VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2017

Lukáš Unzeitig

Tuto stránku nahradíte v tištěné verzi práce oficiálním zadáním Vaší diplomové či bakalářské práce.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 1. dubna 2017 .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 1. dubna 2017 .....

Rád bych na tomto místě poděkoval všem, kteří mi s prací pomohli, protože bez nich by tato práce nevznikla.

Abstrakt

Táto práce je výsledkem bakalářský praxe ve firmě Rieter CZ s.r.o., kde jsem pracoval na pozici softwarového vývojáře. Úlohou této praxe bylo pracovat na přidělených projektech.

Na začátku praxe jsem byl prověřen složitější úlohou, která měla ověřit moje nabité znalosti a především schopnost učit se. Po určitém čase, jsem dostal samostatný projekt, který zahrnoval tvorbu a následné testování jednotlivých komponent které jsem tvořil. Náročnost tvorby těchto komponent postupně rostla, od jednoduchých vstupů, až po složité a rozsáhlé navigace, grafy a pod. Spolu s komponentami jsem tvořil také prezentaci, kde jsem rozebíral konkurenční software a analyzoval jeho nedostatky a výhody a následně navrhl vlastní řešení s připravenými ukázkami.

Klíčová slova: React, Flux, Rieter, render, komponenta, canvas

Abstract

.

Key Words: React, Flux, Rieter, render, component, canvas

Obsah

Seznam použitých zkratek a symbolů

JS	–	JavaScript
WS	–	WebStorm
HTML	–	Hyper Text Markup Language
CSS	–	Cascading Style Sheets
SW	–	software
DOM	–	Document Object Model
MVC	–	Model-view-controller
UI	–	user interface

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Pro mou bakalářskou praxi jsem si vybral firmu Rieter CZ s.r.o., která nabízela pracovní pozici na místě webového vývojáře. Tato pozice požadovala znalosti hlavně JS, HTML a CSS.

V první části mé práce bych rád popsal firmu Rieter a popis mého pracovního zaměření. V druhé popíšu JS knihovnu react, její základní strukturu a architekturu Flux. Ve třetí popíšu projekt, na kterým dělám, a vypíšu seznam všech zadaných úloh, na kterých jsem pracoval. K těmto úlohám doložím ukázky kódů a obrázky výsledných funkcí/vizualizací. Na úplném konci zhodnotím svůj pokrok, nabyté vědomosti, využité znalosti ze studia na škole a celkové zhodnocení této bakalářské praxe.

2 Popis firmy Rieter CZ s.r.o.

2.1 O společnosti

Společnost Rieter CZ s.r.o. se sídlem v Ústí nad Orlicí je součástí švýcarského koncernu Rieter od roku 2001, který je předním světovým výrobcem strojních zařízení a ucelených systémových řešení pro textilní průmysl.

Rieter CZ s.r.o. charakterizuje silný důraz na inovace, které jsou realizovány ve vlastním vývojovém centru. Dalšími charakteristickými rysy jsou zaměření na přesnou strojírenskou výrobu textilních strojů a strojírenských komponentů při použití moderních technologií a procesů, výrobu řídících elektro rozvaděčů a kabelových svazků.

Společnost klade důraz na vysoký stupeň zákaznické orientace a budování dlouhodobých vztahů s našimi zákazníky. Plná integrace do struktur koncernu Rieter, využití zahraničního know-how a zaměření na stabilní růst dávají Rieter CZ s.r.o. velmi dobrou perspektivu dalšího rozvoje.

Obrázek 1: Logo Rietru

2.2 Popis pracovní pozice

Pracuji v týmu Vývoj elektroniky, je v něm 24 lidí kteří aktivně pracují na zákaznických projektech. Od začátku praxe jsem byl nasazený na projektu HMI tester, který měl zahrnovat tvorbu nového uživatelského webového rozhraní pro ovládání textilních strojů. Má první úloha měla vyzkoušet mé schopnosti učit se. Zahrnovala naučení se JS knihovny react, ve které jsem implementoval komplexní matici (graf).

Druhá úloha zahrnovala naučení se a implementace architektury Flux, která měla reprezentovat lokální uložiště, ve kterém se udržovali veškerá data ze stroje.

Další úlohy zahrnovali tvorbu menších jednotlivých komponent/funkcionalit jako vyskakovací modální okna, numpad, jazykových mutací a podobné , které jsem postupně implementovával do projektu.

Poslední a nejkomplexnější úlohou byla analyzování konkurenčních SW a následné navrhnutí a implementace vlastní navigace, které mělo být intuitivní, přehledné a minimalizované oproti původnímu.

3 Využité technologie

3.1 React

Jedná se o JS knihovnu, která pomáhá při tvorbě interaktivního uživatelského prostředí. Slouží pro vytváření webových komponent. V pomyslném MVC reprezentuje práve "V"neboli view vrstvu.

React používal a vyvíjel Facebook, který ho roku 2013 vypustil ven jako opensource. K dnešnímu patří tak k jedněm z nejoblíbenějších a nejaktivnějších repozitářů na GitHubu.

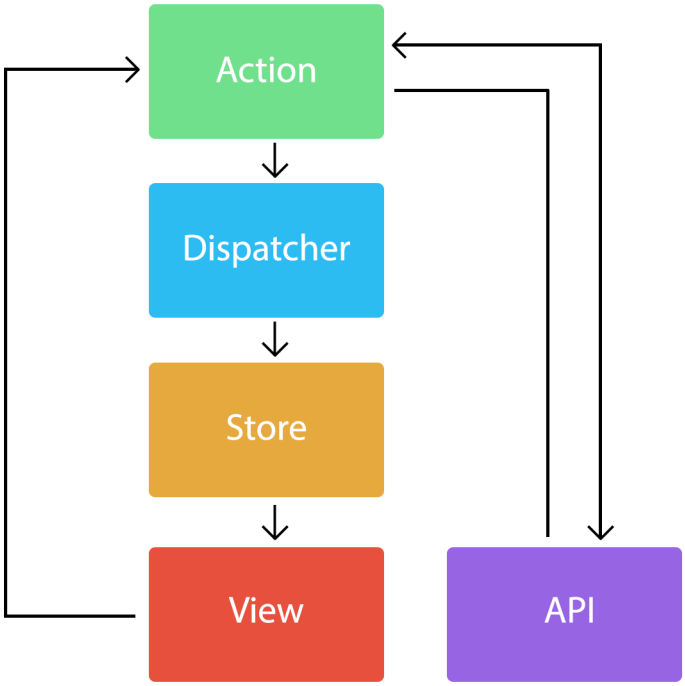
React efektivně aktualizuje a renderuje componenty, kterým se změnil stav a ostatní nechá být, čímž velmi zrychluje chod celé aplikace. Přesněji při změně stavu nějaké komponenty vytvoří nový virtuální DOM, který pomocí chytrých algoritmů porovnává se stávajícím a při nějakém rozdílu přerenderuje jenom tu změněnou část.

```
class App extends React.Component{
    constructor(props){
        super(props);
        this.state = {
            a: 0
        }
    };
    render(){
        return(
            <div> Hodnota a je : {this.state.a}</div>
        )
    }
}
```

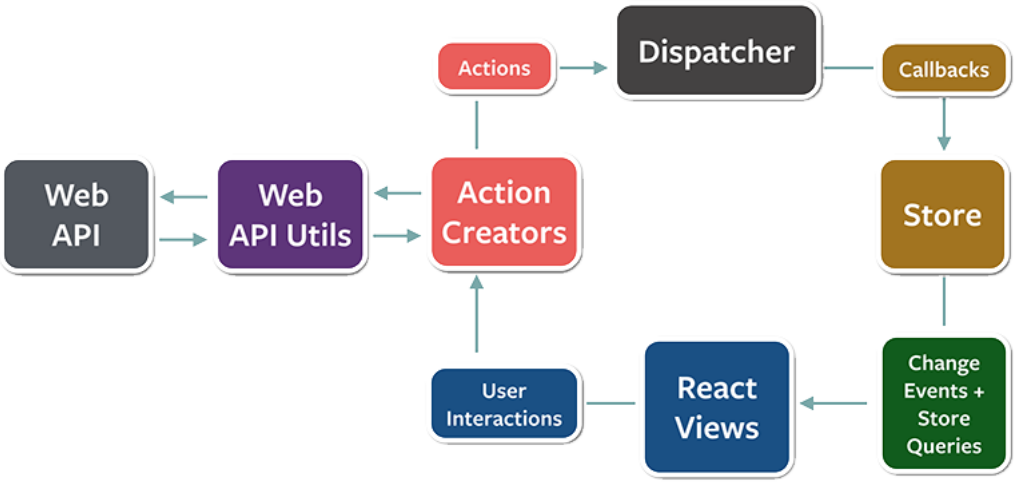
Výpis 1: Ukázka základní react komponenty psané v JSX

3.2 Flux

Flux je aplikační architektura kterou používá Facebook pro vytváření client-side webových aplikací. Aplikuje se při návrhu jednosměrném i obousměrném toku dat v UI. Vychází z předpokladu, že výsledné UI je reprezentovatelné pomocí dat, které do něj tečou z uložišť. Interakce s uživatelem je pomocí akcí, tyto akce zpracovává tzv. Dispatcher, který následně informuje o změnách všechny uložiště a je na implementaci uložiště, jestli následující akci eviduje nebo ne. Skládá se ze tří části - action, dispatcher, store



Obrázek 2: Schéma architektury Flux



Obrázek 3: Schéma architektury flux spolu s reactem

4 Mé projekty a úlohy

Tato část zahrnuje popsání projektů a seznamu všech zadaných úloh

4.1 Popis mého hlavního projektu

Mým hlavním projektem bylo navrhnout a vytvořit nové webové uživatelské rozhraní pro ovládání textilních strojů a to takové, aby bylo hlavně intuitivní, přehledné, jednoduché, moderní a rychlé. Toto rozhraní mělo být implementováno pomocí React frameworku a mělo být spojené s flux architekturou.

Původní rozhraní byla desktopová aplikace, která běžela na Quenixu. Nic méně kvůli rychlosti, omezené platformě a přístupu je toto rozhraní nahrazováno právě webovým, díky čemuž půjde na každé platformě, bude rychlejší a bude k ní snadný vzdálený přístup přes síť ať už třeba přes mobil nebo jiný počítač.

V průběhu toho projektu jsem dostal ještě jeden pro chystaný den kariéry, a to vytvoření loterie, což bylo animované kolo, které se roztáčelo vzdáleně přes mobil a losovalo ceny. Na tomto projektu jsme pracovali v týmu dva, já pracoval na samostatné webové aplikaci, která se starala o samotnou loterii, a také sledovala stavy serveru na které reagovala a po nějaké akci posílala nová data zpět. Kolega pracoval na mobilní aplikaci a samotném serveru.

4.2 Seznam zadaných úloh

Název úlohy	**Čas[h]**
Vytvoření matice pro zobrazení dat ze stroje(graf)	112
Vytvoření a připojení flux architektury k projektu	50
Vytvoření funkcionality pro jazykovou mutaci	16
Vytvoření komponent vstupů a číselné klávesnice, a následné propojení	32
Vytvoření komponenty modálního okna	16
Analýza konkurenčního SW	40
Návrhy možných navigací	60
Implementace navržených navigací	80
Tvorba prezentace návrhů	4
Tvorba sekundární navigace	30
Tvorba komponenty pro rychlé přepínání nastavení	40
Navrhnutí a tvorba ukázkové zobrazovací stránky	12
Tvorba komponenty lišty	25
Tvorba dynamicky měnícího startovací tlačítka stroje	8
Tvorba aplikace loterie	112

Tabulka 1: Úlohy přidělené a jejich časová náročnost

5 Zadané úlohy a postup jejich řešení

5.1 Překlady jazykových verzí

Zadáním této úlohy bylo vytvořit jazykovou mutaci pro 6 jazyků. Řešeních bylo mnoho, nic méně jsem pracoval v reactu a s fluxem. S mutací v reactu jsem zkušenosti neměl, tak jsem hledal řešení na internetu a na jedno narazil. Řešením bylo vytvořit json soubor, kde jsem si definoval všechny jazyky, a překlady. Nalezené řešení fungovalo s reduxem, což je architektura odvozená od flux, tudíž jsem musel kód poupravit, aby fungoval s fluxem.

```
[
   {
   "lang": "cz",
   "page": {
         "menu": { "settings": Nastaveni", "product_setting": "Nastaveni
            produktu"},
           }
   },
   {
   "lang": "en",
   "page": {
         "menu": { "settings": Settings", "product_setting": "Product settings"
            },
           }
   },
]
```

Výpis 2: Ukázka json souboru

Při přepnutí jazyka jsem vyvolal akci, která mi načetla json soubor a vyfiltrovala podle chtěného jazyka přímo objekt, který obsahoval data celé stránky v požadovaném jazyce. Tyto data se poté uložili do storu, a poslali do celé aplikace, kde každá komponenta si vzala slova z objektu, která chtěla. Výsledné řešení je nakonec velmi lehké, jasné a hlavně rychlé.

```
const content = require('../data/content.json');
var page_data = content.filter(obj => obj.lang === language)[0];
```

Výpis 3: Import jsonu a následná filtrace

5.2 Tvorba matice pro zobrazování dat ze stroje

Zadáním této úlohy bylo vytvořit komplexní matici která se chovala zároveň jako graf který se vykresloval podle hodnot zadaných ze vstupů. Samotné buňky celé matice potom uchovávají dvě hodnoty, které chodí ze serveru a nedají se v aplikaci měnit.

Implementace byla jasná, každá buňka musela být samostatný objekt, který udržoval vlastní stavy jako pozici, obě příchozí hodnoty ze serveru, barvu, velikost a pod.. Dále sloupce, který reprezentovaly graf, museli být samostatné objekty aby udržovali svoji šířku a výšku.

Otázkou bylo, jakou technologii na vykreslování zvolit, byla možnost použití samostatný divy, u kterých by byl velký problém v pozicováním. Reálným řešením tedy mohlo být buď použití SVG nebo canvas. Po delším zvažováním a hledáním jsem zvolil JS HTML5 canvas knihovnu fabricjs, která umožňovala efektivně pracovat s tvary jako s objekty. Dále umožňovala solidní interakci, volné kreslení, stylování i animace.

Samotný kód je velmi rozsáhlý. Mám komponentu pro osy X,Y. Dále pro jednotlivé sloupce grafu, pro všechny sloupce, pro jednotlivé buňky matice, pro celou mříž těchto buněk, pak i pro celou matici. Je to dohromady přes 500 řádků kódu. Tvorba této celé komponenty mi zabrala asi nejvíc času ze všech hlavně kvůli složitosti. Všechno se týkalo matematiky, dopočítávání velikostí. Největší práci jsem měl se zjištěním algoritmu, pro dopočítávání pozice dalších sloupců, a to z důvodů že požadovaná hodnota např. 50, neodpovídala 50ti bodů v canvasu a to kvůli osám, na kterých byly hodnoty, které se nepravidelně inkrementovali (2, 10, 20, 30, 50, 80, 160, 320, 500),(-30, -20, 0, 20, 40, 16, 80, 100, 120, 200). Přitom mezera mezi těmito hodnotami v grafu byla vždy o velikosti 70 bodů.

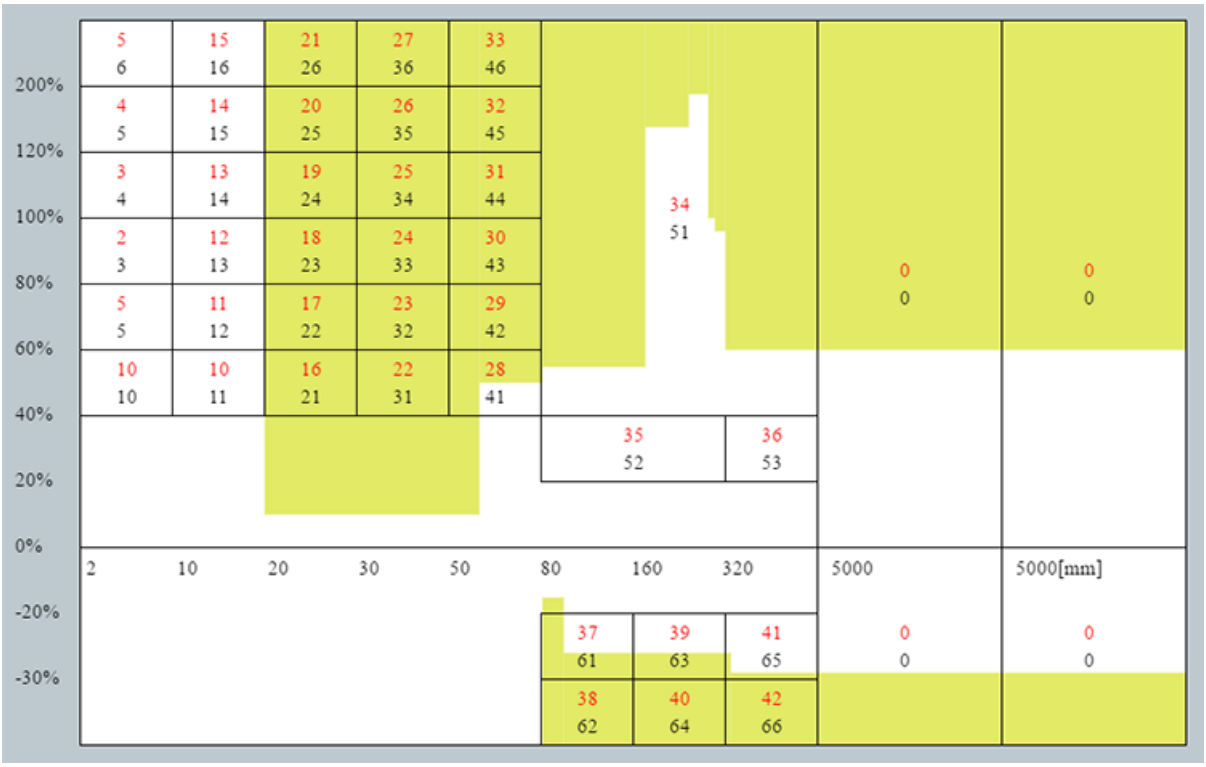
```
getX(a){
    var numbers=[2, 10, 20, 30, 50, 80, 160, 320, 5000];
    var i = 0;
   var cellWidth = 70;
    var number = 0;
    for(;i<numbers.length;i++){
      if (numbers[i] >= a)
        break;
    }
    if(i>=4)
        number = ((a-numbers[i-1])/(numbers[i]-numbers[i-1]) *(i*cellWidth -(i
            -1)*cellWidth)+(i-1)*cellWidth);
    else
      number =(a/numbers[i])*(i*cellWidth);
    return Math.round(number);
}
```

Výpis 4: Algoritmus pro převádění zadané hodnoty x do reálných hodnot v canvasu

Při implementaci této komponenty, jsem měl hned několik problému. Ze začátku samotná spolupráce fabricjs s reactem. Ikdyž react je vlastně JS knihovna a měla by spolupracovat se všemi jinými JS knihovny, vždy tomu tak úplně není. Snaha rozjet client-side knihovnu na serveru proste nešlo. Dále byl problém při posílání odkazu na renderovaný canvas, neboť jsem nemohl vzít odkaz doposud nevyrenderovaného canvasu. To bylo možné až po prvním renderu, musel jsem tedy způsobit další render celé komponenty, ve kterém se konečně přeposlala reference vytvořeného canvasu všem potomkům. Implementace tohoto problému mi zabrala spoustu času, neboť jsem s reactem začínal a to i přesto, že řešením tohoto problému bylo jen pár řádků kódu.

Dalším menším problémem bylo renderování sloupců na buňky matice, které měli barvu bílou. Požadavkem bylo, aby barva sloupců překrývala bílou barvu matice, až na text a rámečky těchto buněk. Jedinou možností bylo nastavení průhlednosti sloupců, aby byli vidět zmiňované texty a rámečky. Jenže nastavení průhlednosti tyto předměty jemně obarvovali což nebylo pěkné. Řešením toho problému nakonec bylo vytvoření bílého podkladu pro celou matici, na který se renderovali barevné sloupce a na ně pak jednotlivé buňky matice, který měli výplň zcela průhlednou.

Po vyřešení těchto problémů šla implementace celkem hladce.



Obrázek 4: Ukázka matice

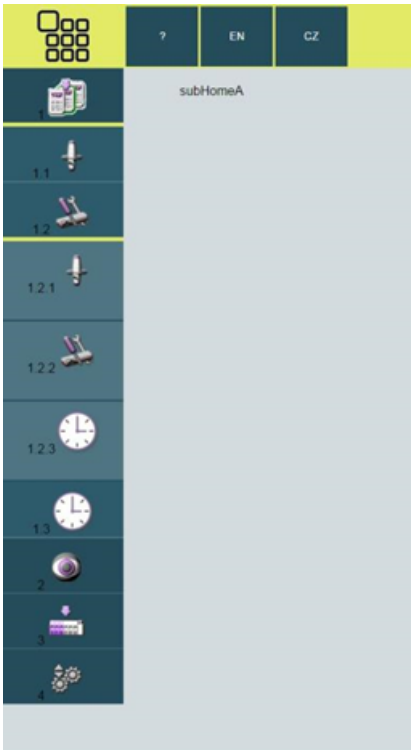
5.3 Návrh a implementace rozsáhlé navigace

5.3.1 Návrh

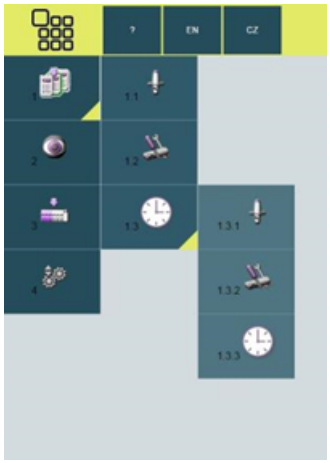
Po větších zkušenostech s reactem, jsem dostal zadání vytvoření a navrhnutí navigace. Aktuální používané menu na strojích je až čtyř úrovňové, kde pod jednou položkou může být až 8 dalších podpoložek. Navíc struktura celé navigace se mění podle přihlášeného uživatele. Pokud stroj obsluhoval zrovna běžný zaměstnanec, nabídka byla jiná, než když ke stroji přišel zaměstnanec, který stroj spravoval. Každý zaměstnanec se ke stroji přihlašoval klíčem, které rozlišovalo tyto zaměstnance.

Cílem tedy bylo vytvořit dynamické menu. Které mělo umožňovat co nejrychlejší přístup k jakékoliv položce, bylo intuitivní, jednoduché a přehledné. Dalším cílem bylo toto menu minimalizovat a to do maximální hloubky tří úrovní.

Začala tedy tvorba návrhu navigací. Navrhnout takové menu, které by splňovalo všechny tyto požadavky nebylo vůbec snadné. Hlavní problém se týkal velikosti obrazovky, kde panely na kterých běží tato aplikace mají poměr stran 4:3, rozlišeni 1024 x 768, a velikost displeje pouhých 15 palců. Jelikož se jedná o dotykové displeje, položky museli být tak velké, aby se na ně dalo jednoduše kliknout. Mé první dva návrhy se s tímto problémem právě potýkaly.



Obrázek 5: Ukázka první navigace horizontálního menu

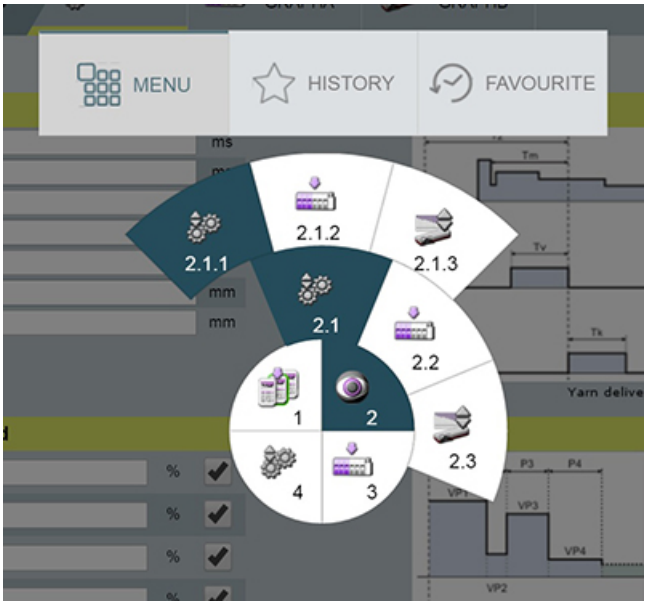


Obrázek 6: Ukázka druhé navigace horizontálně/vertikálního menu

Pro příklad při navštívené čtvrté položky první úrovně, se posouváme na čtvrtou pozici ze shora, podpoložek této položky mohlo být až 7, tudíž jsme se dostali až na 11tou pozici ze shora která je skoro až dole na obrazovce. Při tom stále jedna úroveň byla před námi, která také mohla mít klidně 7 položek.

Řešením tohoto problému bylo vytvoření modálního vyskakovacího okna, které šetřilo místo když bylo schované, a naopak když otevřené, tak mohlo být více méně přes celou obrazovku. Pro toto řešení jsem navrhl dvě odlišené navigace.

Obrázek 7: Ukázka první navigace čtvercového menu v modálním okně



Obrázek 8: Ukázka druhé navigace kruhového menu v modálním okně

Předchozí obrázek [8] řešil veškeré požadavky na navigaci. Díky svému tvaru, bylo za prvé možné opravdu velkých množství položek, v první úrovni 4, ve druhé 8, 12 atd.. Dále řešila otázku rychlého přístupu, kde při navštívené třetí úrovni, byly vidět všechny položky předchozích navštívených úrovní. Dále zde byl moderní design kruhového menu.

5.3.2 Implementace

Implementace takového kruhového menu byla velmi složitá. Hlavní otázkou bylo jak řešit vykreslování položek menu. Prvním pokusem bylo vytvoření části kruhu, který jsem nastavil jako `background-image` každému divu, který jsem posouval kolem středu a rotoval. Nic méně toto řešení bylo velmi nepřesné, a namáhavé, hrát si s každým pixelem při posouvání, pak vůbec tvorba takového tvaru. Navíc nešla efektivně měnit barva aktivní položky, neboť se jednalo obrázek.

Dalším řešením bylo stylování samotného divu, řešilo problém s obarvováním aktivních položek, nic méně vytvořit přesně chtěný tvar bylo ještě těžší, než ho ručně malovat, dělat něco takhle komplexního pouze v CSS není vůbec snadné a taky nedoporučené. Jedinou možností bylo tudíž použít něco jiného než CSS. Vybral jsem si značkovací jazyk SVG, který zvládá tvorbu takovýchto tvarů a to celkem jednoduše. Stačílo pomocí svg elementu `<path>` vytvořit takový tvar, do kterého se přidal ještě text a ikona. Pak určit jeho pozice a vykreslit. Tomuto elementu se dala za běhu programu měnit barva, tak velikost,i pozice. Pozice jsem měl pevně definované, jen jsem si je přeposílal. Kdybych tuto komponentu měl řešit znovu, vytvořil bych pouze první zavádějící pozici, a pak bych už jen ostatní kopíroval a rotoval podle středu kruhu. Tento způsob jsem použil v jiné části projektu.

Řešení dynamicky se měnícího menu, díky inspiraci z jiného projektu bylo celkem jednoduché. Vytvořil jsem si json soubor, kde jsem definoval celou strukturu menu, kde každý objekt měl

svůj název stránky, ikonu, číslo a cestu. Při kliknutí na položku v menu, se tento soubor prošel, vyhledala se požadovaná stránka a ta se poslala do zavádějící komponenty. Následně při tvarování kruhového menu se díky znalosti aktuální cesty navštívené stránky procházela struktura tohoto jsonu, a mohlo se tak formovat komponenta kruhové menu, která se vždy rekurzivně volala pro každou další navštívenou hloubku.

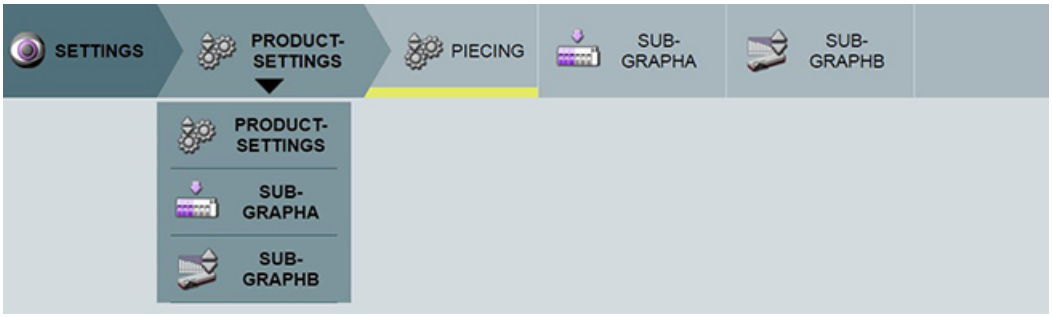
```
"settings": {
        hash: "/settings",
        name: "settings",
        page: null,
        value: "2",
        icon: "../../assets/Overview.png",
        children: {
            "product-settings": {
                hash: "/settings/product-settings",
                name: "product-settings",
                page: "ProductSettings",
                value: "2.1",
                icon: "../../assets/config.png",
                setting: ["shift","unit","group"],
                children: {
                    "piecing": {
                        hash: "/settings/product-settings/piecing",
                        name: "piecing",
                        page: "Piecing",
                        value: "2.1.1",
                        icon: "../../assets/config.png",
                        setting: ["shift","unit"],
                        children: {}
                },
            },
         },
      },
}
```

Výpis 5: Ukázka struktury celé navigace v json souboru

5.4 Návrh a řešení sekundární navigace

V předchozí úloze jsem tvořil navigaci, kterou bylo možno používat až po otevření modálního okna. Chtěl jsem tudíž vytvořit ještě jednoduchého a přehledného, co tuto navigaci doplňovalo, a která by byla permanentně zobrazena na obrazovce, nezabírala moc místa, a ve které by byl možný přechod na jinou položku v aktuální úrovni a zároveň možnosti chodu zpět. Pro toto řešení jsem zvolil vertikální menu. Do kterého jsem časem implementoval i horizontální, které rozšířilo jeho využití a to tak že umožňoval volný pohyb v aplikaci a to celkem rychle.

Obrázek 9: Sekundární vertikální menu



Obrázek 10: Sekundární vertikální + horizontální menu

Díky možnosti rozbalení předchozí navštívené položky, se zobrazila celá předchozí úroveň a bylo tak možné rychlé přesměrování jinam. Zároveň díky odlišnosti barev úrovní, je ihned vidět, v jaké hloubce jste, a které položky patří k sobě. Aktivovanou položku jsem pak ještě odlišoval žlutým podtržením.

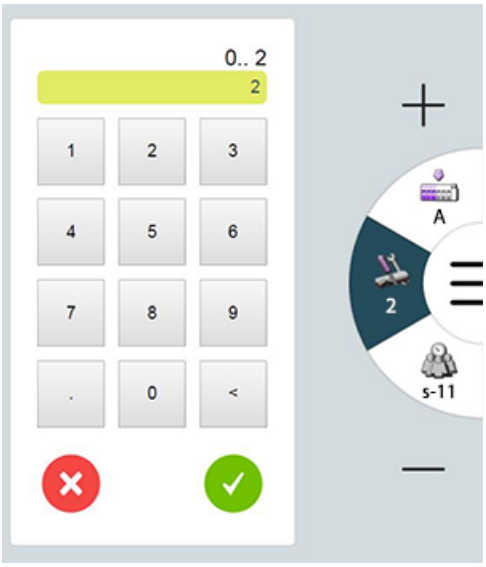
K vykreslování takového menu, jsem kombinoval prvky klasického stylování divů a SVG prvků.

5.5 Řešení přepínání rychlého nastavení důležitých dat

Jednou z nejdůležitějších věcí v celé aplikaci na které dělám je možnost rychlého přepínání mezi důležitými daty, jako je výběr směny, skupiny strojů, jednotek nebo přednastavených konfigurací. Proto řešení mělo být výrazné, přehledné a hlavně intuitivní ovládání, které umožňovalo tyto změny. Předchozí řešení v desktopové aplikaci bylo velmi nepřehledné a složité na pochopení. Změna měla přijít v nové aplikaci, kterou vyvíjím právě já.

Při řešení jsem se inspiroval u hlavního menu, které je kruhového tvaru. Navrhl jsem půl kruh, kde jeho části umožňovali aktivovat nastavení, které chceme měnit, a pak středovým kolem se otevírala možnost této změny. V určitém případě se otevírá list, který umožňuje si

vybrat určitou směnu nebo list nastavení a pod., a nebo pak v případě možnosti volby jednotky, se otevírá číselná klávesnice, která umožňuje zadávat číselný vstup.



Obrázek 11: Ukázka otevřeného číselného vstupu u vybrané volby pro změnu jednotky

Implementace potom vypadala podobně jako u kruhového menu jen byla jednoduší. Pro vykreslování všech položek v kruhu, se vytvořila pouze jedna část kruhu, která se vždy jen okopírovala od předchozí, poté se pootočila o potřebný počet stupňů, který se spočítal z počtu možnosti nastavení / 180. Začínalo se vždy od shora. Informace o tom, která obrazovka měla jaké možnosti nastavení je uložena ve vlastnosti objektu v json souboru, kde je uložena celá struktura navigace.

5.6 Tvorba komponenty lišty

Další zadanou úlohou bylo vytvořit něco, co umožňovalo rychlé přepínání mezi obrazovky, ať už se jednalo o oblíbené, nebo naposledy navštívené. První řešení, o které jsem se pokusil, bylo zakomponovat do modálního okna s klasickým menu taková menu, která právě tyto položky zobrazovala. Pro přepínání zobrazených menu zde byla tlačítka. Toto řešení zůstalo implementované v projektu, ačkoliv se nejednalo o příliš rychlé řešení, jelikož se pro přepnutí na jinou obrazovku muselo většinou provést 2-3 kliknutí, první pro otevření modálního okna, a druhé pro přechod na danou položku, ale to jen v případě že je zrovna zobrazené takové menu, které chci, jinak to bylo o jedno víc. Toto přepínání mezi obrazovky tudíž muselo jít po celý čas zobrazených stránek, aniž bych musel otevírat nějaké modální okno. Zároveň ideálně přepínat na pouhý jeden klik. Pro to mé druhé řešení bylo implementovat podobnou lištu, která se nachází na všech internetových prohlížečích, jako je Google Chrome, Mozzila Firefox, Opera a další.

Obrázek 12: Lišta umožňující přepínání mezi obrazovky

Bylo zde možné klasické přidávání nových záložek, zavírání vytvořených a přepínání mezi nimi. I přes to že výsledné řešení vypadalo skoro stejně jako u daných prohlížečů, způsob implementace a fungování byl trochu jiný. Záložky na liště fungovali jako normální tlačítka v menu, kdy při kliknutí na nějakou položku, se načetl obsah této stránky. Jinak řečeno má aplikace si nepamatovala/neukládala otevřené načtené obrazovky. Obrazovky se na novo renderovali po kliku na danou záložku. Sice se nejednalo o nejrychlejší řešení, nic méně tato implementace byla velmi šetrná k využití paměti.

Samotná implementace byla celkem jednoduchá, stačilo udržovat seznam všech otevřených lišt a zároveň informaci, která je právě zobrazena. Jediný menší problém byl opět se stylováním těchto záložek. Pro jejich tvorbu jsem použil samotné divy, které jsem následně náročně styloval. Jejich stylování jsem rozdělil na 5 částí, kde první byla hlavní střední část, další dvě byly zkosené části na krajích, a poslední dvě byli také zkosené části na krajích, které byly o jeden pixel vždy vylezlé, byly tmavší a způsobovali tak tmavý rám kolem této části.

5.7 Animované kolo-loterie

Jednalo se o úlohu patřící jinému projektu, který měl být pro den kariéry pro příchozí děti. Bylo za úkol vytvořit aplikaci, která umožňovala losování o ceny. Tato aplikace měla komunikovat se serverem a ten zas s aplikací, která běžela na firemním mobilu. Na tomto projektu jsme pracovali ve dvou, kde jsme měli jasně rozdělené úlohy. Já měl na starosti právě aplikaci loterie, která měla běžet na internetovém prohlížeči. Měl jsem představu řešení, kde by se točilo klasické "kolo štěstí", které by mělo fyziku takovou, aby se na signál startu začalo konstantní rychlostí otáčet, a po signálu zastavení postupně zpomalovat až do nulové rychlosti. Řešení toho problému vůbec nebylo snadné, a tak pro usnadnění práce jsem našel na internetu řešení, které mi ulehčilo jak práci fyzikou tak s vykreslováním kruhu. Tento skript jsem importoval do mého reactového projektu a propojil.

Vykreslení kola ve staženém skriptu bylo řešeno canvasem, které bylo voláno metodou `window.requestAnimationFrame(callback)` kde `callback` byla metoda, která vykreslovala celý kruh. Tato metoda provádí update ještě před tím, než je provedeno nové vykreslení stránky. Pro uložení cen, které byli k dispozici, jsem vytvořil objekt, do kterého jsem přidal název cen, počet dostupných, a cestu k obrázku. Problém dělalo, že při restartu aplikace, nebo pouhém obnovení stránky, se tyto hodnoty resetovali na původní. Jako řešení jsem využil lokálního uložiště v prohlížeči. Kde jsem tento objekt vytvořil a pak v aplikaci spravoval. Toto řešení právě zabraňovalo resetování hodnot objektu při pouhém obnovení stránky restartu.

Mé řešení losování bylo trochu podvodné, posílal jsem si ze scriptu do reactu vždy informaci, že se nacházím v novém políčku a ať ukážu nový obrázek. Nic méně ukazovaný / výherní předmět nebyl vůbec závislý na tom, na které políčko zrovna ukazatel ukazuje, ale na pouhé pravděpodobnosti, kterou jsem vypočítal ze seznamu zbývající cen. Pro tento výpočet jsem prvně projel objekt všech výherních předmětů, pokud jejich počet byl větší než 0, vždy jsem uložil cenu do nového pole a to tolikrát, kolik kusů ještě byl. Po takto vytvořeným poli cen, jsem jen vzal náhodnou pozici tohoto pole a vrátil ukazovaný / výherní předmět.

```
let items = [];
let counter = 0;
let itemCounter = 0;
let item;
let objet = [];
for(item in retrievedObject) {
   itemCounter++;
    if (retrievedObject[item].count > 0) {
        object.push(retrievedObject[item]);
        for(let i = 0; i < retrievedObject[item].count; i++ ){
            counter++;
            items.push(itemCounter);
        }
    }
}
let random = Math.floor((Math.random() * counter));
let index = items[random];
item = object[index-1];
```

Výpis 6: Algoritmus získávání výherní ceny

Komunikace se serverem pak probíhala pomocí běžných HTTP dotazů GET/POST, kde pomocí GET metody jsem se dotazoval na stav serveru, jestli mám losovat, nebo roztočit. POST metodou jsem potom posílal zpátky informace že jsem roztočil, že losuji, nebo že jsem vyhrál + předmět který jsem vyhrál.

Ze začátku jsem se setkával s problémem neplynulé animace, až trochu trhané. Zkoušel jsem místo vykreslování kruhu vložit obrázek, který se rotoval kolem své osy, nic méně rozdíl byl zanedbatelný. Problémem byla nejspíš hlavně velikost kruhu, kde celý kruh měl velikost 1500x1500px. Tento problém jsem řešil vykreslování pouze části kola, která byla vidět,. Toto celkem výrazně zvýšilo plynulost celé animace, kterou jsem ještě dolaďoval malými úpravami vykreslovacích funkcí.

Obrázek 13: Kolo štěstí

6 Celkové zhodnocení praxe

6.1 Uplatnění teoretických a praktických znalostí získaných ve škole

Jednoznačně nejdůležitější věcí, která mě vysoká škola naučila a tím připravila na tuto praxi, je zvládnutí samostudia. Od začátku praxe dělám v reactu, ve kterém jsem nikdy předtím nedělal, používám architektury, které jsem nepoužíval a podobně. React je samozřejmě více JS knihovna, takže základní znalosti JS nesměli chybět, také základy HTML a CSS byli velmi důležité. Tyhle požadované základy jsem pochytil v předmětu Vývoj internetových aplikací. Další důležitou věcí, kterou mě škola naučila bylo programátorské myšlení, které jsem pochytil v průběhu celého studia na škole. Velmi důležitý předměty, který mě také připravily na tuto praxi, byly algoritmy a matematika. Bez těchto dvou předmětů bych nezvládl úlohu jako například sestrojení matice, kde jsem pořád něco počítal. Dalším užitečným předmětem byl určitě Správa systému Windows, kde nás učili pracovat a vytvářet virtuální stroje. Toto jsem potřeboval když jsem si měl na svém notebooku rozjet systém pheonix, na kterém běžela předchozí desktopová aplikace.

6.2 Chybějící znalosti a schopnosti

Určitě hlavní chybějící znalost byla úplná neznalost JS knihovny react, která je přitom velmi rozšířená. Proto bych určitě ocenil více předmětů zaměřené na webové technologie.

6.3 Získané znalosti

K získaným znalostem bych určitě zařadil pokročilou znalost webových technologii jako je třeba react,flux.. Dále určitě lepší znalost CSS a HTML jazyka a zlepšení samostudia.

7 Závěr

Na začátku praxe, po zadání první úlohy matice, jsem měl obavy z tak komplexní úlohy psané v frameworku, o kterém jsem slyšel poprvé. Navíc v našem týmu jsem jediný, kdo dělá ve frontendu. Naštěstí díky mé snaze při samostudii a tendenci se zlepšit, jsem se s první úlohou vypořádal. Další úlohy pak pro mě už byly lehčí a lehčí. Dnes už tak téměř nemám žádné problémy s řešením jakékoliv úlohy které dostanu zadané.

Celkově hodnotím moji bakalářskou praxi ve firmě Rieter velmi pozitivně. Od začátku je tu skvělý kolektiv lidí, kteří mě mezi sebe okamžitě přijali. Firma mi nabídla pozici Webového vývojáře, a proto momentálně pracuji pro firmu na zkrácený úvazek zaměstnanecké smlouvy. Těším se na budoucí spolupráci s touto firmou.

Literatura

[1] Rieter History [online]. Dostupné na: http://www.rieter.com/cz/rieter/about-rieter-group/subsidiaries/rieter-cz-sro/historie/

[2] React History [online]. Dostupné na: https://en.wikipedia.org/wiki/React_(JavaScript_library)