zdroj vlastní)

8.2 Základní práce s formulářem

Formuláře představují nedílnou součást webové stránky. Bez formulářů by operace typu přihlašování nebo získání dat od uživatele byla složitá. Základní vytvoření samotného formuláře v Reactu je hodně podobná jako tvorba formuláře v běžném HMTL souboru. Pro definování formuláře se využívá tag form, který následně obsahuje další tagy typu input pro definování vstupů a následně i samotné tlačítko pro reakci formuláře. Rozdíl je možné nalézt v samotném zpracování dat. Jeden způsob přenechává zpracování dat z formuláře samotné DOM v prohlížeči a druhý naopak využívá stavu komponenty pro uchování dat. [32]

První způsob, který se používá nejčastěji se nazývá Controlled. Znamená, že samotné hodnoty, které byly získány od uživatele, jsou uchovány pomocí stavu komponenty. Pokud následně dochází ke změně této hodnoty například, že by jí uživatel přepsal, bude tato změna okamžitě promítnuta do samotného stavu. Pro vytvoření tohoto typu je nutné mít kromě definovaného stavu i funkci, která bude hlídat a měnit změnu vstupních hodnot od uživatele. [32]

Druhým způsobem je přenechání zpracování dat samotné DOM v prohlížeči. Tento způsob se nazývá Uncontrolled. Na rozdíl od předchozího způsobu není nutné vytvářet žádný stav, do kterého by se hodnoty postupně ukládaly. Vše je totiž ukládáno do samotného prohlížeče a získat přístup k těmto datům lze pomocí useRef. [32]

Pro následnou práci s formulářem je nutná jeho definice pomocí tagu form. Tomu je nutné přiřadit příslušnou událost a funkci, která nastane ve chvíli kdy má být formulář zpravován. Nejčastěji se jedná o událost onSubmit, kterou vyvolá tlačítko typu submit. V samotné funkci,

která rozhodne, co se s daty z formuláře stane, je nutné využít ještě jednu důležitou funkci. Jedná o prevenDefault a je volána pomocí události. Tato funkce zamezí běžnému chování formuláře, jako je například automatické znovunačtení stránky. Pokud by tento krok nebyl proveden, data uložená pomocí stavů by byla ztracena.

FormDemonstration.jsx U

src > pages > FormDemonstration.jsx > ...

```jsx
import React, {useState, useRef } from 'react'
function FormDemonstration() {

  const[formData, setFormData] = useState({name: "", age: 0});
  const selectRef = useRef(null);

  const handleChange = (event) =>{
    const {name, value} = event.target;
    setFormData((prevFormData) => ({...prevFormData, [name]: value}))
  }

  const handleSubmit = (e) =>{
    e.preventDefault();
    console.log("Name: ", formData.name)
    console.log("Age: ", formData.age)
    console.log("Gender: ", selectRef.current.value)
  }
  return (
    <form onSubmit={handleSubmit}>
      <label htmlFor="name"></label>
      <input type="text" name="name" value={formData.name} onChange={handleChange} />

      <label htmlFor="age"></label>
      <input type="number" name="age" value={formData.age} onChange={handleChange} />

      <select name="gender" ref={selectRef}>
        <option value="man">Muž</option>
        <option value="woman">Žena</option>
      </select>

    <button type="submit">Odeslat</button>
    </form>
  )
}

export default FormDemonstration
```

Obrázek 34 - Ukázka tvorby formuláře pomocí Controlled a Uncontrolled komponent (zdroj vlastní)

Na obrázku [Obrázek 34] je možné vidět vzorový příklad tvorby jednoduchého formuláře. V tomto příkladu dochází ke zpracování dat formou, kdy jsou data ukládána do stavu i useRef. Následně je zde vytvořena funkce, která reaguje na změnu ve vstupu formuláře. Pokud nastane jakákoliv změna, bude automaticky promítnuta do stavu. Poslední částí je funkce, která je volána ve chvíli odeslání formuláře. Pro zamezení obnovení stránky je zde využita funkce preventDefault.

9 Vizuální zobrazení dat

Při tvorbě každé moderní webové stránky se velmi často lze setkat s velkou množinou dat, kterou je následně nutné v přehledné a funkční formě zobrazit uživateli. Značkovací jazyk HTML obsahuje samotné tagy pro reprezentaci v takových případech. Jedná se o tagy ul a následně table. První zmíněný tag se používá pro definici jednoduchého seznamu a druhý pro tvorbu tabulky. V Reactu je možné využít oba tyto způsoby, nicméně to nelze provést běžným způsobem tak jako v HTML.

9.1 Zobrazení pomocí seznamu

Prvním způsobem je již zmíněné zobrazení pomocí jednoduchých listů. V Reactu se často tento způsob nazývá List and Keys. Představují v Reactu určitou kolekci elementů, které je následně možné zobrazit pomocí JSX. Elementy, které se budou zobrazovat, musí být stejného typu. Samotné elementy se velmi často ukládají do pole, ze kterého se následně s nimi pracuje. Pro samotnou práci se využívají metody map() a filter(). [33]

Postup pro práci se seznamem je následující. Nejdříve je nutné data uložená uvnitř pole pomocí metody map() vložit do nové proměnné, která je následně použita pro zobrazení seznamu. [33]

```
function MyComponent(){
const numbers = [0, 1, 2, 3, 4];
const listItems = numbers.map(number =><li>{number}</li>);
return(
<ul>{listItems}</ul>
);
}
export default MyComponent
```

Obrázek 35 - Příklad pro práci se seznamem (zdroj vlastní)

Funkce filter() poté slouží k jednoduchému fitrování dat podle určité vlastnosti. [33]

```
const numbersEven = numbers.filter(number => number % 2 === 0)
const listItems = numbersEven.map(number =><li>{number}</li>);
```

Obrázek 36 - Ukázka filtrování seznamu (zdroj vlastní)

Poslední nutnostní při práci se seznamem je zaručit, aby každá položka v seznamu měla svůj unikátní klíč. Ten může mít podobu jednoduchého řetezce nebo čísla, ale v samotném seznamu se nemůže vyskytovat více položek se stejným klíčem. Tento klíč je důležitý, protože umožňuje Reactu efektivně aktualizovat samotné prvky seznamu. [33]

```
const listItems = numbers.map(number =><li key={number}>{number}</li>);
```

Obrázek 37 - Ukázka uplatnění keys na prvky seznamu (zdroj vlastní)

Pro vyzkoušení práce se seznamem je možné implementovat příklady z obrázků [Obrázek 35] [Obrázek 36] [Obrázek 37], v libovolné části demonstrační aplikace.

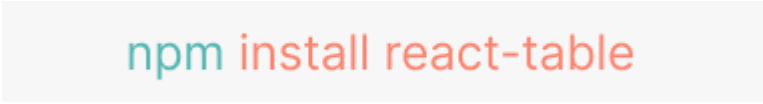
9.2 Knihovna React Table

Druhou možností pro vizualizaci je tabulka, ve které je možné zobrazovat data přehledněji a následně tabulku rozvíjet o užitečné funkce. První možností pro tvorbu je vytvářet tabulku tradičním způsobem. Využít k tomu příslušné tagy, potom vytvořit určité stylování a v poslední řadě doplnit samotnou tabulku o funkce, které výrazně zvyšují kvalitu práce s tabulkou. Toto vše ovšem bude časově náročné a zdlouhavé. Naštěstí existuje jednodušší cesta, která se nazývá React Table. Jedná se o jednu z nejpoužívanějších knihoven v Reactu, kterou vytvořil Tanner Linsley. Je neustále aktualizována o nové funkce, přitom je poměrně malá na uložiště. [34]

Samotná React Table obsahuje funkce jako třídění, stránkování, vyhledávání a mnoho dalšího. Je vhodné jí tedy používat v případech, kdy je nutné zobrazovat obsáhlejší data a následně s těmito daty dále pracovat. Pokud by se pracovalo s malou sadou dat, stačí jednoduchá tabulka vytvořená samotným vývojářem. [34]

Aby bylo možné s React Table pracovat, je nutné do projektu přidat novou knihovnu. K tomu je třeba využít příkaz z obrázku [Obrázek 38].

Při práci se využívá hook useTable, pomocí kterého dochází k vytvoření instance samotné tabulky. Tato instance vrací potřebné funkce a objekty, které se potom používají pro vykreslení samotné tabulky. První je getTableProps, který obsahuje vlastnosti, které je potřeba použít na samotný tag table. Dále getTableBodyProps, kde se nachází vlastnosti pro tag tbody. Další je headerGroups, to je pole reprezentující samotnou hlavičku tabulky. Obsahuje tedy samotné hodnoty atributu Header, který se používá při definování sloupců. Poslední dvě jsou pole rows, to reprezentuje samotné řádky tabulky a druhý je prepareRow, tedy funkce, která se stará o všechny potřebné styly a vlastnosti. [35]



Obrázek 38 - Příkaz pro instalaci React Table (zdroj vlastní)

Po přidání knihovny a tvorbu samotné tabulky je nutné dodržet postup, který zahrnuje získaná data pro tabulku, definování sloupců, použití dat a sloupců pro vytvoření instance tabulky, definování struktury tabulky pomocí HTML tagů a v poslední řadě použít vytvořenou instanci pro spojení se samotnou strukturou. Aby byl postup co nejvíce srozumitelný je rozdělen do pěti samostatných bodů. [35; 34]

První krok představuje přípravu dat, které bude tabulka zobrazovat. Data je možné uchovávat v mnoha podobách, ať už v jednoduchých souborech přiložených k aplikaci, nebo v databázích. V tomto jednoduchém případě bude využita první varianta, tedy soubor typu JSON.

Ve druhém kroku je potřeba definovat sloupce tabulky. Tyto sloupce slouží pro samotné definování tabulky a následně je bude možné vidět v samotné vizuální podobě tabulky. Sloupce se definují do pole jako objekty. Každý objekt se skládá z atributů, kterých může být více. Při vytváření tohoto souboru zde vždy musí být použity dva atributy a to Header a Accessor. První atribut Header představuje samotnou hodnotu, která se bude následně zobrazovat v tabulce jako nadpis pro sloupec. Druhý atribut Accessor představuje klíč pro přístup k definovaným datům. Tento klíč se následně používá pro správné přiřazení hodnot do správného sloupce. Samotná data je možné definovat ve vlastních souborech nebo přímo jako proměnnou uvnitř komponenty

ve které tabulka vzniká. [35; 34] Konečný soubor může mít podobu, kterou lze vidět na obrázku [Obrázek 39].

JS columns.js U ×

src > JS columns.js > ...

```
export const COLUMNS = [
    {
        Header: 'ID',
        accessor: 'id',
    },
    {
        Header: 'Číslo',
        accessor: 'number',
    },
    {
        Header: 'Název',
        accessor: 'name',
    },
    {
        Header: 'Typ',
        accessor: 'type',
    },
]
```

Obrázek 39 - Ukázka definování sloupců pro React Table (zdroj vlastní)

Třetí krok představuje použití Hooks. Prvním je useTable, tedy hook, kterým disponuje samotná knihovna. Díky němu lze vytvořit instanci tabulky a následně získat přístup k potřebných metodám. Z předešlých dvou kroků, ve kterých došlo k vytvoření souboru obsahujího data a následně souboru, kde jsou definovány sloupce, je nyní potřeba tyto soubory přidat do samotné komponenty reprezentující tabulku. K tomu se využije import a následně pomocí useMemo hook dojde k uložení těchto dat do příslušných proměnných. Samotných useMemo se v tomto okamžiku používá k zamezení znovu načítání dat ve chvíli, kdy by došlo k obnovení stránky. Následně se využívá hook useTable, který přijímá dva parametry. Sloupce, které bude tabulka obsahovat a následně samotná data. Po vytvoření instance je potřeba získat jednotlivé části, ze kterých bude tabulka vykreslena. [34]

SimpleTable.jsx U ×

src > components > SimpleTable.jsx > SimpleTable > headerGroups.map() callback > headerGroup.headers.map() callback

```
import React, {useMemo} from 'react'
import {COLUMNS} from '../columns'
import { useTable } from 'react-table'
import REVIRY_DATA from '../data.json'

function SimpleTable() {
    const data = useMemo(() => REVIRY_DATA, []);
    const columns = useMemo(() => COLUMNS, []);

    const tableInstance = useTable({columns, data});
    const { getTableProps, getTableBodyProps, headerGroups, rows, prepareRow } = tableInstance;
```

Obrázek 40 - Ukázka aplikace dat pro React Table (zdroj vlastní)

Ve čtrtém kroku nastává situace, kdy je potřeba definovat samotou tabulku pomocí HTML tagu table, thead, tbody, tr a th v kombinaci se získanými daty od instance tabulky. [34] Zápis je možné vidět na obrázku [Obrázek 41].

```
<table {...getTableProps()}>
    <thead>
        {headerGroups.map((headerGroup) => (
            <tr {...headerGroup.getHeaderGroupProps()}>
                {headerGroup.headers.map((column) => (
                    <th {...column.getHeaderProps()}>
                        {column.render('Header')}
                    </th>
                ))}
            </tr>
        ))}
    </thead>
    <tbody {...getTableBodyProps()}>
        {rows.map((row) => {
            prepareRow(row);
            return (
                <tr {...row.getRowProps()}>
                    {row.cells.map((cell) => {
                        return (
                            <td {...cell.getCellProps()}>
                                {cell.render('Cell')}
                            </td>
                        );
                    })}
                </tr>
            );
        })}
    </tbody>
</table>
```

Obrázek 41 - Definování tabulky (zdroj vlastní)

Posledním krokem je už pouhé doplnění vlastního stylu, například jako na obrázku [Obrázek 42] a velmi jednoduchá tabulka je hotová. Dále je zde možnost rozšíření schopností, které tabulka dokáže. Už v samotném úvodu bylo pár zmíněno a sama tabulka disponuje mnoha funkcemi. V následující části budou popsány nejvíce využívané a to jsou stránkování a filtrování.

TableStyle.css U

src > styles > # TableStyle.css > ...

```
1   table{
2       border-collapse: collapse;
3       width: 40%;
4       text-align: center;
5   }
6   td, th{
7       border: 1px solid black;
8   }
```

Obrázek 42 - Příklad stylování tabulky (zdroj vlastní)

První rozšíření je filtrování, které představuje velmi efektivní nástroj ve chvíli, kdy tabulka obsahuje velké množství dat a užitavel chce toto velké množství dat zúžit. V React Table je možné filtrovat dvěma způsoby. První představuje globální filtrování, které je aplikováno na celou tabulku. Pokud uživatel zadá určitou hodnotu, podle které si přeje tabulku fitrovat, bude

se tato hodnota hledat v každém sloupci tabulky. To může být v mnoha případech velmi neefektivní a časově náročné. Druhý způsob představuje filtrování zaměřené na konkrétní sloupec. To znamená, že pokud si uživatel bude přát tabulku filtrovat, bude mu řečeno podle jaké hodnoty filtrování probíhá. Tento způsob představuje proti globálnímu filtrování výhodu v úspoře času při filtrování a také samotné vyšší přehlednosti. [35]

Nejdříve je potřeba definovat pomocí useState stav, ve kterém se bude aktuální hodnota filtru uchovávat. Následně vytvořit funkci, která bude volána ve chvíli, kdy má dojít k filtrování. V této funkci bude pomocí vyvolané události získána hodnota elementu a následně bude podle zvolené hodnoty filtru přiřazena pro filtrování. Poté je potřeba rozšířit samotnou tabulku o nové rozšíření useFilters a převzít si od instance tabulky hook setFilter. Ve chvíli kdy tyto kroky budou provedeny, stačí pouze přidat příslušný element pro zadání hodnoty filtru. [36] Nutné příkazy k rozšíření je možné vidět na obrázku [Obrázek 43].

```
const [filterInput, setFilterInput] = useState("");

const handleFilter = e =>{
    const value = e.target.value || undefined
    setFilter("number", value)
    setFilterInput(value)}

    const tableInstance = useTable({columns, data}, useFilters);
    const { getTableProps, getTableBodyProps, headerGroups, rows, prepareRow, setFilter, } = tableInstance;

<input type="text" value={filterInput} onChange={handleFilter}/>
```

Obrázek 43 - Ukázka rozšíření filtrování React Table (zdroj vlastní)

Posledním rozšířením, které bude demostrováno je stránkování. Toto rozšíření umožní rozdělit tabulku na více částí, kde v každé časti bude vždy deset záznamů a pomocí příslušných tlačítek mezi těmito částmi procházet. Samotné zavedení této schopnosti je opravdu jednoduché. Stačí pouze provést import usePagination a rozšířit s ní instanci tabulky, tak jako u filtrování. Následně v místě kde dochází k získání dat z instance tabulky, smazat položku rows a nahradit jí položkou page. Poslední krok je i přepsání položky rows uvnitř JSX, kde bude také nahrazeno položkou page. Aby bylo možné mezi jednotlivými částmi přecházet, je potřeba zavést tlačítka s příslušnými funkcemi. Proto instanci tabulky je nutné opět rozšířit o funkce nextPage a previousPage. Poté už pouze stačí definovat dvě tlačítka a tyto funkce každému tlačítku přiřadit. [37] Výsledné úpravy je možné vidět na obrázku [Obrázek 44].

```
const tableInstance = useTable({columns, data}, useFilters, usePagination);
const { getTableProps, getTableBodyProps, headerGroups, page, prepareRow, nextPage, previousPage, setFilter, } = tableInstance;

<div>
    <button onClick={() =>previousPage()}>Predchozi</button>
    <button onClick={() =>nextPage()}>Dalsi</button>
</div>
{page.map((row) => {
```

Obrázek 44 - Ukázka stránkování React Table (zdroj vlastní)

10 JSON Server

Při tvorbě aplikací pomocí Reactu je velká pravděpodobnost, že bude nutné pracovat s velkým množstvím dat. Tyto data je nutné v určitém místě uchovávat. Jako nejčastější způsob je možné využít jednoduchý soubor, ve kterém budou data zapsána a při každém spuštění aplikace budou data načtena. Nejvíce pro svou jednoduchost je možné pracovat se souborem typu JSON.

Nicméně při práci s běžnými soubory je velmi často nutné vytvořit implementaci, ve které je celý soubor potřeba načíst a při každé změně ukládat. [38]

Samotný JSON Server představuje jednoduchý nástroj, který vytváří prostředek pro uchování dat. To umožní velmi jednoduchým a zároveň rychlým způsobem uložená data načítat, mazat, ale i upravovat. [38]

```
npx json-server --watch src/data/db.json --port 8000
```

Obrázek 45 - Příkaz pro spuštění JSON server (zdroj vlastní)

Samotné přidání JSON Serveru do projektu je velmi jednoduché a skládá se ze dvou kroků. V prvním je nutné pomocí terminálu a příkazu provést instalaci samotného serveru. Před samotným spuštěním příkazu je potřeba vytvořit JSON soubor, který bude představovat databázi. Název tohoto souboru je možné zvolit jakýkoliv, v tomto případě ponese soubor název db.json. V tomto souboru je následně dobré definovat si samotná data, která se zde budou uchovávat. Každá položka v datech by měla mít svůj identifikátor. Pokud by tento identifikátor nebyl určen, bude vytvořen samotným serverem. [38] Samotná struktura těchto dat může vypadat například jako na obrázku [Obrázek 46].

db.json U ×

src > data > {} db.json > ...

```
{
  "members": [
    {
      "id": 1,
      "name": "Karel",
      "age": 18
    },
    {
      "id": 2,
      "name": "Honza",
      "age": 26
    }
  ],
  "visits": []
}
```

Obrázek 46 - Ukázka definice dat uvnitř JSON Serveru (zdroj vlastní)

Po vytvoření souboru je možné spustit příkaz z obrázku [Obrázek 45]. Tento příkaz se skládá z více částí. V první časti se žádá o samotnou instalaci JSON Severu, následně je zde příkaz doplněn o samotnou cestu k souboru, kde se data nachází. Poslední částí je specifikace portu, na kterém má být server spuštěn. Tento port je nutné mít nastaven na 8000, protože výchozí port je 3000 a ten už využívá sám React. Je důležité si ovšem pamatovat, že tento příkaz je nutné spouštět pokaždé při spuštění samotného projektu. [38]

Samotná práce se serverem je založená na HTTP žádostech. Pro získání dat ze serveru je využívána žádost GET, pro přidání nové položky POST, pro aktualizaci položky PUT a v poslední řadě pro odebrání DELETE. Pro samotnou komunikaci se serverem je v Reactu možné využít useEffect doplněn o fetch. Dále komunikace probíhá na základě samotné adresy,

na které se data nachází. Při spuštění serveru je uživatel informován o jednotlivých adresách, kde je možné data nalézt. Pokud je potřeba pracovat pouze s jedním objektem z databáze, stačí pouze samotnou adresu na konci doplnit lomítkem a id objektu. [38]

Pro získání dat není nutné specifikovat metodu. Pokud dojde k provedení příkazu z obrázku [Obrázek 47], dojde k načtení všech dat z adresy.

```
useEffect(()=>{
  fetch("http://localhost:8000/members")
  .then(response => response.json())
  .then(members => console.log(members))
})
```

Obrázek 47 - Ukázka získání dat z JSON Serveru (zdroj vlastní)

Pokud ovšem má dojít k přidání, odebrání nebo aktualizaci objektu, není možné použít předchozí metodu. Server musí být totiž informován, o jaký typ metody se jedná. Správný postup pro aplikaci těchto úkonů je možné nalézt na obrázku [Obrázek 48]. Stačí pouze změnit metodu podle úkonu, který má proběhnout. V případě přidání a aktualizaci objektu je nutné přidat do těla samotný nový objekt. A v případě odebrání je potřeba doplnit adresu od id objektu. [38]

```
const[newMember, setNewMember] = useState({"id": 3, "name": "Jana", "age": 50})
    useEffect(()=>{
      fetch('http://localhost:8000/members', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
        body: JSON.stringify(newMember),
      })
      .then(response => response.json())
      .then(user => console.log(user));
    })
```

Obrázek 48 - Ukázka přidání objektu do JSON Serveru (zdroj vlastní)

11 Router

Nedílnou součástí každé webové stránky je schopnost reagovat na aktivitu uživatele a na základě této aktivity poskytnout uživateli odpovídající výsledek. Na každé webové stránce nalezneme navigační menu. Toto menu obsahuje odkazy na příslušné stránky nebo sekce. Pokud si uživatel chce zobrazit danou stránku či sekci, stačí mu pouze si příslušný odkaz zvolit a následuje přesměrování do zvoleného místa.

Směrování na běžných webových stránkách funguje následujícím způsobem. Uživatel se nachází na úvodní stránce celého webu. Pokud se chce podívat na jinou stránku, kliknutím na příslušný odkaz mu bude daná stránka zobrazena. Pro uživatele se tento proces jeví velmi jednoduše, nicméně pozadí tohoto úkonu vypadá následovně. Ve chvíli, kdy uživatel klikl na daný odkaz došlo k zaslání požadavku na server, ten zpět pošle novou HTML stránku, kterou si uživatel přál a tato stránka se následně vykreslí. Nicméně tímto způsobem React nefunguje, v samotném Reactu odpadá nutnost pro poslání požadavku na server. Sám React představuje Sigle Page Application, zkráceně SPA. React je tvořen z komponent a ve chvíli kdy dojde

k vytvoření nového projektu, se ve veřejném adresáři nachází soubor s názvem index.html. Všechno napsané uvnitř komponenty App, která představuje kořenovou komponentu, bude vykresleno ve zmíněném index souboru. Z toho je potřeba si uvědomit, že React využívá po celou dobu pouze jednoho HTML souboru pro vykreslování. To pomáhá především k rychlejšímu vykreslení samotné stránky a komponent na ní. [39]

Pro zpřístupnění směrování uvnitř React existuje knihovna, která se nazývá React Router. Pomocí této knihovny lze velmi efektivním způsobem umožnit React aplikaci směrování. Pro práci je potřeba znát následující základní části, ze kterých se samotný router skládá. První je Routes, která představuje rodičovskou komponentu, ve které dochází k definici samotných cest. [40]

Dále komponenta Route, která se používá pro definování samotné cesty k cílové komponentě. Je doplněna dvěma atributy. První atribut je path, pomocí kterého se specifikuje, která cesta bude sloužit k přesměrování na danou komponentu. Cesta, která bude v atributu path, záleží zcela na uživateli. V případě, že bude docházet k přesměrování na stránku s kontaktem, může být v path zaspáno /contact. Samotný Router už má k dispozici dvě předdefinované cesty. První cesta se zapisuje pouze pomocí zpětného lomítka a používá se k přesměrování na domovskou stránku aplikace. Druhá cesta je doplněna o pouhou hvězdičku a nastane ve chvíli, kdy požadavek od uživatele na zobrazení stránky nebude definován. [39; 40]

Následně komponenta Link, která umožňuje navigovat na jinou stránku. Tento element zpřístupní tag a společně s adresou, na kterou se má přejít pomocí přesměrování. To znamená, že pokud by se na prvek, který bude obalen pomocí Link, stisklo pravé tlačítko, nenastaly by žádné problémy. [40]

Kromě samotného Link je možné využít ještě komponentu NavLink, která má stejné chování, ale je možné jí velmi jednoduchým způsobem stylovat. Nejčastěji se používá při vytváření navigačního menu aplikace a pomocí atributu activeClassName jí lze nastavit například barvu pro případ jejího použití. [40]

HashLink má stejnou funkci jako komponenta Link s rozdílem, že se nepoužívá pro směrování na novou stránku, ale na konkrétní sekci na dané stránce. Pokud bude stránka složena z více sekcí, tak každá jednotlivá sekce musí mít svůj vlastní unikátní hash, který lze zapsat pomocí HTML atributu id. Následně při použití HashLink bude v atributu path samotná cesta doplněna o tento speciální hash, aby router věděl, kde má směrování skončit. [41]

Pro přidání routeru do projektu je nutné spustit v terminálu příkaz z obrázku [Obrázek 49].

```
npm install react-router-dom
```

Obrázek 49 - Příkaz pro instalaci React Router (zdroj vlastní)

11.1 Aktivace

Po úspěšé instalaci knihovny je nutné obalit hlavní kořenovou komponentu. Tuto komponentu je možné nalézt v souboru App.js. Komponenta App bude následně obalena komponentou BrowserRouter, která je součástí při stažení routeru. Ve chvíli kdy je kořenová komponenta obalena, je React router dostupný v celé aplikaci. Ukázku je možné vidět na obrázku [Obrázek 50]

```
import { BrowserRouter } from react-router-dom
root.render(
  <React.StrictMode>
    <BrowserRouter>
    <App />
    </BrowserRouter>
  </React.StrictMode>
);
```

Obrázek 50 - Ukázka aktivace React Router (zdroj vlastní)

Následuje nutnost definování samotných cest. Tento bod je velmi důležitý. Pokud by nebyl proveden, router by nedokázal přesměrovat uživatele na příslušná místa. Tato definice probíhá uvnitř komponenty App z důvodu toho, že se jedná o kořenovou komponentu. Zde je potřeba využít komponenty samotného Routeru tedy Routes a Route. Samotná komponenta Route je následně doplněna o dva atributy path a element. [39]

Pro vyzkoušení práce se samotným Routerem je možné rozšířit demonstrační aplikaci. První krok představuje definování samotných stránek v příslušném adresáři. V demosntračním příkladu se jedná o stránky Home, Page404, Dashboard a Rules. Následně je nutné přidat samotný Router do projektu pomocí příkazu. Po úspěšném přidání Routeru je potřeba provést tři kroky. V prvním kroku dojde k obalení celé komponenty App pomocí komponenty BrowserRouter. Dále je potřeba v komponentě App definovat samotné cesty pomocí komponenty Routes a Route. V posledním kroku už pouze stačí aplikovat komponentu Link na objekt, který bude sloužit pro směrování. Možný výsledek tohoto pokusu je možné vidět na obrázku [Obrázek 51].

index.js M

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
import { BrowserRouter } from 'react-router-dom';
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <BrowserRouter>
    <App />
    </BrowserRouter>
  </React.StrictMode>
);

reportWebVitals();
```

App.js M

```
import './styles/SectionStyle.css'
import Home from './pages/Home'
import Page404 from './pages/Page404'
import Dashboard from './pages/Dashboard'
import Records from './pages/Records'
import {Routes, Route} from 'react-router-dom'
function App() {
  return (
  <div className="App" >
    <Routes>
      <Route path='/' element={<Home/>}/>
      <Route path='*' element={<Page404/>}/>
      <Route path='sprava' element={<Dashboard/>}/>
      <Route path='reviry' element={<Records/>}/>
    </Routes>
  </div>
  );
}

export default App;
```

InfoBox.jsx U

```
import React from 'react'
import InfoBoxStyle from '../styles/modules/InfoBox.module.css'
import { Link } from 'react-router-dom'

function InfoBox(props) {
  return (
    <Link to={props.url}>
      <div className={InfoBoxStyle.infoBox}>
        <p>{props.text}</p>
      </div>
    </Link>
  )
}

export default InfoBox
```

Obrázek 51 - Výsledek demonstračního příkladu s React Router (zdroj vlastní)

Mnoho webových stránek má pro jednotlivé obsahy vždy vytvořenou stránku, na které je možné se přesměrovat pomocí Link. Nicméně někdy samotný obsah nemusí být natolik obsáhlý a tvorba samostatné nové stránky by byla neefektivní. Jednodušší způsob představuje sloučení těchto méně obsáhlejších částí do jedné stránky. Na tuto stránku bude možné stále přistupovat pomocí odkazů, ale místo přesměrování na samotnou stránku bude uživatel přesměrován na konkrétní část stránky. V těchto případech není možné pro přesměrování využívat komponentu Link, protože by sice došlo k přesměrování, ale pouze na začátek dané stránky. Proto je nutné samotný router rozšířit o funkci hask-link pomocí příkazu na obrázku [Obrázek 52]. Samotný HashLink při přesměrování komunikuje s Routerem a ve chvíli, kdy dojde k přesměrování na příslušnou stránku, je Router informován, že obsah, který si uživatel přeje zobrazit, se nachází

v určité části stránky a Router se na danou část musí přesunout. Toto rozšíření umožní přesměrování na stránky a následně na místo sekce odznačené pomocí atributu id. [41]

```
npm install --save react-router-hash-link
```

Obrázek 52 - Příkaz pro přidání rozšíření HashLink (zdroj vlastní)

Pro samotnou práci s komponentou HashLink je stále potřeba mít vytvořenou příslušnou Route na danou stránku. V místě kde má docházet k přesměrování, bude namísto komponenty Link využita koponenta HashLink. Tato komponenta také disponuje vlastnotí pro definování cesty nicméně zde bude samotná cesta doplněna o hash samotné sekce. V uvedemém příkladu na obrázku [Obrázek 53] je možné vidět použití HashLinku, který umožní přesměrování na hlavní stránku aplikace k příslušné sekci Contact. Pokud by se sekce nenacházela na úvodní stránce musela by obsahovat i předdefinovanou Route.

Použití HansLink pro směrování

```
<HashLink to="/#contact">Contact</HashLink>
```

Definice hash pro danou sekci

```
<div id="contact">Contact section</div>
```

Obrázek 53 - Ukázka práce s HashLink (zdroj vlastní)

12 Podmíněné zobrazení

V Reactu podmíněné zobrazení představuje způsob, jak na základě určitých podmínek zobrazit odlišný obsah. Představuje klíčovou roli při tvorbě dynamických webových aplikací. S jeho pomocí lze výrazně zvýšit výkon a efektivitu samotné aplikace, protože zabraňuje vykreslování takovým prvkům, které nejsou v daném kontextu potřebné. Velmi často se používá se spojením zobrazení určité sady komponent nebo samotných HTML elementů. React disponuje řadou způsobů, kterými lze podmíněné zobrazení vykonávat. Mezi základní způsoby se řadí využití if výrazu, použití ternárního operátoru, použití logického AND operátoru a v poslední řadě využití konstrukce switch. [42]

První způsob využívá výrazu if. S jeho pomocí lze například určit, co bude prostřednictvím komponenty vykresleno. Při splnění podmínky dojde k provedení příslušné operace. Tuto metodu nelze kombinovat s JSX. Lze ji pouze zapisovat v částech představujících JavaScript. [42]

```
function MyComponent(){
const isMember = true;
if(isMember){
return(<h1>You are member!</h1>)
}else{
return(<h1>You are not a member!</h1>)
}
}
export default MyComponent
```

Obrázek 54 - Ukázka podmíněného zobrazení pomocí výrazu if (zdroj vlastní)

Druhou možností je využití ternárních operátorů. Pomocí těchto operátorů je možné zapsat podmínku do efektivního jednoho řádku uvnitř JSX. Tento zápis se skládá ze tří operandů. První operand představuje samotnou podmínku, podle které se má příkaz řídit. Zbylé dva operandy představují výrazy, které mají být vykresleny v určitých případech. Pokud je podmínka výrazu splněna, následuje vykreslení prvního výrazu a v opačném případě dojde k vykreslení výrazu druhého. [42]

```
function MyComponent(){
const isMember = true;
return(
<h1>{isMember ? "Welcome member!" : "You are not a member"}</h1>
);          Podmínka    Podmínka splněna    Podmínka nesplněna
}
export default MyComponent
```

Obrázek 55 - Ukázka podmíněného zobrazení pomocí ternárních operátorů (zdroj vlastní)

Třetím způsobem je využití logických operátorů AND a OR. Pokud je použit operátor AND, je nutné, aby byly splněny obě podmínky. Pokud je využit operátor OR, stačí, aby podmínka byla splněna pouze jedna. Příkladem využití tohoto způsobu může být například situace, kdy je potřeba, aby uživatel při vyplňování formuláře vyplnil jak své jméno tak heslo. [42]

Poslední častou možností je podmíněné zobrazení pomocí nástroje switch. Tento způsob je velmi efektivní v situacích, kdy existuje více podmínek, které mohou vykreslování ovlivnit. Stejně jako u metody s výrazem if nelze switch používat uvnitř JSX. Na rozdíl od if výrazu není potřeba definovat samotné podmínky, které musí být splněny. Jako vstup přijme switch proměnnou, která určuje, jaká z definovaných případů je ta správná a následně dojde k jejímu provedení. Pokud ani jeden případ neodpovídá vstupní proměnné, bude proveden výchozí případ, který je pro tuto situaci definován. [42]

13 O webové stránce

Samotná webová stránka představuje nástroj pro správu rybářské organizace. Umožňuje členům i nečlenům organizace zobrazovat seznam lovných revírů a následně i jejich samotný detailní popis. Dále je zde možné nahlédnout do rybářského řádu, kde se nachází nejdůležitější informace, které by měl každý rybář znát. Pro členy organizace aplikace umožňuje jednoduchou evidenci, ve které je možné vytváření návštěv samotných revírů. Tato webová stránka vznikla

jako demonstrační příklad, ve kterém došlo k využití jednotlivých kapitol bakalářské práce. Tento příklad nese název elektronický rybářský systém. Pro jeho vytvoření byl použit obrázek dostupný z [43]a následně texty ze [44].

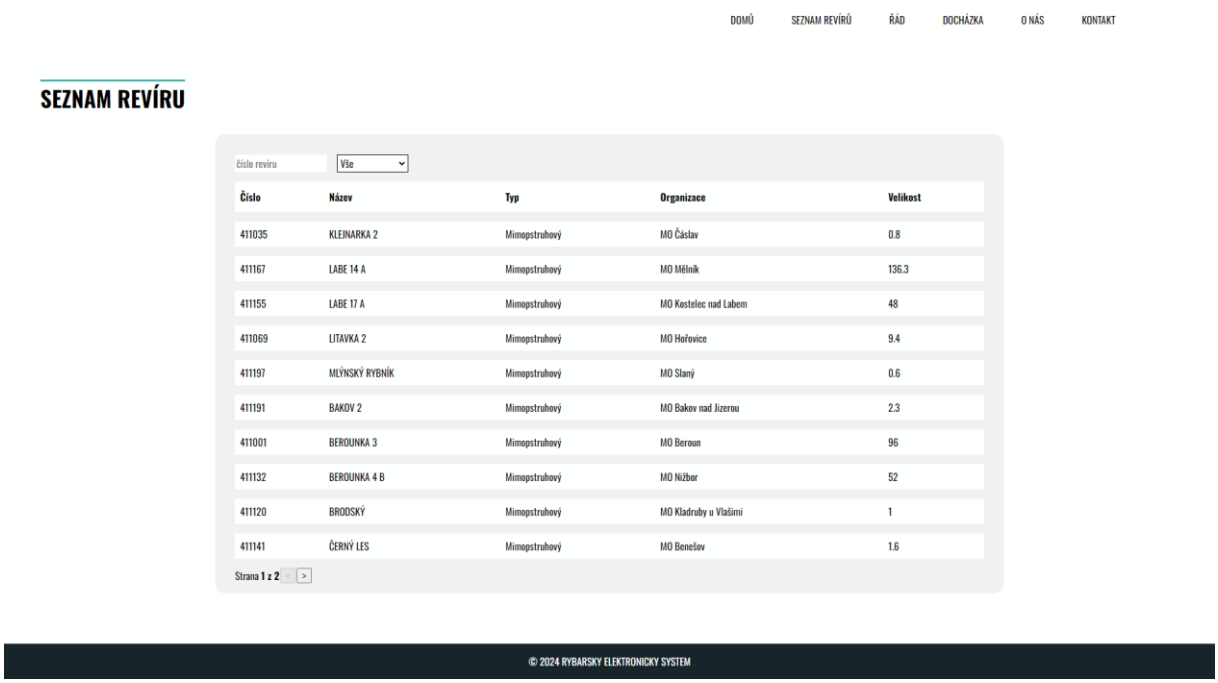
Následující část je zaměřena na popis jednotlivých stránek samotného rybářského systému. Tyto stránky je možné rozdělit do dvou skupin. V první skupině je možné nalézt takové stránky, ke kterým nemusí uživatel disponovat žádným ověřením a může k nim přistupovat kdokoliv. Druhou skupinu představují stránky, ke kterým je možné přistupovat pouze po přihlášení do samotného systému. Pokud by uživatel chtěl tyto stránky zobrazit musí být členem organizace a musí mít nutné údaje pro přihlášení.

Obrázek 56 - Úvodní sekce rybářského systému (zdroj vlastní)

Samotná hlavní stránka evidenčního systému poskytuje základní informace o dané organizaci, která samotné webové stránky využívá. Hlavní stránka je rozdělena do tří sekcí. Hlavním úkolem první sekce je především poskytnutí přehledného směrování na příslušné ostatní stránky s dalším obsahem. Je možné zde nalézt navigační menu a stylované boxy pro zmíněné směrování. Samotný vzhled směrování má dvě podoby. Pokud je uživatel na hlavní stránce, je mu zobrazeno směrování z obrázku [Obrázek 56]. Pokud se ovšem nenachází na hlavní stránce, jsou samotné směrovací cesty z boxů přidány do navigačního menu. To zaručuje, že uživatel má vždy k dispozici všechny potřebné cesty na příslušné stránky.

Obrázek 57 - Informační sekce rybářského systému (zdroj vlastní)

Druhá a třetí sekce hlavní stránky je určena pro vyplnění obsahu od samotné organizace, která bude webové stránky využívat. V obou sekcích dochází k využití ikon z volně dostupné knihovny Lucide. [26] Druhá sekce představuje prostor, který může být využit pro seznamovací část s organizací. V této části se zároveň nachází dvě komponenty stylované do boxů pro oddělení informací. Třetí sekce definuje prostor, který je určený pro předání kontaktu na danou organizaci. Opět zde dochází k využití boxů pro uchování informací, nicméně je potřeba zmínit, že se jedná o stejné boxy jako v předchozí sekci. Díky využití props je možné využívat tyto boxy v odlišných místech a podobách.

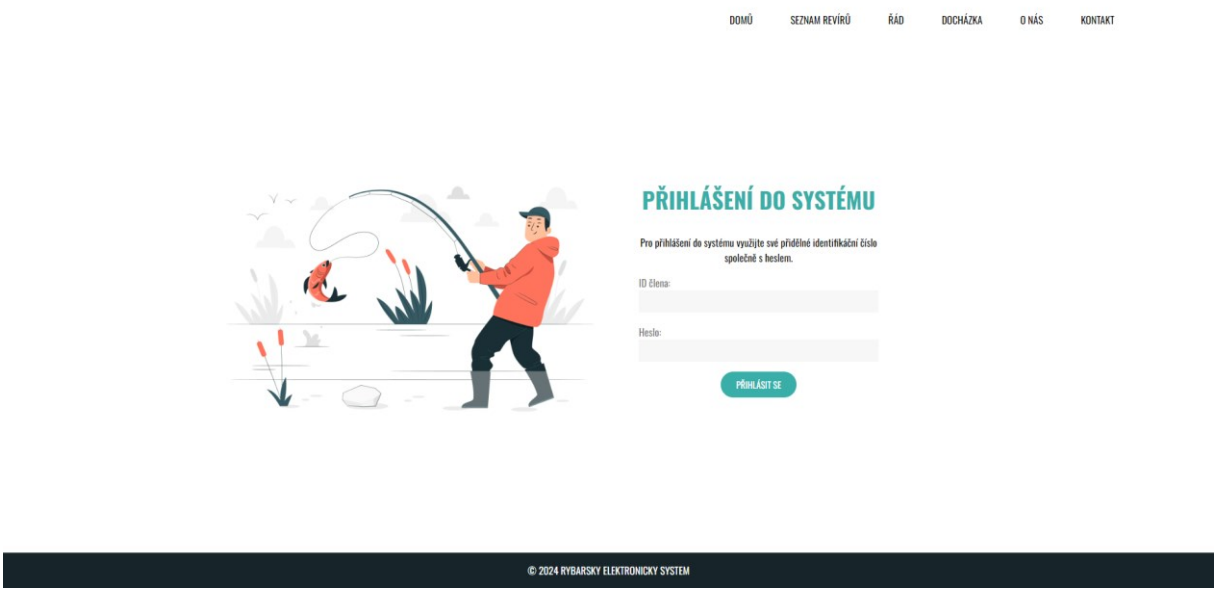


Obrázek 58 - Ukázka seznamu revírů z rybářského systému (zdroj vlastní)

Druhá stránka obsahuje tabulku se seznamem rybářských revírů. Samotný seznam umožňuje uživateli filtrování revírů podle jejich typu a následně vyhledání konkrétního revíru podle samotného čísla revíru. V seznamu se zároveň nachází pouze základní informace o revíru. Pokud je to nutné, disponuje každá položka v seznamu odkazem na samostatnou stránku, kde je možné nalézt kompletní informace o daném revíru.

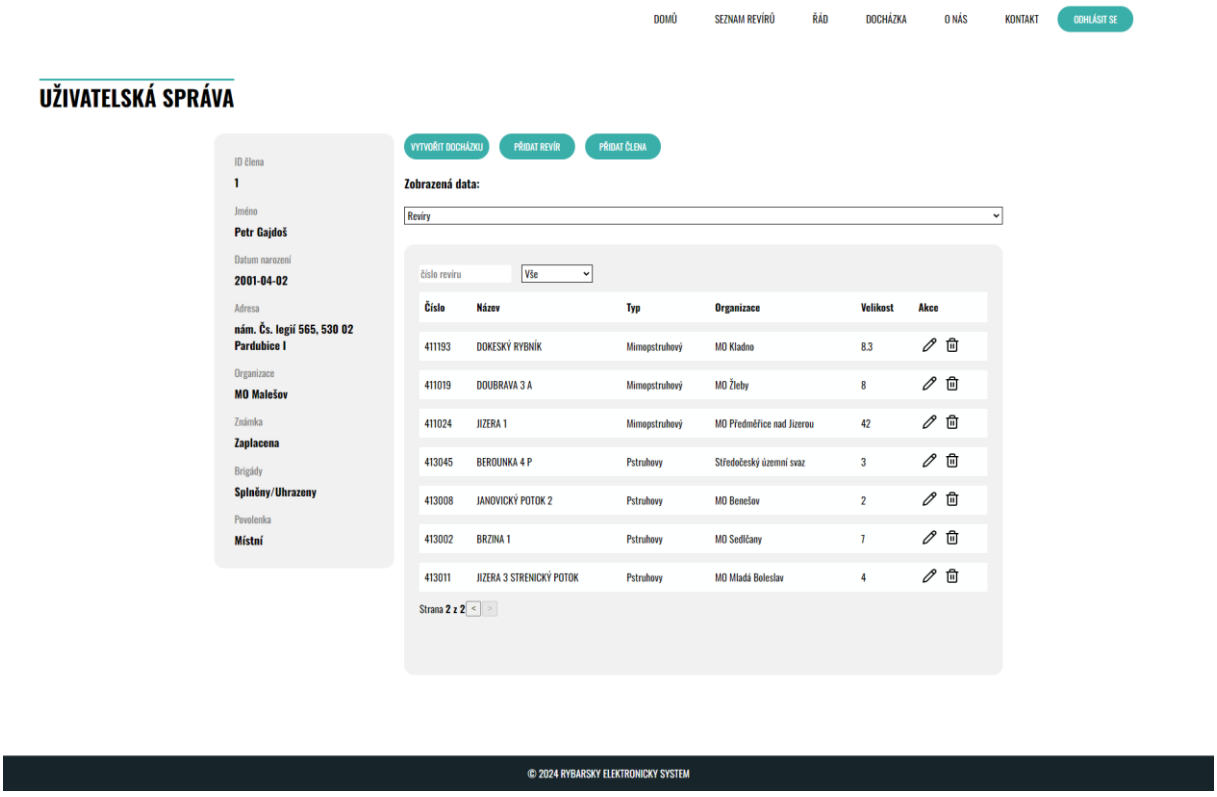
Obrázek 59 - Ukázka rybářské řádu vytvořeného pomocí MUI (zdroj vlastní)

Na další volně dostupné stránce je možné nalézt rybářský řád. Ten byl vypracován na základě použití knihovny MUI. S pomocí této knihovny bylo možné vytvořit přehledný způsob zobrazení důležitých dat.



Obrázek 60 - Přihlašovací stránka rybářského systému (zdroj vlastní)

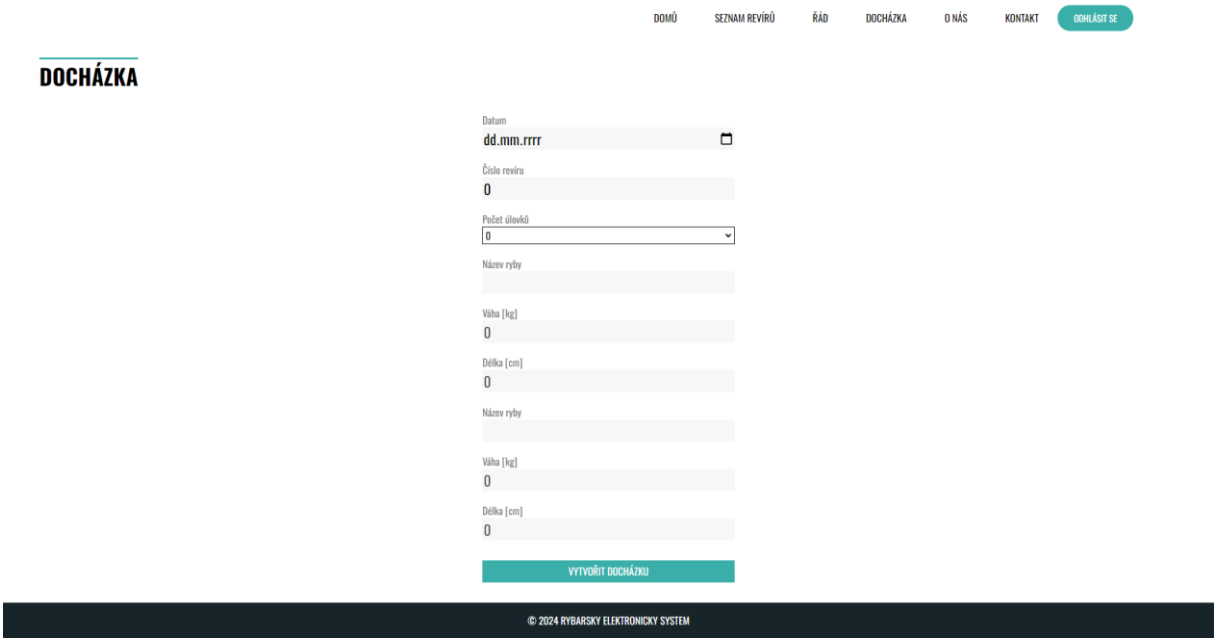
Poslední volně přístupná stránka obsahuje přihlašovací formulář do rybářského systému organizace. Pokud uživatel není členem a nemá přidělené své přihlašovací údaje není možné, aby se dostal na další stránky systému. Pokud ovšem uživatel členem je a má své přihlašovací údaje, je mu umožněno přihlášení do samotného systému.



Obrázek 61 - Uživatelská stránka rybářského systému (zdroj vlastní)

Jakmile dojde k úspěšnému přihlášení, je uživatel přesměrován na hlavní stránku uživatelské správy. Zároveň dochází k zapamatování samotného uživatele. Pomocí useContext dojde k uložení přihlášeného uživatele a ten díky tomu může následně bez nutnosti opětovného přihlášení procházet stránky jak z veřejné skupiny, tak i ze zabezpečené. Ve chvíli, kdy už si uživatel nebude chtít nahlížet do samotné správy, je mu umožněno tlačítkem odhlášení se ze systému odhlásit.

Následná uživatelská stránka je rozdělena do dvou variant. První varianta je určená pro vedení samotné organizace. Tato varianta umožňuje správu revírů i samotných členů. Je možné přidávat, odebírat a upravovat členy a revíry. Pokud ovšem přihlášený uživatel není součástí vedení organizace, je mu zobrazena varianta stránky, ve které mu není umožněna jakákoliv úprava revírů nebo členů. Je mu umožněna pouze tvorba samotné docházky nebo nahlížení do informací o samotných revírech. Když by uživatel byl členem organizace, ale nesplňoval by kterékoliv z pravidel pro získání povolenky k lovu, není mu umožněno vytvářet samotnou docházku.



Obrázek 62 - Ukázka vytvoření nové docházky (zdroj vlastní)

Poslední součástí systému je možnost tvorby docházky k určitému revíru. K tomu je možné využít formulář s příslušnými vstupy. Po vytvoření dojde k přidání nové docházky do databáze a je následně možné tuto docházku upravit nebo smazat. Běžný uživatel má možnost tvorby pouze docházky, nicméně uživatel, který je součástí vedení organizace, může navíc vytvářet nový revír a člena. Následně je mu umožněno tyto další předměty mazat nebo upravovat.

ZÁVĚR

Cílem této bakalářské práce bylo seznámit čtenáře se základními prvky knihovny React.js. U každého prvku je vysvětlena funkčnost a také příklad využití prvku. Složitější prvky jsou demonstrovány samotným kódem, který umožňuje čtenáři vyzkoušet si jeho tvorbu.

Výsledkem této práce je webová stránka pro rybářskou organizaci. Na této stránce najdeme prostor určený pro základní informace o organizaci. Následně umožňuje zobrazovat seznam rybářských revírů a rybářský řád. Pro samotné členy organizace obsahuje webová stránka možnost správy svých rybářských docházek. Samotné vedení organizace má také možnost tvorby, mazaní nebo úpravy členů a revírů.

POUŽITÁ LITERATURA

[1] Getting started with HTML. *Resources for Developers, by Developers* [online]. c1998–2024 [cit. 2024-03-30]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML/Getting_started

[2] SINGHANIA, Ritik. What is the Difference between HTML Elements and Tags? In: *Scaler Academy* [online]. c2024 [cit. 2024-04-30]. Dostupné z: https://www.scaler.com/topics/difference-between-html-elements-and-tags/

[3] DOMANTAS, G. What Is CSS and How Does It Work? *Hostinger tutorials* [online]. c2004-2024 [cit. 2024-03-30]. Dostupné z: https://www.hostinger.com/tutorials/what-is-css#CSS_Advantages_on_Web_Pages

[4] PARUCH, Zach. What Is JavaScript & What Is It Used For? A Basic Guide to JS. In: *Semrush* [online]. [cit. 2024-03-30]. Dostupné z: https://www.semrush.com/blog/javascript/

[5] MAKINDE, Felix. Node.js vs NPM: Understanding the Differences. In: *HostAdvice* [online]. c2024 [cit. 2024-05-05]. Dostupné z: https://hostadvice.com/blog/web-hosting/node-js/node-js-vs-npm/

[6] ELROM, Elad. *React and libraries: your complete guide to the React ecosystem*. New York: Apress, [2021]. ISBN 978-1-4842-6695-3.

[7] Ó TUAMA, Daragh. Visual Studio Code (VS Code): What It Is and Why Developers Love It. In: *Code institute* [online]. [cit. 2024-03-30]. Dostupné z: https://codeinstitute.net/global/blog/what-is-vs-code/

[8] KELLY, Tara. Useful React snippets for VS Code from the ES7 extension. *Medium* [online]. c2024 [cit. 2024-05-05]. Dostupné z: https://medium.com/@tara.kelly16/useful-react-snippets-for-vs-code-from-the-es7-extension-5b22ffc60f0c

[9] UZAYR, Sufyan bin. *Mastering React A Beginner's Guide*. CRC Press, 2022. ISBN 9781003309369.

[10] The Uncomplicated Guide To JSX Syntax. *Kinsta* [online]. c2013-2024 [cit. 2024-04-30]. Dostupné z: https://kinsta.com/knowledgebase/what-is-jsx/

[11] What Is JSX? *The Odin Project* [online]. c2024 [cit. 2024-03-28]. Dostupné z: https://www.theodinproject.com/lessons/node-path-react-new-what-is-jsx#why-do-we-use-jsx

[12] How to use React Fragments? In: *Refine* [online]. [cit. 2024-04-30]. Dostupné z: https://refine.dev/blog/how-react-fragments-is-works/#conclusion

[13] How To Install React on Windows, macOS, and Linux: How To Install React. In: *Kinsta* [online]. c2013 - 2024 [cit. 2024-04-30]. Dostupné z: https://kinsta.com/knowledgebase/install-react/#how-to-install-react

[14] MHATRE, Saurabh. Vite vs Create React App: Choosing the Right Tool for Your React.js Project. In: *Medium* [online]. [cit. 2024-04-01]. Dostupné z: https://saurabhnativeblog.medium.com/vite-vs-create-react-app-choosing-the-right-tool-for-your-react-js-project-8824411247cd

[15] SINGH, Abhishek. Create-react-app files/folders structure explained. In: *Medium* [online]. c2024 [cit. 2024-03-30]. Dostupné z: https://medium.com/@abesingh1/create-react-app-files-folders-structure-explained-df24770f8562

[16] GILL, Navdeep Singh. Understanding Reactjs Project Structure and Best Practices. *Xenonstack* [online]. 2023 [cit. 2024-03-30]. Dostupné z: https://www.xenonstack.com/insights/reactjs-project-structure

[17] RAGALA, Bala Krishna. What Is a React Component? *Knowledgehut* [online]. c2011-2024 [cit. 2024-04-30]. Dostupné z: https://www.knowledgehut.com/blog/web-development/react-component#what-is-a-component?%C2%A0

[18] ONYEKANI, Casmir. How to Use React Components – Props, Default Props, and PropTypes Explained. *Free Code Camp* [online]. c2024 [cit. 2024-04-01]. Dostupné z: https://www.freecodecamp.org/news/how-to-use-react-components/

[19] PUROHIT, Rakesh. Navigating Data Flow Between React Hierarchy Tree Components. In: *Dhiwise* [online]. c2024 [cit. 2024-04-30]. Dostupné z: https://www.dhiwise.com/post/navigating-data-flow-between-react-hierarchy-tree-components

[20] BHIMANI, Kesar. State and Props: The Pillars of React Explained. In: *Dhiwise* [online]. c2024 [cit. 2024-04-30]. Dostupné z: https://www.dhiwise.com/post/state-and-props-the-pillars-of-react-explained

[21] AYEBOLA, Joan. Prop Drilling in React Explained with Examples. In: *Free Code Camp* [online]. [cit. 2024-03-29]. Dostupné z: https://www.freecodecamp.org/news/prop-drilling-in-react-explained-with-examples/

[22] WIERUCH, Robin. *The Road to React: Your journey to master plain yet pragmatic React.js*. Independently published, 2018. ISBN 9781720043997.

[23] COCCA, Germán. How to Style a React Application: Different Options Compared. In: *Free Code Camp* [online]. c2024 [cit. 2024-05-05]. Dostupné z: https://www.freecodecamp.org/news/how-to-style-a-react-app/#how-to-style-your-react-apps-using-pure-css

[24] MALDONADO, Leonardo. Why you shouldn’t use inline styling in production React apps. In: *LogRocket* [online]. c2024 [cit. 2024-05-05]. Dostupné z:

https://blog.logrocket.com/why-you-shouldnt-use-inline-styling-in-production-react-apps/

[25] Material UI - Overview. *MUI* [online]. c2024 [cit. 2024-05-05]. Dostupné z: https://mui.com/material-ui/getting-started/

[26] What is Lucide? *Lucide* [online]. c2024 [cit. 2024-04-30]. Dostupné z: https://lucide.dev/guide/

[27] KAZMI, Abid. All React hooks in one short. In: *Medium* [online]. c2024 [cit. 2024-03-30]. Dostupné z: https://medium.com/@AbidKazmi/all-react-hooks-in-one-short-4b0ed4b5a6e4

[28] DESAI, Het. A Complete Guide to useId() Hook in React 18. *Medium* [online]. c2024 [cit. 2024-04-30]. Dostupné z: https://hetdesai03.medium.com/a-complete-guide-to-useid-hook-in-react-18-22119ecfd87f

[29] KUMAR, Akshay. A Guide to React Hooks With Examples. In: *Built In* [online]. c2024 [cit. 2024-04-11]. Dostupné z: https://builtin.com/software-engineering-perspectives/react-hooks

[30] Event Handling in React JS. *AlmaBetter Bytes* [online]. c2024 [cit. 2024-04-30]. Dostupné z: https://www.almabetter.com/bytes/tutorials/reactjs/event-handling-in-reactjs

[31] DWIVEDI, Apoorva. ReactJS Handling Events. In: *Scaler Academy* [online]. c2024 [cit. 2024-04-30]. Dostupné z: https://www.scaler.com/topics/react/event-handling-in-react/

[32] DICKSON, Boateng. How to Build Forms in React. In: *Free Code Camp* [online]. [cit. 2024-03-30]. Dostupné z: https://www.freecodecamp.org/news/how-to-build-forms-in-react/

[33] Rendering Lists. *React* [online]. 2024 [cit. 2024-04-30]. Dostupné z: https://react.dev/learn/rendering-lists

[34] OLAWANLE, Joel. React Table: A Complete Guide. In: *Hygraph* [online]. c2024 [cit. 2024-04-28]. Dostupné z: https://hygraph.com/blog/react-table

[35] MUSA, Eli. Tanstack: How To Add Tables To Your React App. In: *Open Replay* [online]. c2024 [cit. 2024-04-28]. Dostupné z: https://blog.openreplay.com/tanstack--how-to-add-tables-to-your-react-app/

[36] HARRISON, Paramanantham. React Table: A complete guide with updates for TanStack Table. In: *LogRocket* [online]. c2024 [cit. 2024-04-28]. Dostupné z: https://blog.logrocket.com/react-table-complete-guide/

[37] *React Table Tutorial - 11 - Pagination (Next and Previous)* [[video]. 1. 2020 [cit. 2024-03-15]. Dostupné z: https://www.youtube.com/watch?v=Kjj6Zi89gPc

[38] SHAH, Sanket. How to Use JSON Server in Frontend Development: A Comprehensive Guide. In: *DhiWise* [online]. c2024 [cit. 2024-05-05]. Dostupné z: https://www.dhiwise.com/post/how-to-use-json-server-in-frontend-development

[39] VINCENT ABBA, Ihechikara. React Router Version 6 Tutorial – How to Set Up Router and Route to Other Components. In: *Free Code Camp* [online]. c2024 [cit. 2024-03-29]. Dostupné z: https://www.freecodecamp.org/news/how-to-use-react-router-version-6/

[40] BHATNAGAR, Sachin. React Router - Types, Mechanism, Installation And Examples. *Knowledgehut* [online]. c2011-2024 [cit. 2024-04-11]. Dostupné z: https://www.knowledgehut.com/blog/web-development/react-router#what-is-react-router?%C2%A0

[41] LINDSEY, Emmi. React Router: HashLink vs. Link. In: *Medium* [online]. c2024 [cit. 2024-03-30]. Dostupné z: https://medium.com/@emmilindsey/react-router-hashlink-vs-link-e1e7a81bac1d

[42] OLAWANLE, Joel. Mastering React Conditional Rendering: A Deep Dive. In: *Kinsta* [online]. c2013-2024 [cit. 2024-04-30]. Dostupné z: https://kinsta.com/blog/react-conditional-render/

[43] STORYSET. Fishing rod concept illustration. In: *Freepik* [online]. c2010-2024 [cit. 2024-05-06]. Dostupné z: https://www.freepik.com/free-vector/fishing-rod-concept-illustration_32440116.htm#fromView=search&page=1&position=21&uuid=9d7e8aa3-e684-48a4-9802-4016a0fd6d25

[44] *Český rybářský svaz* [online]. c2019 [cit. 2024-05-06]. Dostupné z: https://www.rybsvaz.cz/beta/