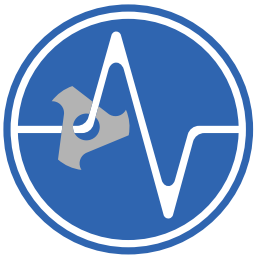


Střední průmyslová škola strojní
a elektrotechnická a Vyšší odborná škola,
Liberec 1, Masarykova 3

FRONTEND APLIKACE RENTALS V REACTU

Maturitní práce

Autor **Petr Dědic**
Obor **Informační technologie**
Vedoucí práce **Ing. Tomáš Kazda, DiS.**
Školní rok **2021/2022**

Anotace (Resumé)

Práce se zabývá přepracováním školní aplikace Rentals. Cílem je funkční frontend nasazený na backend REST full API a nový UI/UX design, který by měl být znovupoužitelný i v jiných školních aplikacích v Reactu.

Summary

The thesis deals with the redesign of the school application Rentals. The goal is a functional frontend deployed on the REST full API backend and a new UI / UX design, which should be reusable in other school applications in React.

Čestné prohlášení

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

V Liberci dne 11.03.2022 .....

Petr Dědic

Obsah

Úvod

Na začátku, ještě před psaním této práce, jsem dostal nápad – nápad, který by studentům naší školy (ale nejen jim) zjednodušil přihlašování a práci s webovými aplikacemi naší školy, jako například Práce a Praxe. Nápad byl jednoduchý, spojit UI rozhraní a smysluplně na sebe aplikace napojit.

K tématu jsem se dostal po diskusích s panem Mgr. Stehlíkem, kdy si stěžoval na některé nevýhody těchto aplikací a jejich nejednotného UI designu. Do toho pan Ing. Kazda potřeboval přepracovat frontend a backend aplikace Rentals, což se mi nádherně hodilo „do karet“.

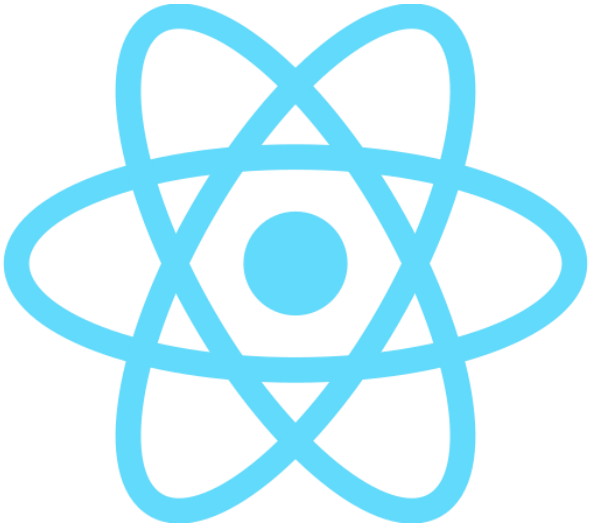
Na práci se tudíž začalo pracovat už koncem třetího ročníku a přes prázdniny jsem nabíral znalosti v tvoření aplikací v prostředí Reactu a jejich návrhu. To však není jediná důležitá část webové aplikace – důležitá je totiž i její funkčnost a přístup uživatele, na což jsem se také v této práci zaměřil.

1 Návrh a prostředí

V první části práce jsem se zaměřil na analýzu a podobu původních prací na téma frontendu aplikace Rentals. Začal jsem původním návrhem od Julie Sanetrníkové (https://github.com/pslib-cz/MP2021-JulieSanetrnikova-Redesign-UI-UX-aplikace-Rentals) a udělal si pořádek v tom, jak budu postupovat. Aplikaci jsem se rozhodl psát v knihovně React a design návrh udělat v aplikaci Figma. Práce se poté nasadí do projektu s API, který dělá Káťa Daňková současně se mnou (https://github.com/pslib-cz/MP2021-22_Dankova-Katerina_REST-API-aplikace-Rentals).

1.1 React

React je JavaScriptová knihovna pro sestavování uživatelského rozhraní (UI). Byla vyvinuta Facebookem a byla veřejně vydaná v roce 2013. Momentálně lze s jistotou říct, že se jedná o nejpopulárnější a nejvíce používanou knihovnou pro sestavování webových, ale i mobilních aplikací. Základem Reactu a jeho krásou jsou komponenty, znovupoužitelné „cihly“, které sestaví celou strukturu aplikace. Tyto komponenty jsou pouhými JavaScriptovými funkcemi, které vrací JSX – to je syntaxční způsob spojení HTML kódu a JavaScriptu. React komponenty jsou rychlé a podporují spoustu vlastností, například vnitřní „state“ a argumenty, které se dají předat v relaci rodič-potomek. Vše se vykresluje ve virtuálním DOMu, kde tyto funkce manipulují se statickým HTML kódem. Za zmínku také stojí, kdo všechno staví na Reactu své aplikace a weby: Facebook (Meta), Netflix, Spotify, … (1)



Obrázek 1 – React logo

1.2 SPA

SPA (Single Page Application) je návrh webových stránek, kde je všechen obsah předáván uživateli pouze na jedné stránce (jedné adrese, chceme-li). Tento samotný návrh je ne vždy pro webovou stránku(y) výhodný, především kvůli většímu obsahu stažených dat, protože stahujeme celý obsah stránky, a to i přesto, že navštívíme třeba jen třetinu stránky. (2)

Toto však React za pomocí virtuálního DOMu eliminuje a za pomocí různých knihoven, například React Router, je schopen vytvořit si „podstránky" tak, aby fungovaly úplně stejně (a hlavně ještě lépe) jako více stránkové aplikace. Aplikace mají poté svižné načítání, jsou rychlé, nenačítají, co nepotřebují k vykreslení a sníží se počet stránek v historii uživatele.

1.3 React a jeho ekosystém

React je open-source – to znamená, že jsme jakožto vývojáři schopni vidět jeho funkční kód a především – jsme díky tomu schopni vytvářet vlastní balíčky, které tento kód obohacují. Říkáme jim „packages". Existuje nekonečné množství balíčků, které se nám mohou hodit a které nám mohou zjednodušit práci s Reactem (state management, animace, routování, přechody, …) – proto je React tolik oblíbený a rozšířený. Můžeme si nějaké důležité a v mé práci použité balíčky vyjmenovat:

1.3.1 React Router

React Router je jeden z nejvíce používaných balíčků k routování (SPA s více stránkami ve virtuálním DOMu). Jeho design je postaven na jménech ze světa sítí, jako je třeba Router nebo Switch. Ve Switchi jsou pak komponenty Route, které představují jednotlivé stránky, které se generují např. na základě URL parametrů. (3)

1.3.2 Styled Components

Tento velmi užitečný balíček nám dovoluje vytvářet „stylované" komponenty. To jsou komponenty, které mají v JSX zápisu CSS kód a není tak potřeba vytvářet CSS soubory, vše je pak totiž na jednom místě, hezky u sebe. (4)

1.3.3 React wavy transitions

Tato aplikace od známého Instagramového influencera, který se zabývá UI a UX designem, dokáže nudné a přímé tranzice mezi změnami stránek (route) změnit v kapkovitý efekt. (5)

1.3.4 UUID

Knihovna, která nám přidá náhodně generované unikátní ID či hashe do naší aplikace. Mně osobně se hodila při tvoření dat a testování jejich chování. (6)

1.3.5 Prettier

Ačkoliv se nejedná o komponentu, ale o rozšíření IDE, stále má na mé práci velký podíl. Jedná se o formátovač kódu, který podporuje nejnovější syntaxi JS a React JSX. V mnoha částech kódu mi pomohl k lepší orientaci v textu a urychlil můj postup. (7)

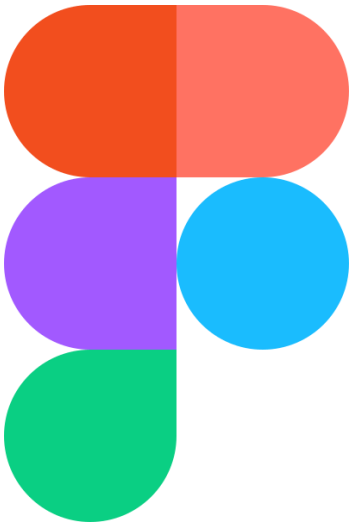
1.3.6 React Big Calendar

React Big Calendar je knihovna, která nám dovoluje pracovat s úžasně možnostmi, jak pracovat s komponentou kalendáře. Má vysokou customizaci a podporuje různé druhy zápisu času, tudíž se nemusíme starat o formát času pro zobrazení. V aplikaci v něm lze vidět časové rozmezí jednotlivých výpůjček. (8)

Při mém štěstí jsem na základním wireframe-u aplikace začal pracovat hned na začátku, což je vlastně standardní postup. Začal jsem návrhem podoby stránky a karet pomocí aplikace pro UI a UX design zvané Figma. Ve své podstatě designuji e-shop osekaný o některé funkcionality (peníze), které jsou však nahrazeny jinými (výpůjčky). Důležitou částí byla také analýza a podoba klasického e-shopu, ze kterého jsem vycházel, aplikace by však přesto měla být v prvku UI designu jedinečná ale lehká a dále rozšiřitelná.

1.4 Figma a wireframe

Figma funguje jako webová aplikace a zároveň jako aplikace, a to díky Reactu, na kterém je postavena (aplikace pro desktop je psaná v React Native). Je ji tak možné použít všude a ke svým designům a návrhům jsem měl snadný přístup. Aplikace funguje podobně jako třeba Photoshop nebo Inkscape, je však primárně zaměřena na webový design a design UI a UX u aplikací nejen pro telefony ale i pro desktop a jiné. (9)



Obrázek 2 – Figma logo

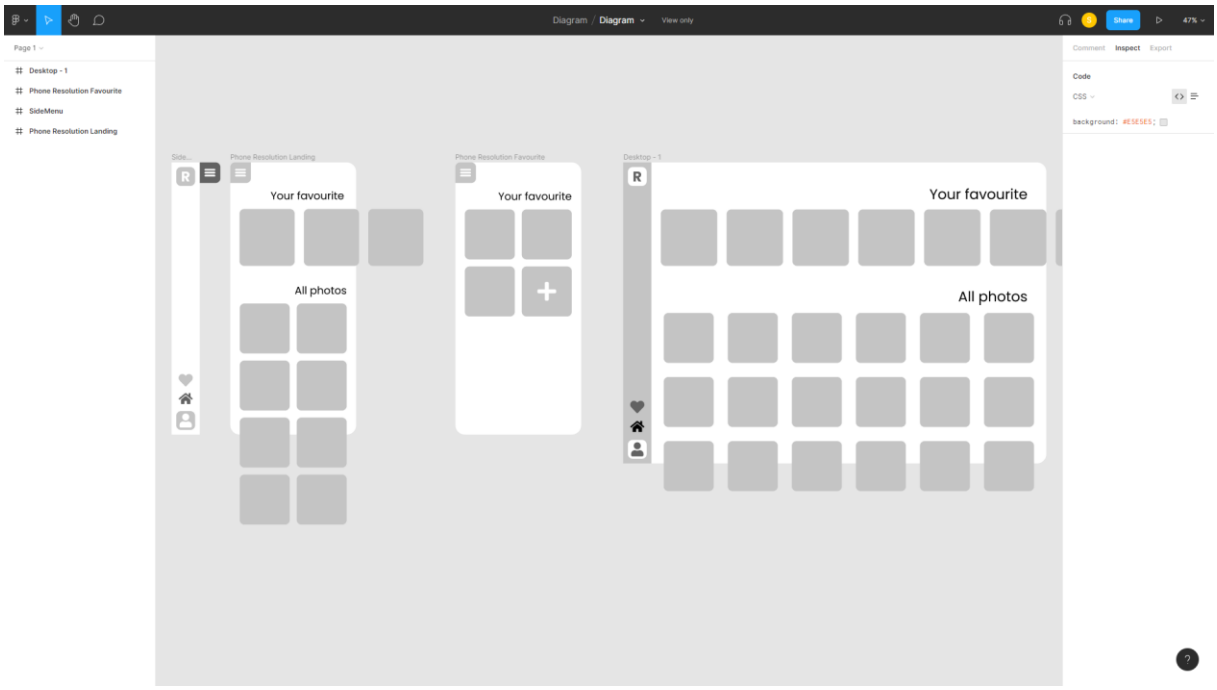
Začal jsem přihlašovací stránkou, kde jsem udělal jednoduché pozadí a tlačítko pro přihlášení, jelikož uživatel, který není přihlášený, nemá do aplikace přístup. Po přihlášení nás uvítá hlavní „home page“, kde na nás čeká veškerá nabídka předmětů k vypůjčení a jejich filtrace. Po levé straně máme k dispozici navigační menu i se základními informacemi o uživateli a funkcionalitou přihlašování.

Další stránka byla stránka s oblíbenými předměty a předměty, které si často vypůjčujeme, to kvůli zjednodušení hledání těchto předmětů. Vypadá úplně stejně jako domovská stránka aplikace.

Poté přišla na řadu stránka s košíkem, zde není moc o čem mluvit, nachází se zde seznam předmětů k výpůjčce, jejich stav ve zvolené době výpůjčky (budou obsazené?) a kalendář s možností vybrat období naší výpůjčky.

Dále máme náš profil, na kterém se nachází historie našich výpůjček, jak minulých, tak co budou a co právě probíhají. Je zde i karta s informacemi o uživateli.

Poslední a pro management a údržbu výpůjček aplikace úplně nejdůležitější stránka je stránka pro správu, přehled a manipulaci s administrátorskými právy. Zde je administrátor schopen spravovat uživatele, předměty a i výpůjčky. Tato stránka má vlastní navigační menu pro urychlení navigace a obsahuje své podstránky na podobném principu jako hlavní navigace (editace předmětů, výpůjček, přehled, …).

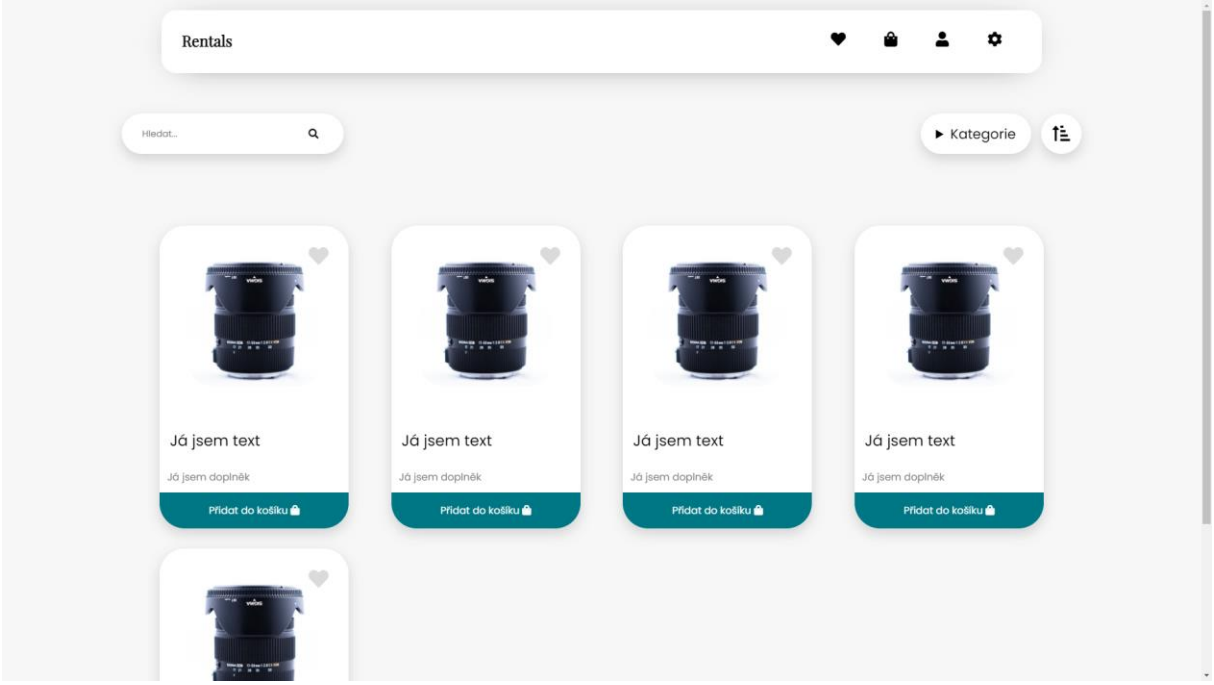


Obrázek 3 - Výstup z aplikce Figma v podobě wireframe

1.5 Z wireframu do barev

Teď začíná ta zábavná část a tou je opravdová podoba aplikace, kterou vidí uživatel, jak ze stránky barev ale i animací a celkově těch malých detailů, co dělají stránku více user friendly (UX). Při tomto jsem vycházel z původního návrhu od Julie Sanetrníkové (https://github.com/pslib-cz/MP2021-JulieSanetrnikova-Redesign-UI-UX-aplikace-Rentals) a většinu barev zvolil právě od ní.

Přemýšlel jsem i o tmavém režimu aplikace, který je u aplikací (a hlavně v Reactu, kvůli jeho lehké implementaci) poslední dobou dost oblíbený a populární. Nakonec jsem se ale rozhodl ho do aplikace nezakomponovat, a to z jednoduchého důvodu – není zde třeba, nebudeme v aplikaci trávit dlouhý čas a určitě mi to uživatelé tmavého režimu prominou. Navíc pro večerní surfování, kdy je tmavý režim také velmi populární, si člověk určitě nezvolí výpůjčky v aplikaci Rentals.



Obrázek 4 - Původní návrh předělání aplikce Rentals

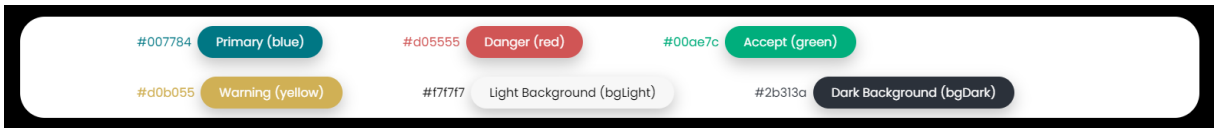
1.6 UI knihovna + komponenty

Nepostupuje většina velkých firem pod nějakým jednotným designovým stylem? A proč vlastně každá aplikace naší školy vypadá úplně jinak? – tyto otázky jsem si pokládal, a právě proto jsem se rozhodl vymyslet a usměrnit designový styl, kterým by se měly naše aplikace řídit. Pojmenoval jsem ho velice příhodně ke slangovému pojmenování naší školy – Proomka („Průmka“).

Otázka byla kam tento styl směřovat – zvolím více user-friendly design a budu se snažit působit na uživatele příjemně nebo budu vystupovat vážně a striktně? Vzhledem k zaměření na studenty naší školy jsem zvolil první možnost.

Proomka tudíž obsahuje designové doporučení a příklady použití, kterými se máme při tvorbě školních aplikací řídit. Názorné příklady na stránce Proomky mají uživatele přimět smýšlet stejně a je zde vysvětleno proč byli některé věci použity. Některé z těchto doporučení je např. zaoblování hranatých rohů u karet a příliš velkých a obdélníkových prvků nebo jejich elevace. Zde je jejich seznam a názorná ukázka:

1) Barvy – Napříč aplikacemi budete používat spoustu různých barviček. Proto je dobré si určit nějaké základní a pojmenovat je. Většinou zvolíme tři hlavní barvy a zkoušíme je na stránce použít v poměru 60/30/10.

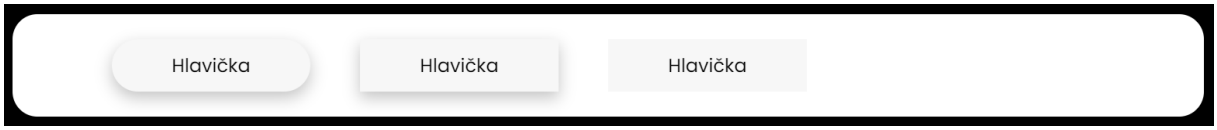


Obrázek 5 - Barvy používané napříč aplikací Rentals

2) Elevace – Při použití elevace se jedná o efekt vyzdvyhnutí nad pozadí. Jednoduše použijeme stíny, které do aplikace přidají „realičnost“ a vdechnou komponentě život. Většinou tento efekt využijeme na pozadí, co se od sebe moc neliší.

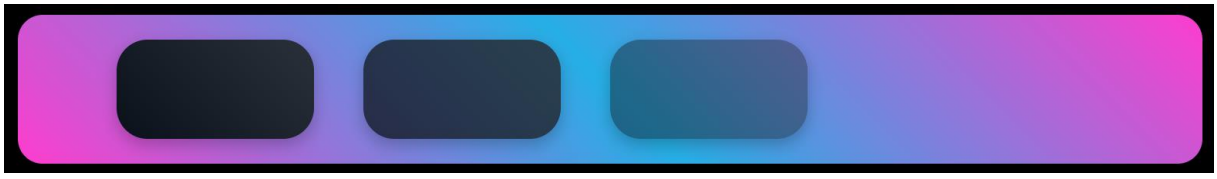
Obrázek 6 - Ukázka elevace prvků

3) Zaoblení – Zaoblené hrany se už používají skoro všude. Důvod? Vypadají dobře a přátelsky na oko. Velikost zaoblení by měla záviset na místě použití komponenty, většina případů ale použije 16px až 48px.



Obrázek 7 - Ukázka použití zaoblení prvků a elevace

4) Průhlednost – Ačkoliv průhlednost prakticky skoro nikde nevyužijeme, stále se dá využít v designu komponent "přes sebe" anebo když chceme vidět pozadí. Dobrý příklad je třeba úvodní stránka Proomky. Microsoft tento design obohatil o „frost“ efekt neboli zmražení, kdy pozadí lehce rozmažeme.



Obrázek 8 - Příklad průhlednosti a „frost“ efektu na barevném pozadí

Při tomto jsem si vzpomněl na podobné knihovny, jako je například Bootstrap. Ten se však skládá i z komponent, znovupoužitelných prvků a přesně to já jsem ve své aplikaci potřeboval. Každý návrh karty či upozornění se tak stal znovupoužitelným komponentem, který je v aplikaci Rentals a napříč Proomku použit. Při tomto jsem získal dosud mnou neznámé a zajímavé poznatky o druhu psaní a typech těchto komponent a knihoven. Zajímavá mi přišla práce s atributy, dynamickými atributy a kombinovanými komponenty (skládá se z více komponent, ale je definován jako jeden samostatný prvek). Zde jsou některé komponenty, které jsou v Proomce obsaženy:

1.6.1 Alert

Jednoduchý alert box sloužící k zobrazení stavové zprávy. Lze mu nastavit property „delay“, která určí, kdy zmizí (to je doprovázeno animací mizejícího status-baru), jinak zmizí až po kliknutí. Jsme také schopni určit barvu a barvu textu.

Obrázek 9 - Příklad alertů v knihovně

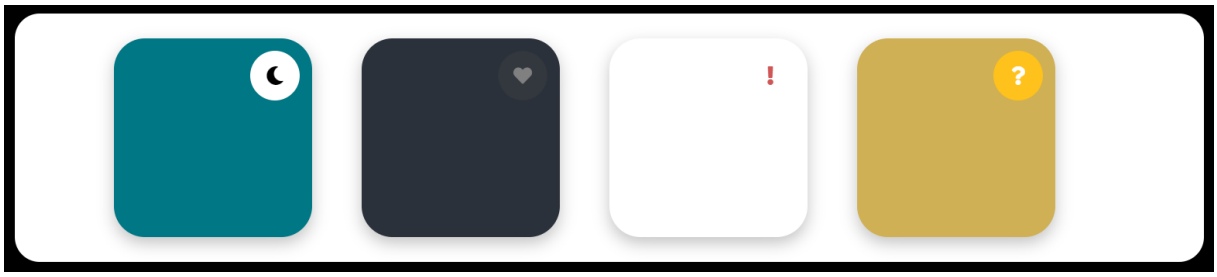
1.6.2 Badge

Komponenta pro viditelné zobrazení a upozornění na nějakou vlastnost. Z té se později vyklubalo používání jako button (ačkoliv to není to úplně správné a nebylo to tak původně zamýšlené). Pro ikony jsem použil sadu ikon od FontAwesome (https://fontawesome.com). Jsme také schopni určit barvu, barvu textu, poté barvy při hover efektu a umístění pomocí props top a right.

1.6.3 Card

Slouží jako kontejner k zobrazování obsahu na různé způsoby. V podstatě se jedná o „blbý“ prvek div, který se chová jako flex kontejner a je to „základ“ všech knihoven s komponenty.

Obsahuje také subkomponent CardMenu – to je box, do kterého můžete umístit navigační či akční tlačítka nebo prvky v kartě. V mé práci jsem ho však nepoužil.



Obrázek 10 - Ukázka použití Badge uvnitř karet

1.6.4 Modal

Modalová komponenta pro zobrazení potvrzení nebo formulářů. Vše uvnitř se zarovná na úplný střed obrazovky a není scrollovatelný. Použit byl proto, aby mohl být „nad“ listem, který se po změně v modalu aktualizuje na pozadí.

1.6.5 Switch

Toggle komponenta pro lehčí práci s formuláři. Dá se použít i místo checkboxu, ale pouze ve specifickém případě, například v nastavení. Jsme schopni nastavit barvy knobu a pozadí, stejně tak jako přiřadit funkci, která se má zavolat při změně stavu.

1.6.6 Navigation

Slouží jako otvírací postranní menu. V aplikaci nebylo potřeba využít této komponenty, jelikož jsem namísto ní použil Proomkabar k navigaci. Vyjíždí ze strany a můžeme do něj přidat navigační prvky, například redirect button.

1.6.7 Proomkabar

Zamýšleno jako jednotné menu v aplikacích od školy. Lze do něj uložit navigační prvky a slouží jako hlavní „navigační menu“ aplikací. Díky němu a jeho stavům (zavřený, zmenšený, plně otevřený) není potřeba na stránce (v aplikaci) mít navbar na obvyklém místě (nahoře uprostřed).

1.6.8 Tooltip

Jednoduchá nápověda při přejetí myší na prvek (na mobilu podržení). Lze nastavit, na jaké straně od prvku se zobrazí.

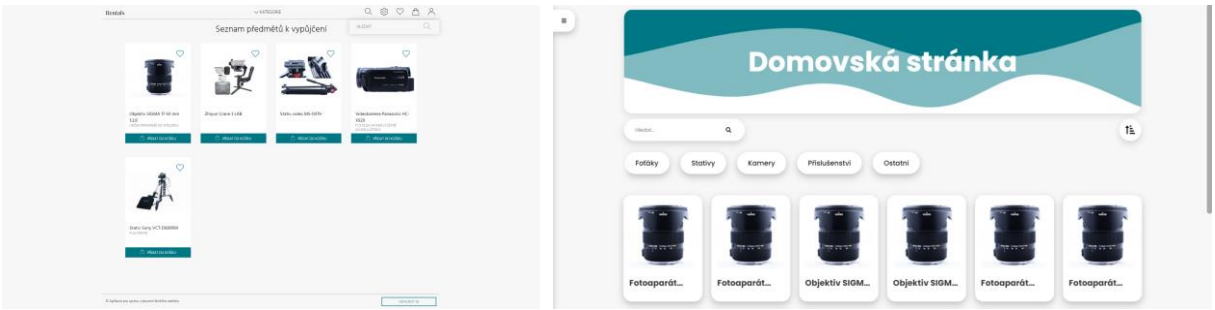
Tyto komponenty je detailně možno vidět i s popisem na stránkách dokumentace Proomky, kde je také popsáno, které třídy detailně obsahují při defaultním stavu (prosté použití): https://petrdedic.github.io/Proomka/ .

1.7 UX a animace

Pro mě osobně nejvíce zábavná část a také nejvíce náročná (opravdu to vezme spoustu času). V této části jsem svůj „focus“ zaměřil na chování stránky, interakci uživatele na stránce a také jeho chování a očekávání (co čeká od stránky, její funkčnost, chování komponent a feedback). Naštěstí jsem na mé praxi ve firmě získal tyto zkušenosti a jsem schopen lépe (samozřejmě ne perfektně) analyzovat uživatele.

1.7.1 UX

Začal jsem tím, podle mého názoru, hlavním a tím byl redesign a přetvoření stránky (tuto část jsem dělal z velké stránky současně s UI designem) z původního návrhu a začal jsem jí dělat více logicky postavenou a pochopitelnou pro uživatele. Jako příklad mohu uvést vyhledávání, které bylo zavádějící a uživatel ve skutečnosti nevěděl, kde a co v něm hledat. Dále také přepracování chaotického rozložení a nepřehledných menu pod sebou, které uživatele mátla a nutili ho rozkoukávat se kolem. Všeobecně jsem design aplikace směřoval spíše do podoby aplikace třeba mobilní a použil jsem spoustu white-space, abych zamezil překliknutí a nešťastné interakci. Uživatel má tak před sebou moderní design s prvky, které zná z jiných aplikací a ví (nebo je schopen instantně vydedukovat) jak který prvek pracuje. To samé platilo i pro uživatele na telefonech a tabletech (p. Ing. Kazda bude ovládat administrátorské záležitosti převážně z telefonu).



Obrázek 11 – Ukázka rozdílů mezi původním a finálním návrhem

Tudíž bylo logické některé komponenty přepsat tak, aby braly ohled na zařízení, na kterém se zobrazují. Tímto není myšlena výška a šířka elementů, mobilní optimalizace

tohoto typu má být dělána automaticky v CSS a Media Queries, ale jiná funkcionalita třeba v podobě scrollování do strany (to se na PC nedělá a nepoužije se, raději se zobrazí šipky na ovládání) nebo použití scroll-snap, který pomáhá projíždění dlouhých seznamů karet na telefonu. Další funkcionalitou, na kterou se často zapomíná u mobilních zařízení je vrnění.

V neposlední řadě jsem řešil také věci jako omezení dat, které uživatel stahuje (na co stahovat všechny obrázky, když zrovna potřebuji a zobrazuji jen 12?) a zobrazení v době načítání. Koukat na stupidní kolečko nikoho nebaví, přidal jsem tudíž animaci karet, když se načítá jejich obsah.

1.7.2 Animace

Je to docela zajímavé téma ve spojení s UX. Nesmíme to totiž moc přehnat, ale zase to musí být pro uživatele hned zřejmé. Tímto je myšlena například malá feedback animace při kliknutí tlačítka – změní barvu, zvětší se anebo na se pozadí udělá vlna. Neměla by to být pouze hover interakce (přejetí myši), ale na telefonu nějaké zavrnění. Můžeme sem zařadit i fade animaci, kdy prvek, který mizí, začne ztrácet barvu a stane se průhledným, dokud nezmizí. Dalším příkladem může také být „gooey“ animace Switche, kdy jakoby přeteče z jedné strany na druhou. Cílem je jednoduše omezit trhavé prvky na stránce a zpříjemnit její visuální chování, nesmí se to však přehnat, aby to nevypadlo nepřirozeně a divně.

1.8 Styled Components a má aplikace

Jak jsem již psal, Styled Components nám dovolují použít CSS zápis v JSX kódu. To by samo o sobě nebylo nic až tak zajímavého, ale ta zajímavá část se nachází trošku v něčem jiném.

Styled Components totiž umějí používat props z našeho hlavního komponentu a lze tak používat například barvy z Contextu stylů (tmavý a světlý) a také podporuje zápis s tenárním operátorem, což se extrémně hodí v případech, kdy měníme třeba podobu tlačítka podle jeho stavu (zmáčknuto, vypnuto). Navíc podporuje zápis SCSS (Sass) a ten nám tak krásně zjednoduší práci např. s vnořováním.

Někteří by však mohli navrhnout, že pro CSS kodéra, který se nemusí v JSX zápisu vyznat (ačkoliv je opravdu primitivní) a zná pouze CSS, může být zápis s JS funkcemi a operátory nepřehledný. To je naprosto validní poznámka, která je však dobře vyvrácena

stylem, kterým Styled Components fungují. Tato knihovna si totiž vytváří vlastní (dost chaoticky pojmenované) třídy, které se na nastylované komponenty aplikují. Podporuje však i vlastní pojmenování třídami, takže se nemusíme bát, že by naše karta nemohla mít třídu „card“ anebo by tato třída byla přebyta – je tudíž možné upravovat CSS kód úplně normálně, jak je u většiny zvykem. (4)

```
import styled from "styled-components";

const StyledAdminList = styled.div`
  width: ${(props) => (props.isSmall ? "100%" : "80%")};
  padding: ${(props) => (props.isSmall ? "0 " : "10%")};
  border-radius: 1.5rem;
  -webkit-box-shadow: 0 8px 20px 0px #d1d1d1;
  box-shadow: 0 8px 20px 0px #d1d1d1;

  background-color: white;
  text-align: center;
  height: auto;

  overflow-y: scroll;
  &::-webkit-scrollbar {
    display: none;
  }

  & .item-list:nth-child(even) {
    background-color: #f7f7f7;
  }
`;

const AdminList = (props) => {
  return (
    <StyledAdminList isSmall={props.isSmall}>{props.children}</StyledAdminList>
  );
};

export default AdminList;
```

Obrázek 12 - Podoba zápisu Styled Components s props a tenárním operátorem + ukázka vnoření v CSS

2 Funkcionalita aplikace

Design a návrh mám a jsem z něj schopen přejít do více technické části práce a tou je její funkcionalita. Tím je myšleno, jak bude pracovat React s daty a komunikovat s API serverem. API je RESTful a tudíž můžu očekávat určité chování a data, která dostanu dopředu. Na backend části aplikace dělala ve své práci Káťa Daňková a pomocí konzultace s ní jsem byl schopen aplikaci připravit do stavu, kdy stačí pouze připojit Axios.

2.1 Axios

Axios je React knihovna, která ulehčuje komunikaci s API přes HTTP. Jedná se o velice developer-friendly klient, který není potřeba nijak nastavovat a pomocí něj je práce s API a jeho datovými typy mnohem lehčí. (10)

```
const fetchUser = async () => {
  const { data } = await Axios.get("/api/User/Inventory/" + id, config);
  setUser(data);
  setLoading(false);
};
```

Obrázek 13 – Ukázka funkce na získání dat pomocí Axios

2.2 Context

Přemýšlel jsem, jestli ve své práci použít globální úložiště Reactu zvané jako Context. Ten funguje na princip toho, že se nachází v hierarchii React komponentů úplně na vrcholu a funguje jako obal, který uchovává a je schopný poskytnout komponentám, které obaluje, svůj obsah. Nemusíme však mít pouze jeden Context, různých obalů můžeme použít kolik chceme, jen se to většinou kvůli props a state komponent nepoužívá, jelikož nám ve většině případů dává větší smysl použít je. Většinou se snažíme ukládat nějaká globální data, která se hodí ve více místech a ve více komponentech, například přihlášeného uživatele nebo režim aplikace.

V aplikace je tudíž tato funkcionalita použita pro uložení přihlášeného uživatele. Ten je uložen do Contextu pomocí knihovny k autentizaci zvané „oAuth“. Za pomocí kódu od p. Mgr. Stehlíka a knihovny „OpenID Connect (OIDC) Client“ jsem napojil Context na oAuth server a s API pracuji pomocí JWT tokenu (hashe).

2.3 Filtrování kategorií

Různé předměty jsou součástí různé kategorie, stejně jako u e-shopu, je dobré rozlišovat skupiny mezi foťáky, stativy a držáky. Samo o sobě by filtrování jedné skupiny nebyl problém, controller na to je, ale „sranda“ začíná v momentě, kdy kategorie spojím. V aplikaci jsem tudíž pro každou zvolenou kategorii udělal vlastní zavolání do API a tyto položky poté přidal do jednoho listu a udělal z nich list bez duplikátů. Následně bylo potřeba filtrované předměty oddělit od ostatních.

```
const [SelectedCategories, setSelectedCategories] = useState([]);
const [filteredFiles, setFilteredFiles] = useState([]);
const EditSelectedCategories = (newItem) => {
  if (SelectedCategories.indexOf(newItem) === -1) {
    SelectedCategories.push(newItem);
    setFilteredFiles(
      storedFiles.filter((x) => SelectedCategories.includes(x.categoryId))
    );
  } else {
    SelectedCategories.splice(SelectedCategories.indexOf(newItem), 1) &&
      setFilteredFiles(
        storedFiles.filter((x) => SelectedCategories.includes(x.categoryId))
      );
  }
  console.log(SelectedCategories);
};
```

Obrázek 14 – Ukázka kódu na filtrování kategorií a položek v nich

2.4 TUS

TUS je protokol pro práci se soubory a jejich nahráním do databáze. Běží na základech http protokolu a je jednoduše implementován. Jelikož je používán velkými firmami jako je Google nebo Vimeo, je hodně populární a optimalizován. Já jsem ho v aplikaci vytvořil pro administraci a editaci obrázků předmětů k vypůjčení. TUS jsem jednoduše napojil na backend od Káti Daňkové, která pro mě připravila veškeré controllery v API a já tak pouze posílal obrázek a následně ho zobrazoval. (11)

2.5 Dokumentace stránek aplikace

Pro navigování na stránce slouží v levé části aplikace otevírací menu. Tím lze změnit URL adresu stránky a aplikace nám tak otevře konkrétní stránku, a to i přesto, že

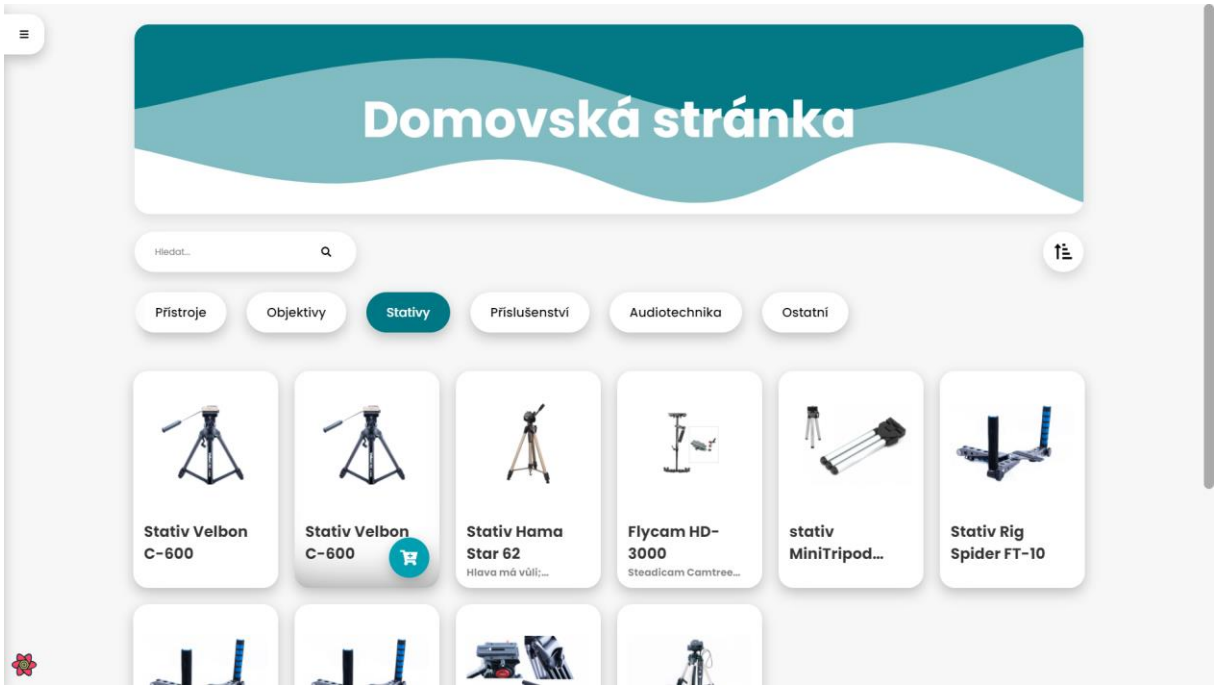
je aplikace SPA. Při načtení stránek vyčkává vykreslovací metoda na příjem accessTokenu pro komunikace s databází (a ověření uživatele) a po načtení tokenu vytváří Axios dotazy.

```
const FetchItems = () => {
  return useQuery(
    "items",
    async () => {
      const { data } = await Axios.get("/api/Item", config);
      return data;
    },
    {
      // The query will not execute until the userId exists
      enabled: !!accessToken,
    }
  );
};
```

Obrázek 15 - Query čekající na accessToken

2.5.1 Domovská stránka

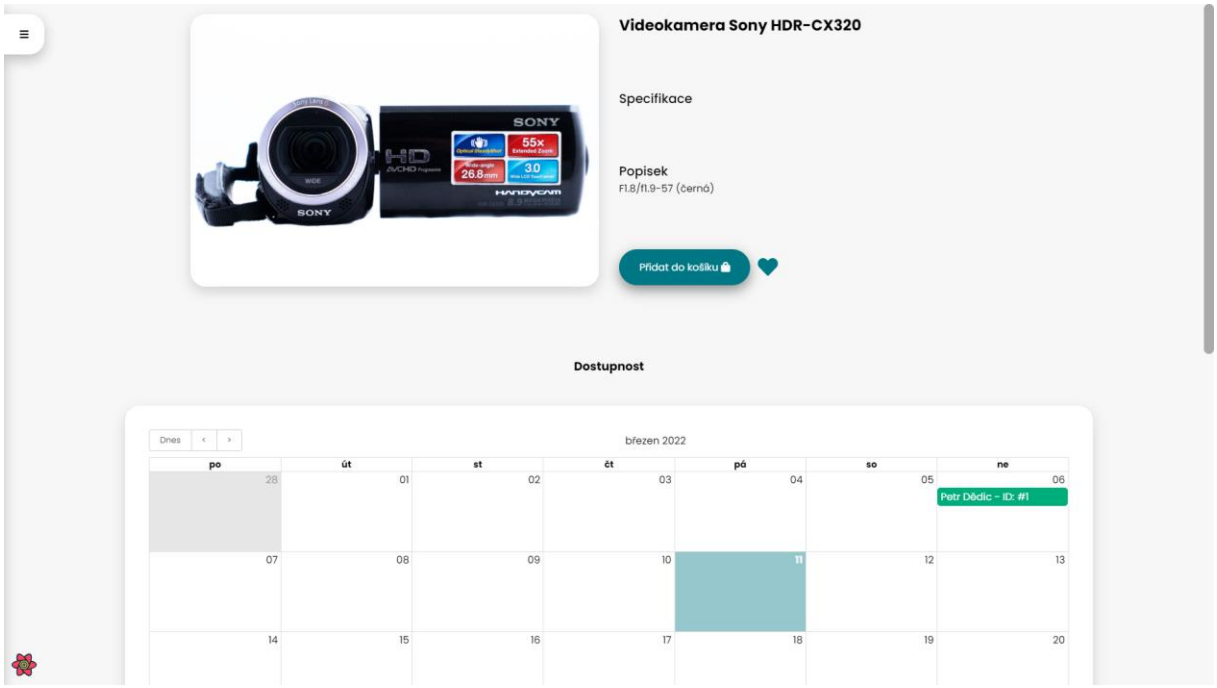
Na této stránce se nám zobrazí všechny předměty, které si momentálně můžeme vypůjčit. Je zde filtrování podle kategorií (i více naráz) a hledání podle jména předmětu. Předměty jsme schopni přidat díky hover tlačítku do našeho košíku.



Obrázek 16 - Hlavní stránka s prvkem přidání do košíku a filtrováním

2.5.2 Detail předmětu

Na stránku s detailem předmětu se dostaneme po kliknutí na obrázek v kartě z hlavní stránky. Na této stránce lze vidět kalendář s daty, kdy je předmět nedostupný pro výpůjčku a též se zde nachází předměty, které jsou kompatibilní. Předmět zde můžeme též přidat do oblíbených a tím si ho rychleji zpřístupnit na stránce „Oblíbené“, je-li předmět oblíbený, je srdíčko, které nám tento stav sděluje, (indikátor) vybarvené. Předmět zde též můžeme přidat do košíku.



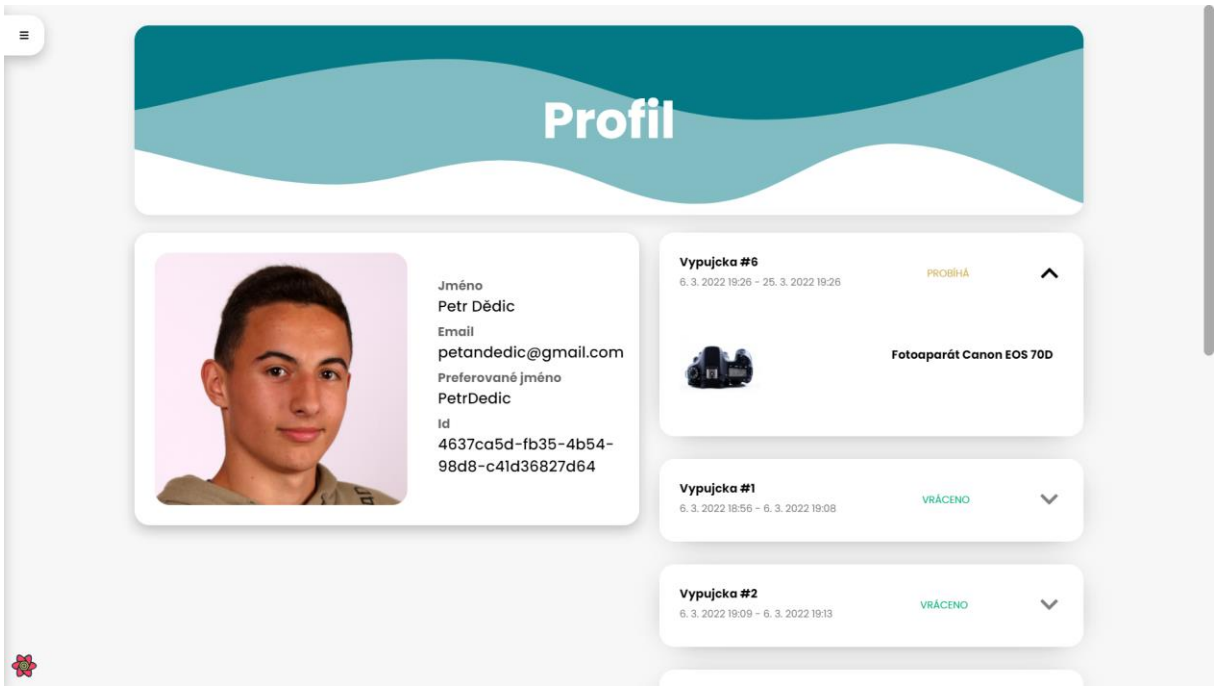
Obrázek 17 - Detail předmětu s kalendářem dostupnosti

2.5.3 Oblíbené

Stránka je stejná jako „Domovská stránka“, pouze zde nejsou předměty všechny, ale pouze naše oblíbené. Jsme schopni se odtud dostat na detail předmětu nebo ho z oblíbených odstranit.

2.5.4 Profil

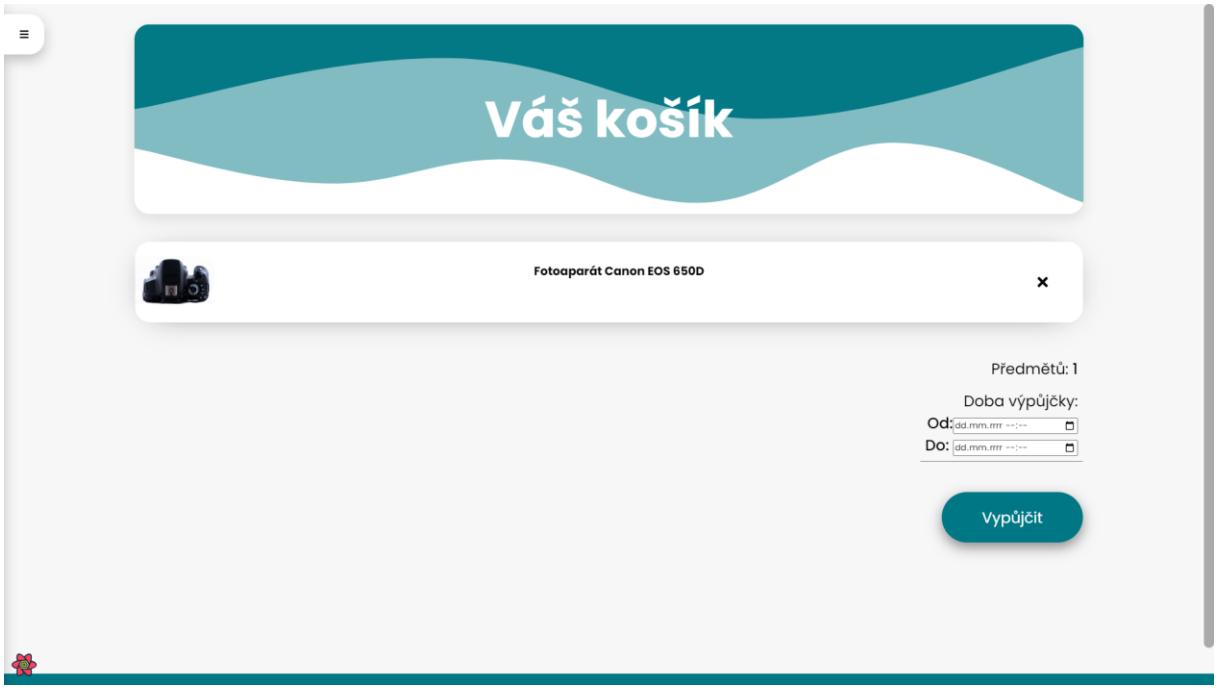
Stránka „Profil“ slouží k zobrazení informací o momentálně přihlášeném uživateli. Je zde také seznam všech výpůjček, které uživatel má a lze zobrazit jejich informace, stav a obsah pomocí drop-down tlačítka.



Obrázek 18 - Profil s historií výpůjček

2.5.5 Košík

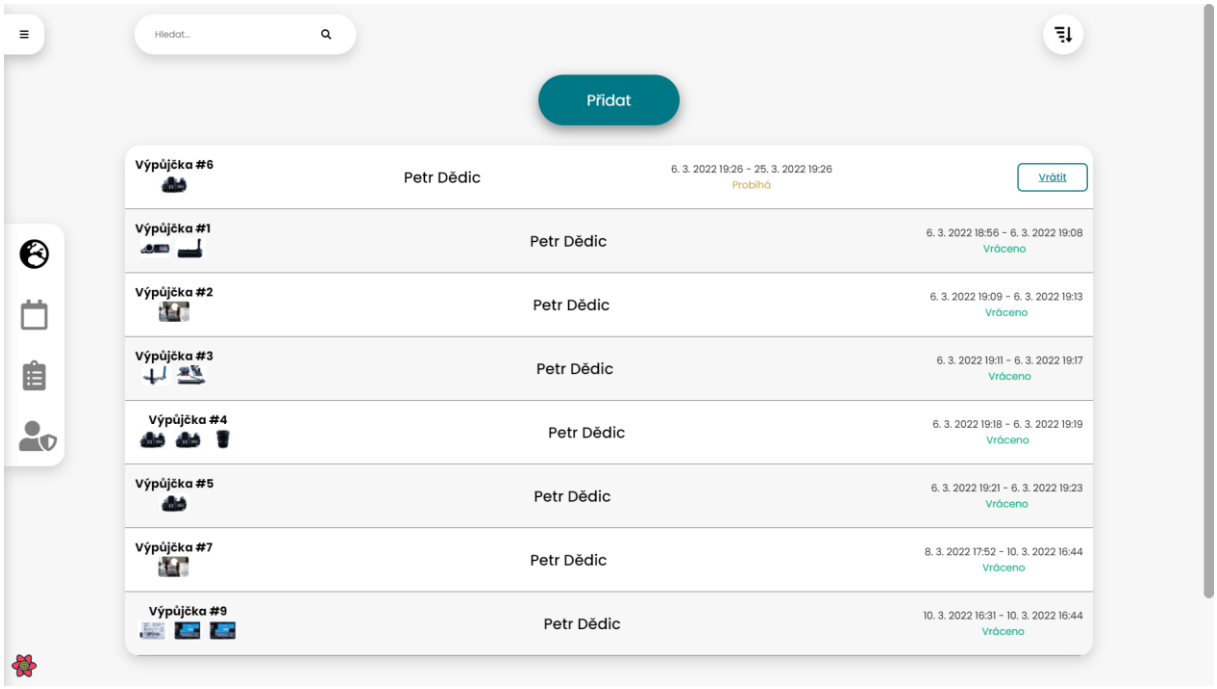
Na této stránce se nachází uživatelův košík s předměty. Jsou zde detailně vypsány a jsme také schopni vybrat, od kdy do kdy bude výpůjčka probíhat.



Obrázek 19 - Uživatelův košík

2.5.6 Admin

Na této stránce provádí administrátor všechny potřebné operace. Má zde k dispozici seznam všech výpůjček a jejich správu, dále jejich kalendář, správu všech předmětů (i těch smazaných) a přehled nad všemi uživateli a jejich předměty (inventáři). K navigaci mezi těmito podstránkami mu slouží další postranní menu.



Obrázek 20 - Administrátorský přehled nad výpůjčkami

3 Napojení na backend – vložení do projektu, publikování

Celý můj projekt byl nakonec vložen do projektu většího – toho, který dělala Káťa současně se mnou. Visual Studio nám je schopno vytvořit projekt, který obsahuje jak část s API, tak část s Reactem, vše, co mi tedy stačilo udělat bylo, že jsem svůj projekt vzal a nakopíroval ho do toho Káti.

Některé balíčky však bylo potřeba doinstalovat a zároveň bylo také potřeba „dopojit" proxy, kterou Visual Studio v projektu používá pro správnou funkčnost stránek a controllerů API. To se sice může jevit jako více práce, ve skutečnosti jsem však z prázdného template-u potřebné soubory dokopíroval.

3.1 Publikování knihovny na NPM (Node Package Manager)

Svoji knihovnu jsem chtěl používat a sdílet s ostatními, proto jsem nechtěl stále někoho nutit stahovat si soubory z mých projektů. Při případné úpravě by byli všichni nuceni všude stáhnout ode mě nové soubory, ale museli by i spoléhat na moji časovou kapacitu a ochotu jim nové soubory dávat.

Využil jsem tedy toho, co používá spousta knihoven a balíčků, a to je správce balíčků od Node.js. Všechny balíčky dostupné na NPM jsou open source, stejně tak jako moje knihovna komponent, která zde byla publikována.

3.1.1 Publikace a sestavení

Po dobu publikace knihovny jsem postupoval podle návodu ze stránky gitconnected (https://levelup.gitconnected.com/publish-react-components-as-an-npm-package-7a671a2fb7f). Oproti normální publikaci projektu (buildu) bylo potřeba změnit, jak se bude projekt sestavovat. Je zapotřebí přepsat default Babel script (Babel je kompilátor Js), který komponenty sestaví do jednotlivých souborů použitelných v prostředí prohlížečů. Každý prohlížeč podporuje jinou verzi JS (ECMAScriptu) a my tudíž určíme minimální verze prohlížečů.

```
{
 "presets": [
  [
   "@babel/env",
    {
     "targets": {
     "edge": "17",
     "firefox": "60",
     "chrome": "67",
     "safari": "11.1"
      },
   "useBuiltIns": "usage",
   "corejs": "3.6.5"
    }
],
   "@babel/preset-react"
]
}
```

Obrázek 21 - Konfigurace Babel pro deployment

Po nastavení Babel-u musíme projekt buildnout (sestavit) do výše zmíněných verzí JS. Náš projekt se po úpravě build kódu sestaví do složky „dist“, ve které je v podobě námi nadefinovaného nastavení.

```
"build": "rm -rf dist && NODE_ENV=production babel src/lib --out-dir
dist --copy-files";
```

Obrázek 22 – Script pro sestavení projektu

Po sestavení stačí nastavit v package.json jméno, verzi a popřípadě changelog. Nyní jsme schopni projekt publikovat (musíme mít účet na NPM) a používat ho kdekoliv pomocí příkazu „npm install“.

4 Spojení aplikací od školy

Téma zní zajímavě a těžce, pokud si pod tím tedy něco představíme. Ve zkratce jde o snahu spojit všechny React aplikace naší školy a nasadit je do jedné globální aplikace, která bude uchovávat důležitá globální data, jako je například přihlášený uživatel a režim aplikace (tmavý/světlý/hybridní).

Původně jsem ani nevěděl, jak toho docílit a jak si bude aplikace posílat data, ale nakonec jsem po dlouhém používání Googlu a vlastní představivosti objevil pár zajímavých způsobů, jak toho docílit.

4.1 MicroFrontends

Na stránce https://dev.to/kpiteng/microfrontends-with-react-47jb je popsán způsob provedení tohoto spojení aplikací a je nazván MicroFrontends. Abych popsal přesnou funkcionalitu tohoto řešení bych musel jít hodně do detailu a odbočil bych od téma této práce, tudíž ve zkratce:

Máme tři aplikace (kontejner a dvě aplikace, které budou běžet „v něm“). Pomocí package react-app-rewired přepíšeme defaultní build config aplikací, které budou uvnitř naší hlavní aplikace a nastavíme každé vlastní port (běží totiž na stejné adrese[!]). V kontejnerové aplikaci načítáme upravené buildy našich dvou aplikací a zařídíme zde mount/unmount logiku a předáváme aplikaci (a zároveň z ní bereme) jméno, hosta (port) a history props jako parametry.

Všechny aplikace běží nezávisle na sobě, jsou to normální React aplikace, a tak zamezujeme, že chyba v jedné aplikaci zásadně, a pokud vůbec, hlavní aplikaci, a i ostatní omezí. Na stejném principu funguje třeba Facebook nebo Instagram, kdy jednotlivé týmy spravují jednotlivé části aplikace a neriskují tak nefunkčnost ostatních částí (Facebook Stream. Facebook zeď, Facebook MarketPlace).

Někdo by se však mohl ptát, jak tyto aplikace navzájem komunikují, každá přeci sdílí data s tou druhou?

4.2 Sdílení dat

Ve skutečnosti máme způsobů víc, jedním je mít globální úložiště někde na serveru a ptát se na všechny informace z každé aplikace přes API – to by šlo a bylo by to nejvíce efektivní, ale zabilo by to náš druhý cíl – globální úložiště dat uvnitř kontejnerové aplikace.

Pokud se však zamyslíme, co mají všechny tyto aplikace společné, moc nás toho nenapadne. Někdo by mohl navrhnout localstorage, ale ten funguje pro každý port zvlášť (alespoň v Google Chrome, který používám). Poslední možností, která mě napadla, je objekt window. Ten přeci vidí každá z jednotlivých aplikací a k localstorage a ukládání dat může mít tak mezi sessions mít přístup jen naše kontejnerová aplikace. Tento postup jsem vyzkoušel a osvědčil se jako funkční, i když netuším, jak ho testovat pro slabší zařízení a prohlížeče, které pracují jinak s localstorage. Celkově jsem tedy vytvořil spíše koncept něčeho, co by se dalo použít, ale potřebovalo by to hodně odladit a dovymyslet v detailech.

Závěr

Na závěr mé práce jsem dokázal nebo se aspoň pokusil splnit věci, které jsem si „zadal“. Zkoušel jsem práci obohatit i o více technickou část (viz. Kapitola 4.2 Sdílení dat) a snažil jsem se o grafické spojení školních aplikací. Kdybych měl v práci pokračovat bez časového limitu, určitě bych dále rozšířil komponenty a jejich funkcionality v mé knihovně a některé z nich opravil, případně přepsal. Při objevování způsobů tvoření knihoven a NPM packages jsem totiž nebyl nikým a ani nikde definitivně usvědčen, který způsob psaní je ten správný a na svůj názor jsem přišel až při psaní.

Dále jsem, bohužel, kvůli některé slabší podpoře nového React Router v6 ze strany některých komponent byl nucen použít React Router v5. Ten sám o sobě není špatný a není vůbec špatné na něm stavět, ale použít nejnovější majoritní verzi by bylo samozřejmě lepší.

Dále bych určitě rád prozkoumal a dotáhl MicroFrontends a sdílení dat, jsou to pro mě, a i ostatní, neprozkoumané vody a ten koncept, ačkoliv nedotažený a jen těžce testovaný, mi přišel více než zajímavý a užitečný. Časem by se dala přepracovat (ve své podstatě jen změnil způsob) získávání dat například u aplikace Praxe nebo Práce, aby se dala jednoduše překopírovat a vložit do kontejnerové aplikace. Zajímavé by také bylo fungování těchto aplikací, každá by totiž mohla mít svoji vlastní subdoménu, i když netuším, jak by pak byli na stejné adrese atd.

Cíle, které jsem si zadal, byli buďto plně nebo pokud budu do výsledku počítat i následnou úpravu alespoň z vyšší části naplněny. Postupoval jsem dle mnou rozložené časové osy, ovšem v posledních pár týdnech se na mě nakupili problémy s API. Ty jsem však po diskusích s vedoucím práce a Káťou byl schopen vyřešit relativně rychle.

Nejaktuálnější verze aplikace Rentals je zde ve „velkém“ projektu: https://github.com/pslib-cz/MP2021-22_Dankova-Katerina_REST-API-aplikace-Rentals/tree/Try/Rentals_API_NET6

Stránka pro dokumentaci komponent a knihovny: https://petrdedic.github.io/Proomka

Odkaz na stažení knihovny: https://www.npmjs.com/package/proomkatest

Poděkování

V této části bych měl poděkovat lidem, kteří mi v práci pomáhali a podporovali mě.

Ing. Tomáš Kazda – Mému vedoucímu práce především za volnost, kterou mi při práci dal. Svěřil mi projekt a postup s vypracováním nechal zcela na mně, což mi dost pomohlo.

Mgr. Michal Stehlík – Panu učiteli děkuji především za čas, který byl schopen obětovat mým dotazům ohledně designu a podob stránek. Dále také děkuji za rady a vychytávky, které se mnou sdílel a hodně tak napomohl svižnému tempu psaní práce.

Bc. Ondřej Jodas – Mému „domácímu“ pomocníkovi za rady při návrhu některých funkcí a případné připomínky u některých zvolených postupů. Díky jeho radám a zkušenostem se práce dobře psala a v prostředí, ve kterém práce vznikala, se díky jeho přítomnosti pracovalo celkově skvěle.

Seznam obrázků

Použité zdroje

1. **Fireship.** React in 100 Seconds. *YouTube.* [Online] 8. srpen 2020. [Citace: 15. únor 2022.] https://www.youtube.com/watch?v=Tn6-PIqc4UM.

2. **Wikipedia contributors.** Single-page application. *Wikipedia.* [Online] 3. 2022 2022. [Citace: 8. 2022 2022.] https://en.wikipedia.org/wiki/Single-page_application.

3. **Remix.** React Router. *React Router.* [Online] [Citace: 11. leden 2022.] https://reactrouter.com/.

4. **Components, Styled.** Styled Components. *Styled Components.* [Online] [Citace: 2. březen 2022.] https://styled-components.com/.

5. **Harris, Joe.** react-wavy-transitions. *npm, Inc.* [Online] 2. srpen 2021. [Citace: 10. březen 2022.] https://www.npmjs.com/package/react-wavy-transitions.

6. **Tavan, Christoph.** uuid. *npm, Inc.* [Online] 8. prosinec 2020. [Citace: 10. březen 2022.] https://www.npmjs.com/package/uuid.

7. **Prettier.** Prettier · Opinionated Code Formatter. *Prettier.io.* [Online] [Citace: 10. březen 2022.] https://prettier.io/.

8. **react-big-calendar.** *npm, Inc.* [Online] 11. březen 2022. [Citace: 11. březen 2022.] https://www.npmjs.com/package/react-big-calendar.

9. Wikipedia contributors. Figma (software). *Wikipedia.* [Online] 10. únor 2022. [Citace: 8. bžezen 2022.] https://en.wikipedia.org/wiki/Figma_(software).

10. Zabriskie, Matt. Promise based HTTP client for the browser and node.js. *Axios.* [Online] [Citace: 10. březen 2022.] https://axios-http.com/.

11. tus.io. tus - resumable file uploads. *Tus.io.* [Online] [Citace: 10. březen 2022.] https://tus.io/.

A. Seznam přiložených souborů

Na přiloženém datovém nosiči se nachází následující soubory a složky:

MP2021-22_Dědic-Petr_Frontend-aplikace-Rentals-v-Reactu.docx

- Obsahuje tento dokument, editovatelná verze

MP2021-22_Dědic-Petr_Frontend-aplikace-Rentals-v-Reactu.pdf

- Obsahuje tento dokument, tisknutelná verze

Aplikace

- Zdrojové kódy v projektech