FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Rozšíření aplikace pro podporu zakázkové výroby luxusních hodinek
Student:	Tomáš Pospíšil
Vedoucí:	Ing. Michal Valenta, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Webové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2021/2022

Pokyny pro vypracování

Funkční prototyp aplikace pro podporu výroby luxusních hodinek vznikl v rámci diplomové práce Vojtěcha Polcara. Jedná se o webovou aplikaci, kde pro frontend je použit framework React, backend je napsán v jazyku Java a data jsou spravována databázovým strojem PostgreSQL.

Postupujte v těchto krocích:
1. Seznamte se stávajícím stavem projektu a vývojovým prostředím určeným pro vývoj a nasazení systému a zdokumentujte je.
2. Společně s vedoucím práce zvolte rozšíření, která v rámci bakalářské práce implementujete. Vycházejte přitom z priorit zadavatele (firma Prokop&Brož) a již rozpracovaných požadavků (issues) ve vývojové platformě (Gitlab).
3. Zvolená rozšíření implementujte, otestujte, zdokumentujte a nasaďte do ostrého provozu v součinnosti se zadavatelem.

Elektronicky schválil/a Ing. Jaroslav Kuchař, Ph.D. dne 19. ledna 2021 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Rozšíření aplikace pro podporu zakázkové výroby luxusních hodinek

Tomáš Pospíšil

Katedra softwarového inženýrství
Vedoucí práce: Ing. Michal Valenta, Ph.D.

13. května 2021

Poděkování

V první řadě bych rád poděkoval svému školiteli Ing. Michalu Valentovi, Ph.D. za odborné vedení, pomoc a ochotu při zpracování mé bakalářské práce. Rovněž bych rád poděkoval svému kolegovi Ing. Vojtěchovi Polcarovi za pomoc a cenné rady. Poděkování také patří mé rodině, která mě podporovala po celou dobu mého studia i při tvorbě této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principu při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisu. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisu, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programu, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli zpusobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelum). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným zpusobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 13. května 2021 . . . . . .

Odkaz na tuto práci

Pospíšil, Tomáš. *Rozšíření aplikace pro podporu zakázkové výroby luxusních hodinek*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Abstrakt

Tato bakalářská práce se zabývá rozšířením webové aplikace pro výrobu luxusních hodinek. Součástí je analýza předchozího řešení, návrh nového designu, návrh funkcionalit a implementace rozšíření. Společně s přepracováním částí aplikace obsahující informace o hodinkách je systém rozšířen o správu souborů a generování PDF dokumentů.

Pro vývoj byly použity technologie React a Java Spring. Výsledná aplikace je připravena pro nasazení do produkčního prostředí.

Klíčová slova informační systém, webová aplikace, React, formuláře, generování PDF

Abstract

This bachelor's thesis is dedicated to the expansion of a web application focused on the manufacturing of luxury watches. This thesis provides an analysis of a previous solution, and then outlines the design of visuals and functionalities of a new proposed solution. A component of the system containing descriptions of watch models is reworked, and is expanded by adding functionalities for file management and PDF document generation.

This project was developed using React and Java Spring technologies. The resulting application is ready to be deployed for commercial use.

Keywords information system, web application, React, forms, PDF generation

Obsah

Seznam obrázků

Seznam tabulek

Úvod

V dnešní době je spousta menších firem s daty a dokumenty pouze na disku nebo cloudovém úložišti. Tento formát ovšem nemusí být vždy pro firmu ideální, nemusí poskytovat potřebné funkcionality pro efektivní fungování firmy, nebo se s větším počtem dat stane nepřehledným ne-li úplně nepoužitelným. Proto je rozumným řešením nechat si vytvořit informační systém na míru, který bude obsahovat pro firmu rozumné rozdělení dat a dokumentů v rámci systému, společně s funkcionalitami na míru pro správný chod firmy. Právě takový systém byl implementován v rámci diplomové práce Ing. Vojtěchem Polcarem pro menší českou firmu Prokop & Brož zabývající se výrobou luxusních hodinek na míru.

V rámci bakalářské práce jsem navazoval na předchozí řešení informačního systému. V jednotlivých kapitolách mé práce došlo k refactoringu neefektivně řešených částí a implementaci požadavků, které nebyly uskutečněny v předchozím řešení. Během tvorby proběhlo několik schůzek s představitelem firmy, kde došlo k detailnímu projednání požadavků.

První kapitola nastiňuje jednotlivé cíle, kterých jsem v rámci bakalářské práce chtěl dosáhnout. V druhé kapitole jsou představeny převzaté a přidané technologie společně s principy, které byly využity pro tvorbu klientské a serverové části aplikace.

Třetí kapitola obsahuje představení zadavatelské firmy, jak vznikla a detailnější popis čím se vlastně zabývá. Nastíněno je i předchozí řešení aplikace a návrh nových řešení do budoucna včetně jejich benefitů prospěšných pro firmu. Součástí jsou i grafické návrhy předělávaných částí společně s návrhem tisknutelného reportu.

Ve čtvrté kapitole jsou nastíněny vybrané části implementace, jakými jsou například dynamické generování rozsáhlých formulářů, změna uživatelského rozhraní nebo generování tisknutelných reportů ve formátu PDF.

V poslední části práce jsou popsány testovací procesy dosaženého stavu aplikace, včetně jejího nasazení a dokumentace.

Kapitola 1

Cíl práce

Cílem bakalářské práce je rozšíření a nasazení informačního systému pro podporu zakázkové výroby luxusních hodinek. Dílčími cíli jsou zefektivnění ukládání dat na straně databáze a úprava neefektivního uživatelského rozhraní. Dalším cílem je vytvoření tisknutelného reportu dle přání zadavatele, který je na obchodních schůzkách předáván klientovi.

Výsledné změny budou prezentovány představiteli společnosti a finální výrobek bude nasazen do reálného provozu a povede k usnadnění evidence, celého procesu tvorby hodinek a jejich prodeje.

Kapitola 2

Technologie pro implementaci

V následující kapitole jsou popsány převzaté technologie z původního řešení a technologie přidané v rámci rozšíření aplikace.

2.1 Frontend

Frontendem nazýváme část webové aplikace nebo webové stránky, se kterou může uživatel interagovat. Zahrnuje veškerý design a může obsahovat i větší část aplikační vrstvy, které následně pouze komunikují se serverovou částí ve formě Rest API. [3]

Je vyvíjen v následujících technologiích:

- **HTML** - neboli „*HyperText Markup Language*“ je značkovacím jazykem pro tvorbu webových stránek.

  „HyperText“ označuje odkazy, díky kterým se můžeme v dokumentu přesouvat například ze stránky na stránku.

  „Markup“ označuje značky, neboli „tagy“, které slouží k rozložení stránky a označení jejich prvků. Nejčastěji se jedná o párové tagy, kde první je počáteční a druhý ukončovací TEXT , označující odstavec, ale může obsahovat i tagy nepárové viz. </br>, označující zalomení řádku.

- **CSS** - neboli „Cascading Style Sheet“ je formátovací jazyk sloužící ke grafické úpravě zobrazovaných webových stránek napsaných v jazyce HTML nebo XML. Nastavuje například barvy, velikost a styl textu, rozmístění objektů na stránce a mnoho dalších.

- **JavaScript** je skriptovací jazyk webových stránek, který se stáhne společně s HTML a běží v prohlížeči uživatele bez komunikace se serverem. Doplňuje HTML o funkcionality, které se v něm vytvořit nedají, jako jsou animace, kontrola správně vyplněného formuláře, renderování samotné aplikace a další. [4]

2.1.1 React

React je JavaScriptová knihovna, která je vyvíjena Facebookem a komunitou vývojářů, pro tvorbu uživatelského rozhraní. Framework se zaměřuje pouze na „View“ vrstvu architektury MVC, která prezentuje data uživateli. [5]

Je založený na znovu použitelných, nezávislých komponentách, což si můžeme představit jako jednu část uživatelského rozhraní (např. navigační lišta, nebo list přátel na Facebooku), které dohromady vytváří komplexnější uživatelské rozhraní. Každá aplikace má hlavní „Root“ komponentu, na kterou jsou navázány child komponenty.

Komponenta je nejčastěji JavaScriptová třída obsahující state a render funkci. Ve state jsou uložena data, která chceme zobrazit ve chvíli, kdy je komponenta renderovaná a render funkce složící k vytvoření takzvaného „React Elementu“. [6]

2.1.1.1 Document Object Model

Document Object Model je objektová reprezentace obsahující strukturu HTML nebo XML dokumentu v paměti. Pomocí API rozhraní samotného DOMu můžeme přistupovat k jednotlivým objektům a dle potřeby je upravovat pomocí skriptovacích jazyků, v našem případě pomocí JavaScriptu, nebo jim přidat event handlery, které se spouští, pokud je například do textového pole zadán znak nebo došlo ke kliknutí na tlačítko.

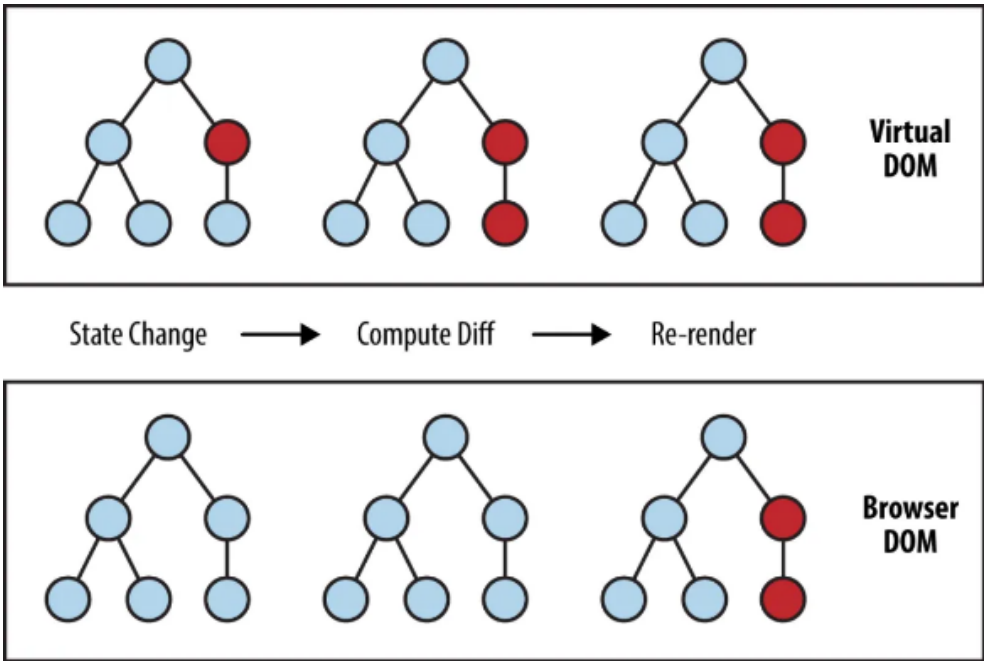
DOM je reprezentován ve stromové struktuře, kde jednotlivé HTML elementy představují uzly. [7]

Pokaždé když dojde ke změně stavu aplikace, mění se také DOM a tyto časté akce snižují výkon. Samotná změna objektu je rychlá, ovšem následně dochází k novému vykreslení všech „child elementů“ a to při rozsáhlých strukturách značně ovlivňuje finální výkon.

2.1.1.2 Virtual DOM

Spousta JS frameworků aktualizuje DOM častěji, než je nutné, proto React prosadil virtuální DOM (VDOM). Jedná se pouze, jak název napovídá, o virtuální kopii DOMu, která se při změně nejprve mění místo něj a vypočítává nejlepší možnou metodu, kterou je možné výsledné změny docílit. Virtuální DOM je tedy stejný jako DOM, ovšem nemá možnost přímo měnit to, co uživatel vidí. Proto jsou změny VDOMu mnohem rychlejší.

VDOM vzniká ve chvíli, kdy je přidán nový element do uživatelského rozhraní, zachovává stejnou stromovou strukturu jako DOM. Dojde-li ke změně stavu v uzlu, jsou změny vytvořeny v novém VDOMu. Změna stavu uzlů je reprezentována červenou barvou viz. obrázek 2.1. Poté dojde k porovnání nově vytvořeného VDOMu s původním VDOMem pomocí „diff“ algoritmu. Původní strom provede re-render změněných objektů a následně React aktualizuje pouze tyto objekty v DOMu. [8]



Obrázek 2.1: Průběh změny DOMu za pomoci VDOMu [1]

2.1.2 TypeScript

TypeScript je open-source programovací jazyk vytvořený společností Microsoft, který je nadstavbou JavaScriptu. Je rozšířen například o statické datové typy, třídy a tak podobně. To pro programátora znamená, že nemůže do proměnné typu číslo vložit text a naopak. Může si nastavit typ návratové hodnoty funkce a typy vstupních parametrů, díky čemuž se dokáže vyhnout chybám při vývoji aplikací. TypeScript oproti JavaScriptu provede kontrolu kódu ještě před spuštěním, aby odhalil potenciální chyby na základě typů proměnných, jedná se o takzvaný „static type checking“. [9]

V TypeScriptu můžeme libovolně používat syntaxi JavaScriptu, jelikož TypeScript je po kompilaci převeden na JavaScript za pomoci „transpileru“. Transpiler je druh kompilátoru, který vezme originální kód a přetvoří ho na stejně fungující kód v jiném jazyce. Nejde ovšem převést vlastnost typování. Pokud bychom kompilovali TypeScript jako JavaScript, mohlo by dojít k chybám. Kód by byl zkompilován bez errorů. Bylo by možné vkládat text do proměnné pro číslo a to by mohlo ohrozit chod programu. Kód by byl v JavaScript spustitelný, ale TypeScript kompilátor by nenechal program zkompilovat a upozornil by na chyby.

2.1.3 React Query

Jedná se o React knihovnu sloužící k načítání, cachování a synchronizaci dat ze serveru. Bez této knihovny byl nejčastěji status načítání dat řešen změnou globalního stavu pomocí knihovny „React Hooks“ 2.1.4.

```
1     const [isLoading, setIsLoading] = useState<boolean>(false);
2     const [isError, setIsError] = useState<boolean>(false);
3     const [data, setData] = useState<>();
4
5     const getData = async () => {
6       setIsLoading(true);
7       try {
8         const response = await axios('url');
9         setData(response);
10      }
11      catch (e){
12        setIsError(true);
13      }
14      setIsLoading(false):
15    }
```

Zdrojový kód 2.1: Příklad načítání dat bez knihovny React Query

K dotažení dat ze strany serveru a následnému uložení do proměnné je potřeba definovat 3 staty pomocí `useState` hooku 2.1.4. První pro zjištění zda při stahování nedošlo k chybě, další pro informace o dokončení stahování a poslední pro uložení dat viz. kód 2.1. To vede k častému opakování kódu, který může znepřehlednit tvořenou aplikaci. Po stažení dat nelze určit, zda data jsou stále aktuální, nebo zda došlo k nějaké změně na straně serveru. [10]

Knihovna pomůže odstranit mnoho opakujících se a nepřehledných řádků kódu, které jsou nahrazeny jednoduchou a přehlednou logikou. Díky možnosti vynechání stavů, může dojít k úspoře paměti aplikace a ke zrychlení aplikace samotné.

```
1     const { isLoading, error, data } = useQuery('fetchData',
2     () => axios('url')
3     );
```

Zdrojový kód 2.2: Příklad načítání dat s pomocí knihovny React Query

Knihovna React Query disponuje dvěma základními hooky, kterými jsou `useQuery` a `useMutation` .

2.1.3.1 useQuery

Jedná se o funkci, která bere dva vstupní parametry. Prvním je unikátní klíč, díky kterému od sebe dokážeme odlišovat jednotlivé hooky a druhým parametrem je asynchronní funkce. Výstupem této funkce jsou buď data získaná ze serveru nebo je navrácena chyba (error).

Pro usnadnění práce s daty jsou vráceny uživatelem vybrané hodnoty. [11] Jedná se například o:

- **data** - na počátku je hodnota nastavena na `undefined`, jinak obsahuje poslední úspěšně stažená data
- **error** - chybová hláška, která vznikla při komunikaci se serverem
- **isError** - `boolean` hodnota, označující zda při stahování dat došlo k chybě
- **isLoading** - `boolean` hodnota, označující zda dochází k prvotnímu stahování dat
- **isSuccess** - `boolean` hodnota, oznamující dokončení stahování
- **refetch** - manuální funkce pro znovunačtení dat
- **isFetching** - `boolean` hodnota, označující zda dochází k manuálně vyvolanému znovunačítání dat [12]

`isFetching` se v praxi využívá jako podmínka při vykreslování komponenty v aplikaci. Pokud bude `isFetching == true`, nedojde k vykreslení například tabulky, ale jakmile dojde ke stažení dat, hodnota `isFetching` se změní na `false`, čímž dojde k vykreslení komponenty. Knihovna zajišťuje opětovné načítání dat a tím je udržuje neustále aktuální.

2.1.3.2 useMutation

Funkce `useMutation` obsahuje jeden vstupní parametr, kterým je asynchronní funkce. Využívá se k manipulaci dat na straně serveru, nejčastěji pomocí HTML metod: `POST`, `PUT` a `DELETE`.

```
1   const [saveData, {isSuccess, isError}] = useMutation(
2   (formData) => axios.post('api/forms', {formData}),
3   {
4     onError: () => {
5       console.log("Proces byl zastaven");
6     }
7   }
8   );
9
10  saveData(formData);
```

Zdrojový kód 2.3: Ukládání dat na server pomocí useMutation

Stejně jako u `useQuery`, je možnost vracet vybrané hodnoty, např.:

- **error** - chybová hláška, která vznikla při komunikaci se serverem
- **isSuccess** - `boolean` hodnota, označující úspěšné provedení requestu
- **onError** - funkce, která se spustí ve chvíli, kdy vznikne chyba
- **onSuccess** - funkce, která se spustí, když dojde k úspěšnému provedení requestu

- **saveData** - označení query, kterým je následně voláno jako funkce v kódu [13]

2.1.4 React Hooks

Jedná se o knihovnu, která umožňuje přidání a využití hooků do funkčních komponent aplikace. V praxi to znamená, že již není třeba deklarovat třídu pro jednotlivé komponenty a dostačující je pouze funkční komponenta.

Funkční komponenty jsou pro programátora jednodušší na implementaci. Jsou tvořeny menším množstvím kódu, což napomáhá celkové přehlednosti aplikace. React sám sliboval zlepšení výkonu funkčních komponent v novějších verzích Reactu. [14]

Hooky jsou funkce, přidané do Reactu ve verzi 16.8, které nám umožňují ovlivňovat životní cyklus a stav funkční komponenty.

2.1.4.1 useState

Jedná se o hook, který nastavuje lokální stav funkční komponenty a poskytuje dva parametry. Prvním je funkce, díky které můžeme hodnotu hooku měnit a druhým je hodnota hooku. Při deklaraci je možnost nastavit typ a počáteční hodnotu.

```
1    //deklarace hooku
2    const [isVissible, setIsVissible] = useState<boolean>(true);
3
4    setIsVissible(false);
```

Zdrojový kód 2.4: Příklad využití funkce useState

2.1.4.2 useEffect

Jedná se o hook, který nám dovoluje vytvářet „side effects“. To jsou operace, které dokáží ovlivňovat další komponenty, ale nemohou být spuštěny během renderu. Jedná se například o stahování dat nebo manuální změny v DOMu. Nejčastěji tohoto bylo dosaženo pomocí metody `componentDidMount`, která byla zavolána ve chvíli, kdy byl nově vzniklý objekt přidán do DOMu. Další možností byla metoda `componentDidUpdate`, která reagovala na změny vzniklé v objektu. Hook dokáže reagovat na změny proměnných, inputů nebo je volán při každém přerendrování stránky.

V `useEffectu` je možné přidat do hranatých závorek libovolné proměnné, na jejichž změnu bude hook zavolán viz. zdrojový kód 2.5.

2.1.5 JSX

JSX (JavaScript XML) je rozšíření do JavaScriptu, které rozšiřuje jeho syntaxi a připomíná šablonovací jazyk. Popisuje, jak by mělo uživatelské rozhraní

```
1   //effekt se provede pri kazdem renderu
2   useEffect(() => {
3     console.log("effekt");
4   });
5
6   //probehne pouze pri prvnim renderu
7   useEffect(() => {
8     console.log("effekt");
9   },[]);
```

Zdrojový kód 2.5: Příklad využití funkce useEffect

vypadat pomocí HTML tagů a JSX je následně konvertováno do JavaScriptu. JSX se stal oblíbeným rozšířením, jelikož vytvoří menší a přehlednější kód oproti tvoření React Elementů za pomoci JavaScriptu. [15]

```
1   //vytvoreni React Elementu pomoci JSX
2   <MyButton color="blue" shadowSize={2}>
3   Click Me
4   </MyButton>
5
6   //zkompilovano do JavaScriptu
7   React.createElement(
8   MyButton,
9   {color: 'blue', shadowSize: 2},
10  'Click Me'
11  )
```

Zdrojový kód 2.6: Převedení JSX do JavaScriptu

2.1.6 React Hook Form

React Hook Form je knihovna sloužící k tvorbě a validaci formulářů. Odlišuje se od ostatních knihoven pracujících s formuláři především využíváním referencí místo neustálé kontroly změny stavu komponenty. Díky tomuto přístupu je finální aplikace rychlejší, z důvodu ne tak častého rerenderování komponent. Umožňuje změnit podobu výsledného formuláře přímo v implementaci. [16]

V knihovně je implementován hook `useForm()`, který obsahuje metody jako jsou `handleSubmit`, `register` a `errors`. Programátor si tedy tyto metody nemusí implementovat sám. Výsledný kód je čistší a přehlednější oproti jiným knihovnám.

2.1.7 Redux Form

Redux Form je knihovna využívána k tvorbě formulářů, jejichž stav je následně uschováván v aplikaci. Redux Form se skládá ze tří součástí. První je `Redux Reducer`, který reaguje na změny provedené ve formuláři a přepisuje stav formuláře. Druhou částí je `reduxForm()`, který obalí vytvořený formulář

do Higher Order Component (funkce, která vezme jako parametr komponentu a vrátí nově vytvořenou komponentu). HOC formuláři předá nové funkcionality pomocí *props*. Poslední součástí je komponenta `Field`, která slouží ke spojení vygenerovaných input fieldů do jednoho celku, který je uložen ve stavu aplikace. [17]

2.2 Backend

Backendem označujeme část webové aplikace nebo programu, ke které uživatel nemá přístup a nemůže ji měnit. Jelikož v současnosti je většina webových stránek dynamických (jejich data se mění a je potřeba je ukládat v databázi), využívají se skripty, které ze strany serveru získávají požadovaná data, pro zobrazení výsledné stránky uživateli. [18]

Příkladem může být nákup uživatele na eshopu. Pokud uživatel vyplní registraci, nebo provede nákup, prohlížeč pošle požadavky na backend, který je vyhodnotí a provede změny na frontendu. Všechny kroky, které se provedou před zobrazením stránky, jsou součástí backendu.

2.2.1 Spring

Spring je aplikační framework pro jazyk Java, který se využívá při vývoji J2EE aplikací (součást platformy Java, využívaná pro vývoj podnikových aplikací a informačních systémů). Zastřešuje implementaci aplikační vrstvy aplikace, aby se vývojáři mohli soustředit na tvorbu obchodní logiky projektu. Spring je rozdělen do několika knihoven v závislosti na požadovaných funkcionalitách tvořeného projektu. V projektu byla využita například knihovna **Spring MVC**, která je založena na MVC architektuře. **Spring MVC** se stará o jednotlivé HTML požadavky na základně vytvořeného API. [19]

2.2.2 PostgreSQL

Jedná se o pokročilý, open-source objektově-relační systém, který využívá jazyk SQL, včetně jeho některých pokročilých funkcionalit (např. subselect, trigger, view). Většina funkcionalit jazyka SQL zůstala stejná, ovšem existují i takové funkcionality, jejichž syntax byl pozměněn nebo je funkcionalita reprezentována pod jiným názvem. PostgreSQL umožňuje spouštět vytvořené procedury v několika programovacích jazycích, např.: Pearl, Python nebo C. Může být využíván jak pro malé aplikace, fungující na jednom stroji, tak pro webové aplikace fungující s více uživateli najednou. [20]

Je podporován na hlavních operačních systémech (Linux, Windows, macOS). Od roku 2001 splňuje podmínky ACID.

2.2.3 Liquibase

Liquibase je open-source knihovna sloužící ke správě schématu databázového systému. Uchovává záznamy o provedených změnách v tabulce `databasechangelog`, která jsou tvořeny v závislosti na `changeSetech`. V `changeSetu` je možné upravovat tabulky nebo jejich sloupce, každá změna je uložena v databázi společně s autorem a datem provedené změny. Díky tomu je jednoduché, se kdykoli vrátit na předchozí verzi databáze. [21]

```
1     <changeSet id="1589624944143-157" author="Tomas Pospisil">
2     <modifyDataType tableName="bu_attachments" columnName="type"
          newDataType="varchar(100)"/>
3     </changeSet>
```

Zdrojový kód 2.7: Změna data typu sloupce v tabulce za pomoci liquibase

2.2.4 iText

iText je open-source knihovna využívající se k tvorbě a úpravě PDF dokumentů. Umožňuje vytvářet PDF dokumenty na základě dat z databáze nebo souboru XML. Samotná úprava dokumentu je v Javě řešena pomocí API, které se využívají pro grafickou úpravu dokumentu, změnu písma nebo rozpoložení stránek. [22]

Kapitola 3

Analýza a návrh

3.1 Představení firmy

Firma Prokop & Brož se zaměřuje na výrobu luxusních hodinek na míru. Jedná se o malou firmu založenou v roce 2012 hodinářem Martinem Brožem a obchodníkem Janem Prokopem. Jedinečnost jejich práce spočívá v možnosti zákazníka ovlivnit konečnou podobu hodinek, ale i sledovat jednotlivé fáze jejich výroby.

Hodinář Martin Brož je specialista nejen na hodinky, ale i na věžní hodiny. Jeho prvotním snem bylo stát se zlatníkem, ale nakonec se na přání rodiny rozhodl pro hodinářskou školu, na které se rozvinula jeho touha po tvorbě vlastních hodinek. Tento sen se mu splnil roku 2001, kdy začal vyrábět hodinky dle vlastního designu. Začínal opravami starožitných hodin, hodinek a výrobou jejich součástek. V roce 2002 si otevřel vlastní obchod a dílnu, kde poskytoval zákazníkům veškerý servis a začal vytvářet i vlastní součástky. [23]

Jan Prokop je vystudovaný bankéř a finančník, který začínal svou kariéru jako prodejce luxusního nábytku. Později se přesunul k poradenství v oblasti bytového designu a kancelářského návrhářství. V roce 1997 si založil firmu zaměřenou na průmyslový design, obchodní činnost a poradenské služby. Začal spolupracovat s hodinářskou firmou ELTON a jako její obchodní ředitel napomohl její přeměně. Jeho téměř dvacetileté zkušenosti ho dovedly k tvorbě speciálních hodinek vytvořených podle individuálních požadavků zákazníka. [24]

První setkání se uskutečnilo v roce 2011 na zahrádce restaurace U Pinkasů, kde oba zjistili, že sdílí společnou vášeň pro hodinky. Na konci roku 2011 dostala firma Creative Services s.r.o., v té době vlastněna Janem Prokopem, nabídku od Plzeňského Prazdroje na vytvoření designu pro exteriérové hodiny připomínající orloj. Orloj měl být vytvořen k oslavě 170. výročí pivovaru. Společnost Creative Services s.r.o. si přizvala na pomoc hodináře Martina Brože a skupinu jeho přátel. Tento společný projekt otestoval spolupráci mezi

Janem Prokopem a Martinem Brožem, která nakonec vedla k založení jejich společné firmy. [25]

Hlavním snahou firmy je, aby zákazník nebyl jen pasivním účastníkem, ale aby se aktivně podílel jako spoluautor na vzniku hodinek. Z toho důvodu zákazník dochází do firmy na konzultační schůzky, na kterých jsou s ním projednávány detaily a úpravy (materiály, rytiny, tvar apod.), aby finální výrobek splňoval představy a požadavky zákazníka. Doba tvorby hodinek může dosahovat i několika měsíců, jedná-li se o velmi detailní hodinky, nebo pokud dochází v průběhu výroby k častým změnám. Pro prvního držitele hodinek je součástí koupě doživotní servis. [26]

3.2 Pojmy

V následující části představím pojmy, která se budou vyskytovat v dalších částech bakalářské práce.

3.2.1 Specifikace hodinek

Celkový popis hodinek obsahující název, kategorii, cenu a skupiny komponent. Specifikace se dělí na 4 typy:

- **zakázková** - reprezentující realizovanou zakázku, obsahuje navíc informace o držiteli a individualizace na přání zákazníka
- **maketa** - reprezentuje výstavní kousek, který se dá také využít pro náhradní díly, pokud momentálně nejsou jiné na skladě
- **vzorová** - obsahuje předem předvyplněné skupiny komponent, je využívána jako základ pro nové specifikace
- **nerealizovaná** - byla vytvořena dle přání zákazníka, k její realizaci ale nedošlo

3.2.2 Skupiny komponent

Seskupení typů komponent (např. číselník, ručičky), ze kterých se skládají hodinky. Skupiny jsou předem definované v systému, a proto vždy obsahují stejné typy komponent. Což v praxi znamená, že skupina ručičky se bude vždy skládat z hodinové, minutové a sekundové ručičky.

3.2.3 Typy komponent

Jedná se o jakoukoliv obecnou část hodinek. Pokud bychom se podívali např. na typ komponenty „Pouzdro“, konkrétní komponentou by poté mohlo být „Pouzdro Reform“, které již má přidány všechny atributy s možnými přednastavenými hodnotami atributů.

3.2.4 Atributy

Slouží k detailnímu popisu komponenty, může se jednat např. o typ použitého materiálu, velikost nebo individualizaci komponenty.

3.2.5 Defaultní atributy

Systémově přednastavené atributy pro jednotlivé typy komponent, které obsahují buď předdefinovanou hodnotu nebo žádnou hodnotu nemají. Slouží pro rychlejší vytváření nových komponent a zachování stejné formy pro dané typy komponent.

3.3 Předchozí řešení

Předchozí řešení bylo vytvořeno Ing. Vojtěchem Polcarem v rámci diplomové práce. Jednalo se o velmi rozsáhlou aplikaci (viz. obrázek 3.1) ve formě informačního systému pro kompletní evidenci veškerých procesů spojených s tvorbou a prodejem hodinek. Aplikace evidovala jednotlivé komponenty, součástky, specifikace, evidenci zákazníků, ale i procesy spojené s výrobou, jakými jsou popisy jednotlivých kroků. Jednotlivé kroky byly ve formě úkolů. Úkoly spojené s výrobou byly rozděleny jednotlivým kontraktorům.

Informační systém byl rozdělen na tři části. První částí byla klientská aplikace, v jazyce JavaScript, přesněji za pomoci frameworku React. Komunikace mezi klientskou a serverovou částí bylo implementování ve formě REST.

Druhou částí byla část serverová, napsaná v jazyce Java ve frameworku Spring a poslední částí byla část databázová psaná v jazyce PostgreSQL. Pro komunikaci mezi serverem a databází sloužilo JDBC. JDBC je API, které definuje rozhraní pro přístup k relační databázi.

3.3.1 Vývojové prostředí

Pro tvorbu aplikace byl využíván verzovací systém Git ve formě webového repozitáře GitLab[1]. Aplikace byla rozdělena do dvou projektů, na část frontendu a backendu. V každém projektu byly vypsané *issues*, které popisovaly problémy k opravě nebo požadavky zadavatele. K jednotlivým *issues* byla vytvořena větev pro implementaci a po jejímž dokončení vytvořen „merge request“, kterým došlo ke sloučení do větve **development**.

Po sloučení dojde ke spuštění automatizačního nástroje Jenkins, který je nakonfigurován, aby po každé aktualizaci větve **development** kód z větve stáhl a pokusil se ho sestavit. Pokud sestavení aplikace proběhne v pořádku dojde k nasazení aplikace na server.

[1] `https://gitlab.com/`

Větev **development** byla nasazena na serveru od společnosti **DigitalOcean**[2] v docker kontejneru.

3.3.2 Doménový model

Doménový model 3.1 dělím do dvou skupin, kterým bych se dále detailněji věnoval. První skupinou jsou objednávky a druhou specifikace.

Entita **Orders** (zakázka) obsahovala základní informace o objednávce a byla spojena se specifikací. Objednávka mohla obsahovat přílohy, pro které byla připravena entita **Attachments** (přílohy). Dále byla spojena s entitami **Customers** a **Dealers**, které reprezentovaly zákazníka/y a obchodníka/y a byly zaštítěny abstraktní třídou **Person**. Na entitu obchodníka byla navázána množina entit: **Users** (uživatelé), **Roles** (role), **Privileges** (oprávnění) a **Employees** (zaměstnanci), sloužící k rozdělení přístupových práv do aplikace. Na zaměstnance a obchodníka je možnost navázat úkoly přes entitu **Person**.

Samotné úkoly (**Orders Workflow**) byly generovány při vytvoření zakázky pro každou komponentu a napojeny na **Orders**. Úkoly byly dále členěny pomocí entity **Orders Workflow Step** na jednotlivé kroky a rozděleny do dvou skupin v závislosti na autorovi vytvořeného kroku pomocí entit **Dealer Step** a **Supplier Step**. Vytvořené kroky byly následně přiděleny konkrétní osobě z entity **Person**.

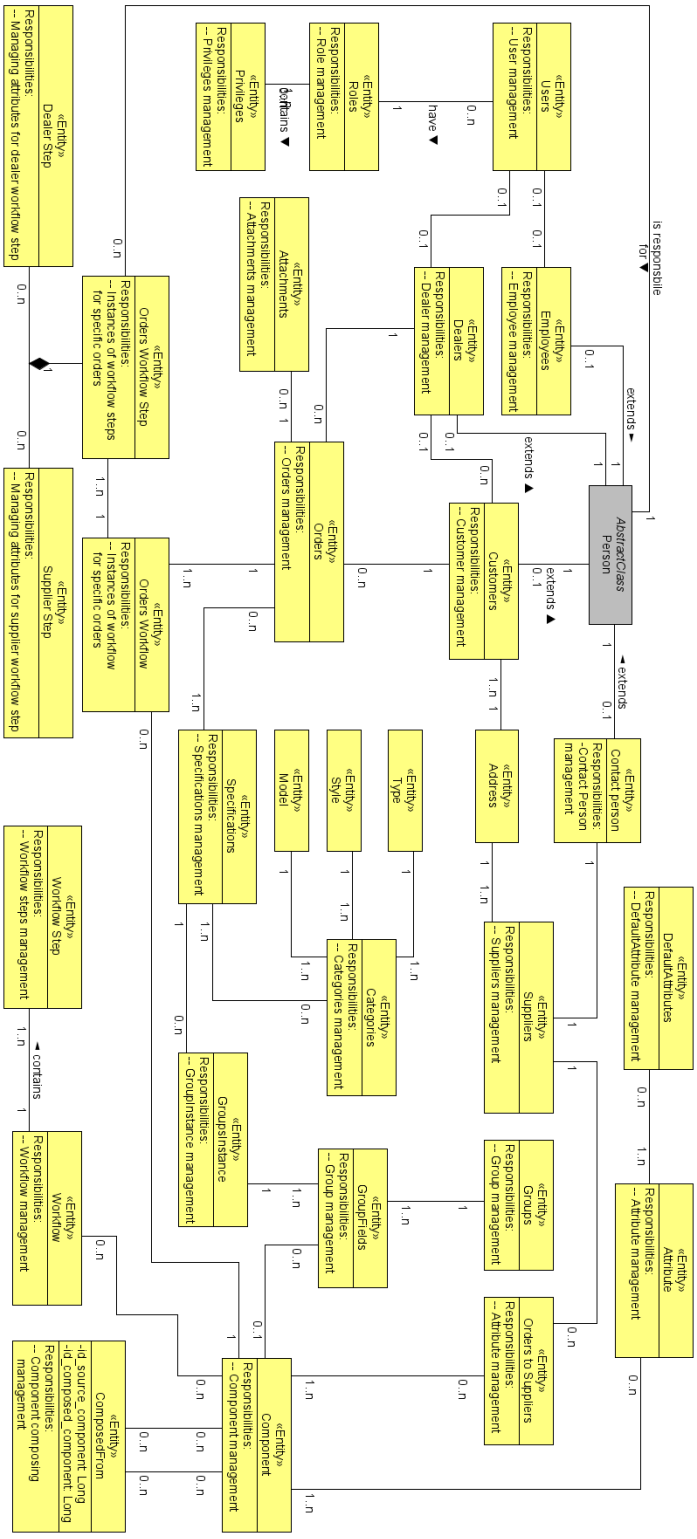
Zmiňovaná specifikace byla definována entitou **Specifications** (specifikace), které byla přidělena kategorie (**Categories**) ve formě typu, stylu a modelu (skrze entity **Type, Style** a **Model**). Součástky a komponenty byly vedeny společně pod entitou **Component**, která obsahovala společně znaky pro obě skupiny. Na komponenty byla navázána entita **ComposedFrom**, která zaznamenávala složení jednotlivých komponent ze součástek. Jestliže pro komponentu v tabulce neexistoval záznam, jednalo se o součástku, v opačném případě se jednalo o komponentu.

Komponenty a součástky jsou použity v entitě **GroupFields**, kde jsou využity jako základní stavební jednotky a v tomto seskupení reprezentují typ komponenty. V entitě **Groups** je uložen pouze přehled vytvořených vzorových skupin, ze kterých se vytváří instance **GroupsInstance**. Instance se vztahují již ke konkrétní specifikaci a obsahují konkrétní skupiny atributů.[2]

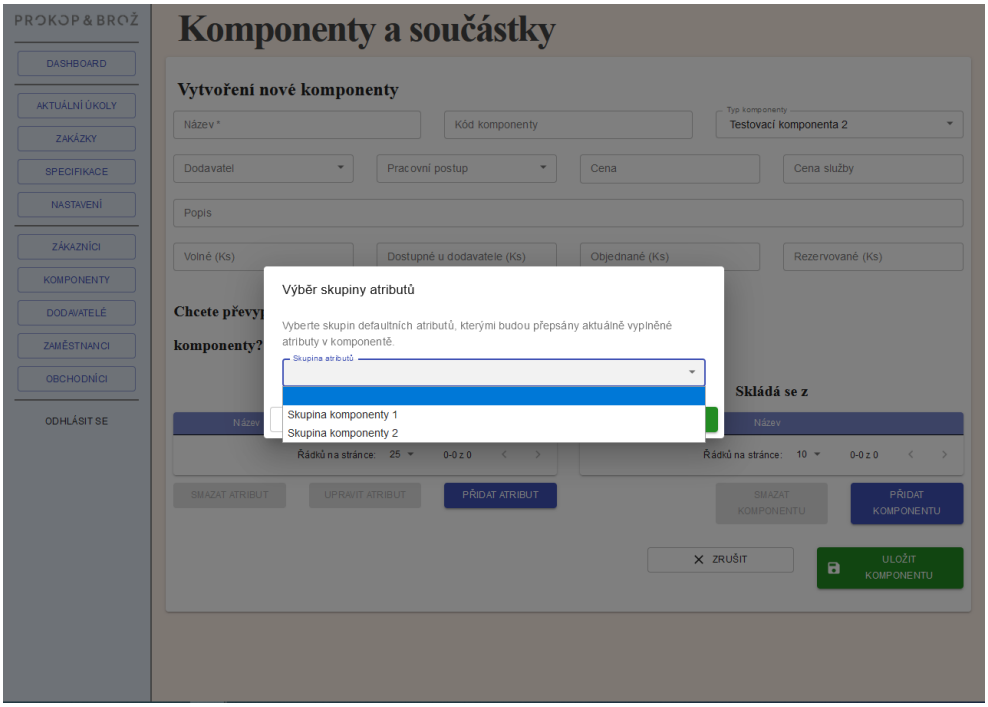
3.3.2.1 Komponenty a atributy

Uložení atributů a defaultních atributů 3.2.5 bylo rozděleno na straně databáze do čtyř tabulek. První byla `cf_attribute`, která obsahovala název a hodnotu všech atributů. Vyskytovaly se zde atributy defaultní a atributy z nich vzniklé. Do databáze byly navíc ukládány i atributy odvozené z atributů defaultních,

[2] `https://www.digitalocean.com/`



Obrázek 3.1: Doménový model počátečního stavu aplikace [2]



Obrázek 3.2: Výběr skupin atributů, při tvorbě nové komponenty

které nenabývaly žádné hodnoty. To by do budoucna mohlo znamenat zbytečné zaplnění databáze, které by vyústilo ve zpomalení systému.

Druhou tabulkou byla `cf_attribute`, která obsahovala vygenerovaná id reprezentující skupinu defaultních atributů, název skupiny a id typu komponenty, ke které se defaultní atributy vztahují.

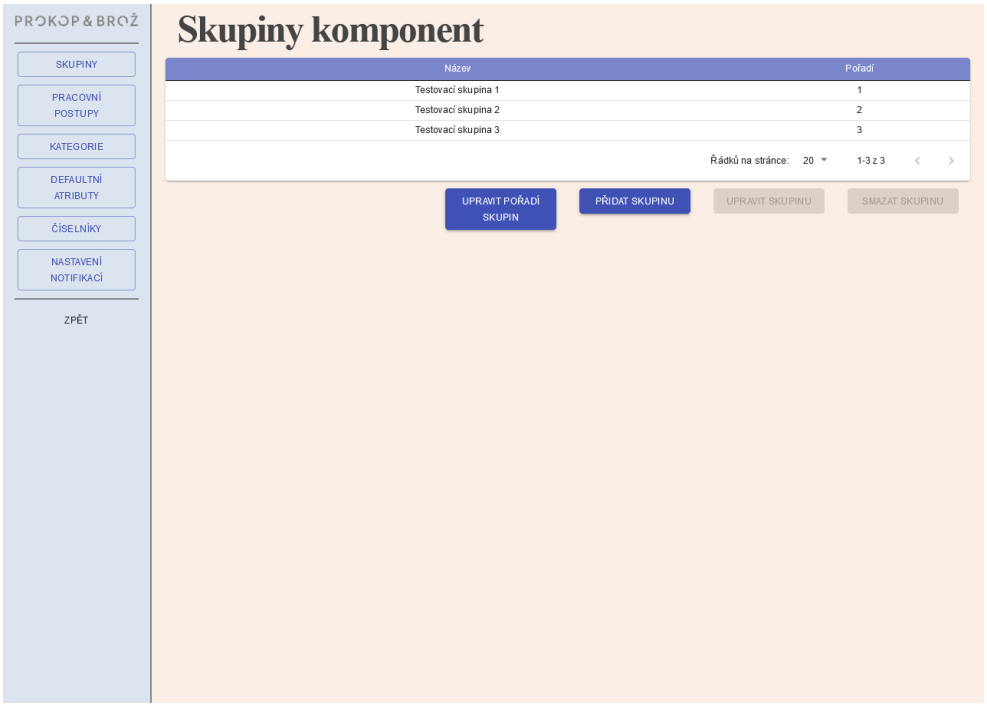
Třetí tabulka (`cf_attribute_in_default_attribute`) sloužila jako dekompozice vazby M...N mezi tabulkami defaultních atributů a atributů.

Čtvrtá tabulka (`cf_attribute_in_components`) sloužila jako dekompozice vazby M...N mezi tabulkami atributů a komponent.

Při vytváření nové komponenty byla v systému implementována funkce, pro usnadnění zadávání atributů uživateli, která přiřadí defaultní atributy. Systém ovšem zobrazil uživateli všechny skupiny defaultních atributů, bez ohledu na to k jakému typu komponenty byly vytvořeny (viz. obrázek 3.2). Při větším množství skupin by byl výběr nepřehledný. Přiřazení typu komponenty ke skupině defaultních atributů je patrné z obrázku A.1.

3.3.2.2 Skupiny a specifikace

K práci systému se skupinami sloužily v databázi tři tabulky. První byla `bu_specification`, která v sobě ukládala obecné informace o skupině, jako



Obrázek 3.3: Přehled vytvořených skupin

je id, název a pořadí, ve kterém byly skupiny vyobrazeny v přehledu skupin ve webové aplikaci. Toto pořadí se v aplikaci dalo upravovat viz. obrázek A.8.

Druhou tabulkou byla `bu_group_instance`, která reprezentovala skupiny použité ve specifikaci. Obsahovala id, název, id specifikace a pořadí, jehož změna nebyla implementována, a proto nabývala hodnot, ve kterých jsou skupiny seřazeny v přehledu skupin 3.3, což vedlo k duplikování hodnot v daném sloupci. Skupiny byly ve specifikaci zobrazeny v takovém pořadím, v jakém byly přidávány do specifikace pomocí modálu A.7.

Třetí tabulka (`bu_group_field`) obsahovala id, název pole, které se zobrazilo uvnitř skupiny ve specifikaci viz. obrázek A.4, id komponenty, id skupiny, id instance skupiny a id typu komponenty. Do tabulky byly ukládány dva typy záznamů:

- záznam obsahující id, název pole, id skupiny, ve které byl zobrazován na přehledu skupin a id typu komponenty. Jednalo se tedy o vzorový záznam, který byl používán jako šablona při tvorbě specifikací.

- záznam obsahující id, název pole, id komponenty, pokud byla přiřazena při tvorbě specifikace, id instance skupiny vyskytující se ve specifikaci ve formě záložky A.4 a id typu komponenty. Záznamy popisovaly reprezentaci skupin v reálných zakázkách.

Specifikace byla na straně databáze řešena tabulkou `bu_specification` obsahující id, název, cenu, id kategorie (kolekce, do které je specifikace zařazena) a typ specifikace 3.2.1. Na straně frontendu ve specifikaci docházelo často k chybám při mazání skupin. Jelikož po prvním smazání došlo k odznačení zvolené skupiny, ale uživateli bylo stále přístupné tlačítko „smazat skupinu“, tak následně po znovupoužití tlačítka se smazala náhodná skupina viz. obrázky A.5 a A.6.

3.3.2.3 Shrnutí

Aplikace byla velmi rozsáhlá a obsahovala řadu funkcionalit, z nichž některé nebyly dostatečně vyhovující. Na straně frontendu byly příliš často využívány modály (při přidání, úpravě i mazání), což by bylo pro uživatele nepraktické a zdlouhavé. Na straně backendu byl největší zpozorovaný problém ve formě možného přehlcení databáze, což by vedlo k následnému zpomalení aplikace.

3.4 Analýza požadavků

Cílem následující části je definovat rozšíření webové aplikace a popsat její funkčnost. Analýza požadavků je zásadní pro vývoj softwarového projektu, zaměřuje se na úkoly, které popisují změny nebo nové funkcionality projektu. Požadavky jsou rozděleny na funkční a nefunkční.

3.4.1 Funkční požadavky

Funkční požadavky jsou využívány při vývoji softwaru. Jejich součástí je popis funkcionality ve formě prvotního kroku, procesu a finálního stavu, ve kterém se má tvořený systém nacházet.

Je důležité, aby požadavky byly zadavatelem detailně specifikovány, aby v průběhu jejich implementace nedocházelo ke vzniku zavádějících kroků. Z nejasně specifikovaných požadavků mohou vzniknout funkcionality, které nebyly představou zadavatele, což vede k prodloužení implementace samotné a k vyšší ceně finálního produktu. [27]

3.4.1.1 FP1 - Defaultní atributy

Defaultní atributy 3.2.5 jsou základní jednotkou každého typu komponenty, proto je pro uživatele důležité mít možnost s atributy libovolně manipulovat a upravovat je.

- **FP1.1 - Přidání defaultního atributu** - přidání nového defaultního atributu nijak neovlivní již vytvořené komponenty. Takovým komponentám bude atribut nastaven s prázdnou hodnotou, i kdyby měl hodnotu předdefinovanou. Atribut se zobrazí nově vytvořeným komponentám nebo komponentám upravovaným.

- **FP1.2 - Odstranění defaultního atributu** - samotné odstranění pouze skryje atribut nově tvořeným komponentám, k tomuto dochází, aby byl stále zachován v komponentách již vytvořených, kterých byl součástí.
- **FP1.3 - Úprava defaultního atributu** - umožnění úprav existujících atributů, jedná se o název, anglický název a hodnotu atributu.
- **FP1.4 - Změna pořadí defaultních atributů** - povolení změny pořadí, ve kterém se atributy zobrazují v komponentách. Uživatel si tedy může nastavit na začátek atributy, které se vyplňují častěji a naopak na konec dá méně využívané atributy.

3.4.1.2 FP2 - Evidence skupin komponent

Evidence skupin bude obsahovat všechny vytvořené skupiny 3.2.2. Skupina obsahuje libovolný počet typů komponent, které existují na straně databáze. Typy komponent jsou zprvu prázdné a bez atributů. Komponenty a atributy nabývají konkrétních hodnot až při výběru komponenty uživatelem ve specifikaci 3.2.1.

- **FP2.1 - Vytvoření nové skupiny** - uživatel může vytvářet nové skupiny, do kterých následně přiřadí typy komponent, ze kterých se bude skupina vždy skládat.
- **FP2.2 - Odstranění existující skupiny** - uživatel může odstranit skupinu z výběru, ovšem na straně databáze zůstávají její kopie, pokud je použita v některých, již existujících specifikacích.
- **FP2.3 - Úprava existující skupiny** - umožnění úprav názvu a anglického názvu existující skupiny.
- **FP1.4 - Změna pořadí typů komponent** - umožnění změny pořadí typů komponent, pro usnadnění vyplňování dat uživatelem nebo pro systematické pořadí komponent, kterým je daná skupina popisována klientovi na schůzce.
- **FP1.5 - Přidání typu komponenty** - přidání existujícího typu komponenty na straně databáze do skupiny.
- **FP1.6 - Odstranění typu komponenty skupiny**
- **FP1.7 - Úprava typu komponenty skupiny**
- **FP1.8 - Změna pořadí skupin** - povolení změny pořadí typů komponent, ve kterém jsou prezentovány ve skupině.

3.4.1.3 FP3 - Evidence komponent

Evidence komponent bude obsahovat všechny vytvořené komponenty. S viditelným názvem, kódem, popisem a skladovým záznamem o komponentě.

- **FP3.1 - Vytvoření nové komponenty** - uživatel může vytvořit novou komponentu, kde se mu podle vybraného typu komponenty zobrazí přiřazené defaultní atributy.
- **FP3.2 - Odstranění existující komponenty**
- **FP3.3 - Úprava existující komponenty** - umožnění úprav obecných informacích o komponentě a jejich atributů.
- **FP3.4 - Zobrazení detailu komponenty** - možnost zobrazení kompletního detailu komponenty včetně seznamu specifikací, ve kterých je komponenta použita.
- **FP3.5 - Vytvoření kopie komponenty** - umožnění zkopírovaní obecných dat a atributů komponenty do komponenty nové. Nově vzniklá komponenta nesmí mít stejné jméno, jelikož název musí být vždy unikátní.
- **FP3.6 - Označení komponenty za vzorovou** - umožnění označení komponenty za vzorovou při vytváření, úpravě a kopírování.
- **FP3.7 - Přiřazení typu komponenty** - umožnění zvolit libovolný existující typ komponenty na straně databáze, na jehož základě dochází ke stažení příslušných atributů.
- **FP3.8 - Vyplnění atributů** - umožnění přiřazení hodnot atributů.
- **FP3.9 - Přiřazení součástek do komponenty**
- **FP3.10 - Filtrování komponent** - umožnění filtrování komponent na základě názvu, kódu komponenty, čísla zakázky, typu komponenty, specifikace a zda je komponenta označena jako vzorová.

3.4.1.4 FP4 - Evidence specifikací

Evidence specifikací bude obsahovat soupis všech vytvořených specifikací 3.2.1. V soupisu se budou vyskytovat obecné informace o specifikaci, jako je název, typ, cena atp., soupis všech využitých skupin s konkrétními typy komponent a atributy.

- **FP4.1 - Vytvoření nové specifikace** - uživatel může vytvořit novou kompletní specifikaci, reprezentující hodinky, skládající se ze skupin, typů komponent, atributů a obecných informacích.

- **FP4.2 - Odstranění existující specifikace** - uživatel může odstranit specifikace ze systému, pokud se nevztahuje k existující zakázce.
- **FP4.3 - Úprava existující specifikace** - umožnění úprav skupin, typů komponent, jednotlivých atributů a obecných informací ve specifikaci.
- **FP4.4 - Zobrazení detailu specifikace** - zobrazení kompletního detailu specifikace, se všemi skupinami, typy komponent, atributů a jeho příloh.
- **FP4.5 - Vytvoření kopie specifikace** - umožnění zkopírování skupiny včetně všech jejích vyplněných dat.
- **FP4.6 - Přiřazení typu specifikace** - umožnění uživateli přiřadit typ specifikace viz. 3.2.1 při vytváření,úpravě a kopírování.
- **FP4.7 - Přiřazení skupin ke specifikaci**
- **FP4.8 - Odstranění skupin ze specifikace**
- **FP4.9 - Filtrování specifikací** - umožnění filtrování specifikací na základě názvu, čísla zakázky, zákazníka, obchodníka a typu specifikace.

3.4.1.5 FP5 - Soubory

Systém musí umožňovat přidávání libovolných typů souborů k zakázce a specifikace. Může se jednat o grafické návrhy, které vznikaly v průběhu schůzek s klientem nebo o faktury spojené s výrobou atp.

- **FP5.1 - Nahrání souboru ke specifikaci**
- **FP5.2 - Odstranění souboru ze specifikace**
- **FP5.3 - Nahrání souboru k zakázce**
- **FP5.4 - Odstranění souboru ze zakázky**

3.4.1.6 FP6 - Generování reportů

Systém bude schopný vytvořit tisknutelný report pro specifikace, kterou bude možné tisknout ve dvou verzích (pro zákazníka a pro kontraktory bez uvedení ceny a poplatků spojených s prodejem). Musí obsahovat všechny vybrané skupiny, typy komponent a atributy z důvodu detailních schůzek v průběhu tvorby hodinek.

- **FP6.1 - Vygenerování reportu specifikace se všemi detaily**
- **FP6.2 - Vygenerování reportu specifikace bez ceny**

3.4.2 Nefunkční požadavky

Nefunkční požadavky slouží k popisu vlastností a omezení potřebných k tvorbě kvalitní a stabilní aplikace či systému. Nefunkční požadavky byly přejaty z diplomové práce Ing. Vojtěcha Polcara. [2]

- **NP1 - Bezpečnost** - systém bude využívat autentizaci a autorizaci, aby nedošlo k úniku citlivých dat.
- **NP2 - Webová aplikace** - systém bude implementovaný ve formě webové aplikace.
- **NP3 - Responzivita** - aplikace bude optimalizována pro zobrazení jak na počítači, tak na mobilním zařízení.
- **NP4 - Legislativa** - systém bude nakládat s osobními údaji uživatelů v souladu se zákonnými podmínkami.
- **NP5 - Zálohování** - systém bude provádět pravidelné zálohování dat.
- **NP6 - Rozšiřitelnost** - systém bude možno rozšiřovat o další funkcionality.

3.5 Návrh

Následující kapitola je zaměřena na popis návrhu aplikace. První část obsahuje detailní popis nově vytvořeného doménového modelu, který reprezentuje nové rozestavení entit pro tvorbu atributů, defaultních atributů a komponent.

V dalších částech je zahrnuto předělání tvorby nové specifikace, včetně detailu specifikace a návrh tisknutelného reportu.

3.5.1 Defaultní atributy a komponenty

Prvním cílem návrhu bylo vytvořit efektivnější databázovou strukturu, ve které by každá komponenta měla jasně definovaný typ, dle kterého by k ní byly přiřazeny jednotlivé atributy. Jednalo by se o atributy defaultní, které by vždy definovaly jeden určitý typ komponent. K tomuto by byla použita entita **Default attributes**, která by obsahovala id, název, anglický název, id typu komponenty, pořadí ve kterém budou atributy uspořádány ve skupině a možnou přednastavenou hodnotu.

Defaultní atributy budou spojeny vazbou 1...1 s číselníkem **ComponentType**, který bude sloužit k rozdělení jednotlivých komponent na základě jejich typu (např. typ komponenty „Pouzdro“ bude obsahovat komponenty jako jsou Pozdro Reform či Poudro Titan). Tato vazba také bude definovat, jaké atributy má daný typ komponenty vždy obsahovat.

Každé komponentě, reprezentované entitou **Component**, budou na základě vybraného typu komponenty dotaženy defaultní atributy, které budou prázdné nebo s přednastavenou hodnotou. Tím dojde ke sjednocení atributů napříč komponentami jednoho typu. Pokud dojde k vyplnění hodnoty u atributu, bude tento atribut vytvořen v entitě **Attribute**, která bude obsahovat jeho hodnotu a id defaultního atributu ze kterého vychází. Tento vztah bude reprezentován vazbou 0...N mezi entitami **Default Attributes** a **Attributes**.

Vytvářením záznamů pouze vyplněných atributů dojde k úspoře využitého místa na straně databáze.

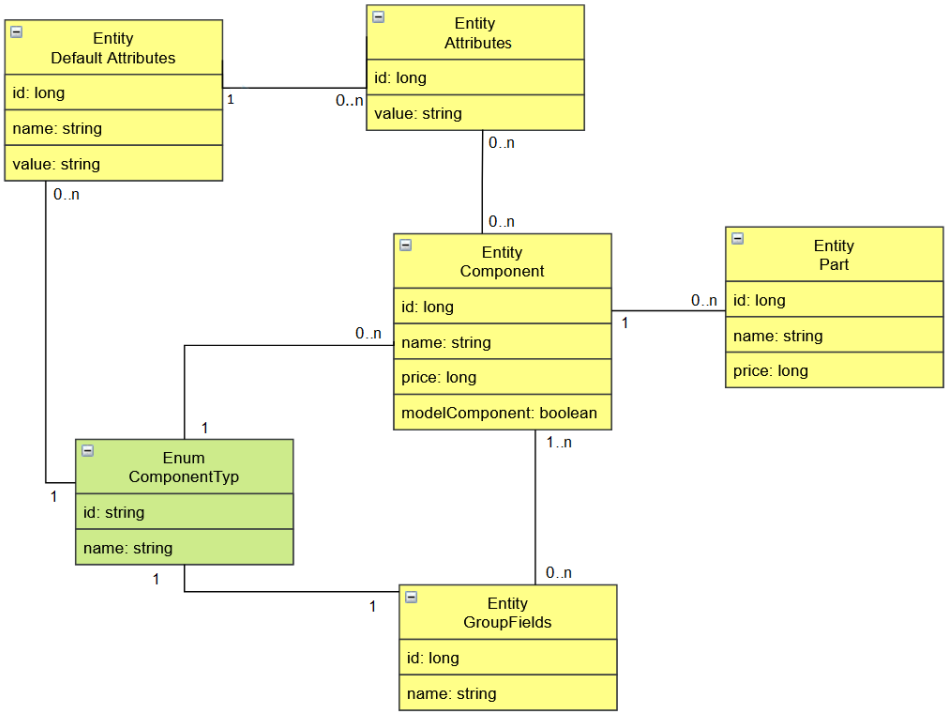
Při vytváření nového defaultního atributu dojde k jeho přidání do všech existujících komponent daného typu. Tento atribut bude viditelný na detailu komponent s prázdnou hodnotou, i kdyby měl hodnotu přednastavenou. Přednastavená hodnota se bude zobrazovat jen při vytváření nové komponenty, až po uložení bude dané komponentě vytvořen záznam v entitě **Attributes** s touto hodnotou. Proto komponenty vytvořené před přidáním nového defaultního atributu do systému budou mít tuto hodnotu prázdnou.

Odstraněním defaultního atributu z databáze přestane být atribut zobrazován při vytváření nové komponenty. Ovšem v tabulce **Attributes** budou záznamy o něm stále existovat a u starších komponent tak bude atribut stále zobrazován, aby nedošlo ke ztrátě informací o komponentách.

Při zobrazování komponenty ve webové aplikaci dochází nejdříve k porovnání dat mezi defaulními atributy typu komponenty (tabulka **Default Attributes**) a atributy přiřazenými ke komponentě (tabulka **Attributes**). Ke každému defaultnímu atributu je mapována hodnota z dotažených atributů, jestliže záznam s id defaultního atributu v tabulce atributu neexistuje, je zobrazen s prázdnou hodnotou. Na posledním místě jsou vyobrazeny atributy z tabulky **Attributes**, které nebyly namapovány na žádný defaultní atribut.

V další části návrhu došlo k oddělení součástek od komponent. Vznikla nová entita **Parts**, která reprezentuje malé součástky (matičky, šroubky apod.), ze kterých jsou sestavovány komponenty. Entita **Parts** spojena s entitou **Component** vazbou 0...N.

Napojení komponent na specifikaci bylo zachováno, došlo pouze k drobným úpravám entit. První entitou byla **GroupInstance**, která reprezentovala instanci skupiny přiřazené ke konkrétní specifikaci (jednalo se např. o skupinu Pouzdro, skupinu Sklo nebo skupinu Strojek). K původnímu návrhu (viz. 3.1) byl přidán atribut anglického názvu skupiny pro tisk. Na entitu **GroupInstance** zůstala napojena entita **GroupField**, která byla rozšířena o atribut pořadí, ve kterém daná instance byla zobrazena ve skupině a `boolean` označující zda má být instance obsažena v tisknutelném reportu.



Obrázek 3.4: Návrh doménového diagramu pro defaultní atributy

3.5.2 Specifikace

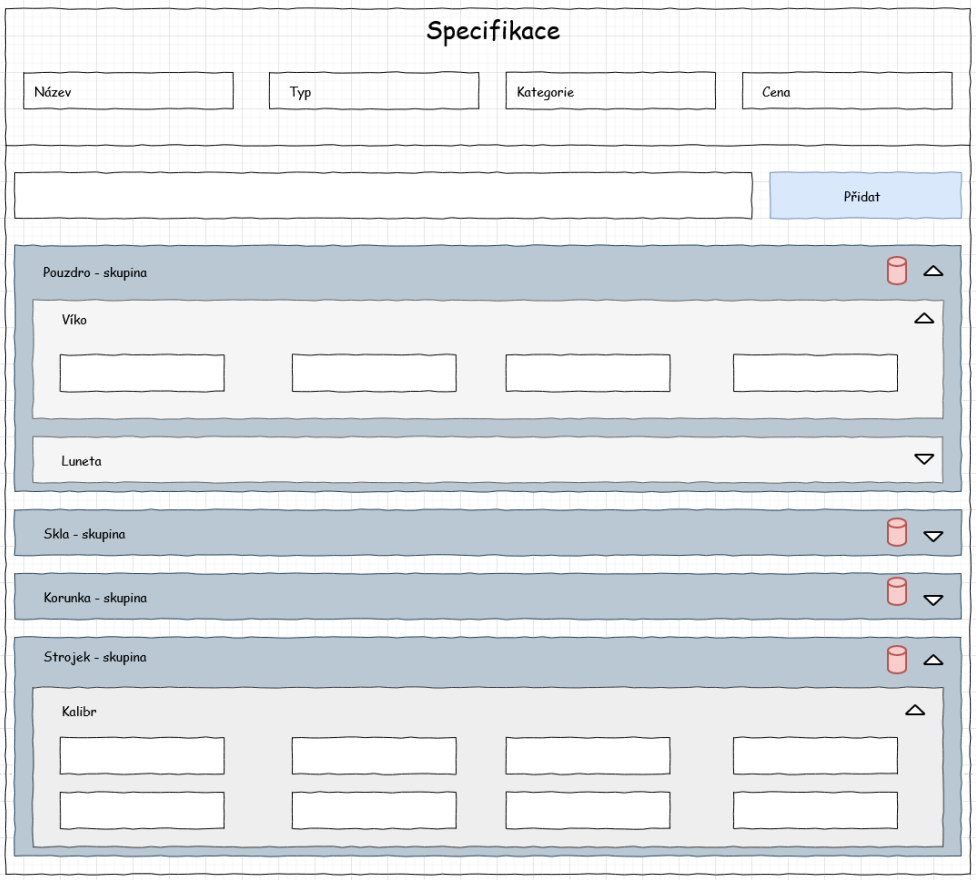
Ve specifikaci často docházelo k mazání špatné skupiny komponent, z důvodu využití předem vytvořené komponenty z Material UI[3], která vykreslovala skupiny ve formě záložek (viz. obrázek A.4).

V návrhu bylo zachováno rozdělení stránky do dvou částí. První obsahuje základní informace o specifikaci jako jsou: název, typ komponenty, kategorie a cena.

Druhá část obsahuje seřazené skupiny, dle pořadí z databáze, zobrazené ve sloupci pod sebou. Každé skupině byla vytvořena roztažitelná komponenta, které v sobě měla další vnořené roztažitelné komponenty reprezentující jednotlivé typy komponent. Typy komponent v sobě obsahují textová pole reprezentující atributy.

Mazání bylo navrženo formou symbolu popelnice na pravé straně roztažitelné komponenty, který po kliknutí vykreslí modal pro potvrzení akce. Takto se zamezí mazání náhodné skupiny. Nevyužité skupiny budou zobrazeny v selectu pod obecnými informacemi pro možnost přidávání, jelikož se zobrazí pouze skupiny zatím nevykreslené, nedojde k vytvoření duplikací.

[3] `https://material-ui.com/`



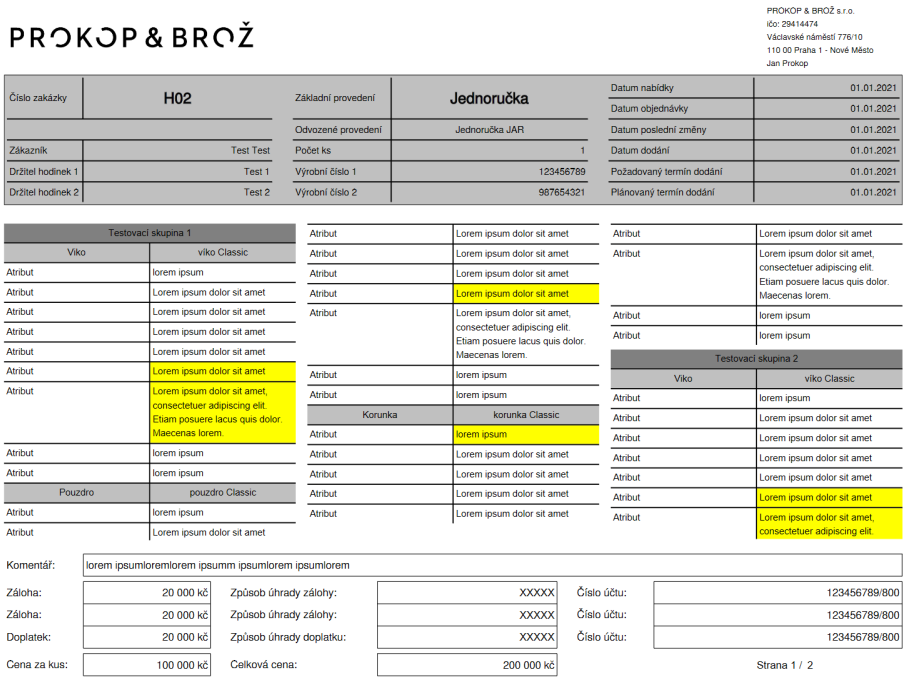
Obrázek 3.5: Návrh detailu specifikace

Uživatel si bude moci rozbalovat a sbalovat jednotlivé skupiny či komponenty dle potřeby, což by mělo výsledný produkt udělat uživatelsky přívětivějším.

3.5.3 Tisk

Tisknutelný report specifikace, který bude sloužit jako soupis důležitých částí hodinek, včetně obecných informacích ohledně samotného prodeje, bude na schůzce předán klientovi. Návrh dokumentu byl rozdělen do tří částí.

První částí dokumentu byla hlavička, která obsahovala logo firmy, včetně adresy sídla firmy, kontaktní osoby a tabulku s obecnými informacemi o zakázce. Tabulka samotná byla dále rozdělena do tří sloupců, v prvním bylo znázorněno číslo zakázky začínající písmenem, které označuje rok výroby a následně číslem reprezentujícím pořadové číslo zakázky daného roku, jménem zákazníka a výpisem držitelů hodinek, jedná-li se o model řadové výroby nebo je specifikace tvořena na více kusů hodinek. Uvedení držitelů hodinek je pro



PROKOP & BROŽ

PROKOP & BROŽ s.r.o.
IČo: 29414474
Václavské náměstí 776/10
110 00 Praha 1 - Nové Město
Jan Prokop

Číslo zakázky	H02	Základní provedení	Jednoručka	Datum nabídky	01.01.2021
		Odvozené provedení	Jednoručka JAR	Datum objednávky	01.01.2021
Zákaznik	Test Test	Počet ks	1	Datum poslední změny	01.01.2021
Držitel hodinek 1	Test 1	Výrobní číslo 1	123456789	Datum dodání	01.01.2021
Držitel hodinek 2	Test 2	Výrobní číslo 2	987654321	Požadovaný termín dodání	01.01.2021
				Plánovaný termín dodání	01.01.2021

Testovací skupina 1	
Víko	víko Classic
Atribut	lorem ipsum
Atribut	Lorem ipsum dolor sit amet
Atribut	Lorem ipsum dolor sit amet
Atribut	Lorem ipsum dolor sit amet
Atribut	Lorem ipsum dolor sit amet
Atribut	Lorem ipsum dolor sit amet
Atribut	Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam posuere lacus quis dolor. Maecenas lorem.
Atribut	lorem ipsum
Atribut	lorem ipsum
Pouzdro	pouzdro Classic
Atribut	lorem ipsum
Atribut	Lorem ipsum dolor sit amet

Atribut	Lorem ipsum dolor sit amet
Atribut	Lorem ipsum dolor sit amet
Atribut	Lorem ipsum dolor sit amet
Atribut	Lorem ipsum dolor sit amet
Atribut	Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam posuere lacus quis dolor. Maecenas lorem.
Atribut	lorem ipsum
Atribut	lorem ipsum
Korunka	korunka Classic
Atribut	lorem ipsum
Atribut	Lorem ipsum dolor sit amet
Atribut	Lorem ipsum dolor sit amet
Atribut	Lorem ipsum dolor sit amet
Atribut	Lorem ipsum dolor sit amet

Atribut	Lorem ipsum dolor sit amet
Atribut	Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam posuere lacus quis dolor. Maecenas lorem.
Atribut	lorem ipsum
Atribut	lorem ipsum

Testovací skupina 2	
Víko	víko Classic
Atribut	lorem ipsum
Atribut	Lorem ipsum dolor sit amet
Atribut	Lorem ipsum dolor sit amet
Atribut	Lorem ipsum dolor sit amet
Atribut	Lorem ipsum dolor sit amet
Atribut	Lorem ipsum dolor sit amet
Atribut	Lorem ipsum dolor sit amet, consectetuer adipiscing elit.

Komentář:	lorem ipsumloremlorem ipsumm ipsumlorem ipsumlorem				
Záloha:	20 000 kč	Způsob úhrady zálohy:	XXXXX	Číslo účtu:	123456789/800
Záloha:	20 000 kč	Způsob úhrady zálohy:	XXXXX	Číslo účtu:	123456789/800
Doplatek:	20 000 kč	Způsob úhrady doplatku:	XXXXX	Číslo účtu:	123456789/800
Cena za kus:	100 000 kč	Celková cena:	200 000 kč		

Strana 1 / 2

Obrázek 3.6: Návrh tisknutelného reportu

firmu důležité z důvodu kompletního servisu pro prvního držitele. Druhý sloupec obsahuje informace o modelu, počet kusů a výrobní číslo hodinek. Poslední sloupec obsahuje soupis dat, která zaznamenávají časovou osu od data nabídky po datum dodání.

Druhá část je pro klienta nejdůležitější, skládá se ze soupisu veškerých skupin, komponent a atributů, které jsou v systému označeny pro tisk. Právě tato část je s klientem nejvíce konzultována, jestliže si klient nechává tvořit hodinky ze vzorových komponent, je změna atributů (např. individualizace víka nebo pouzdra) vyobrazena v listu žlutou barvou. Výška této části se dynamicky mění v závislosti na počtu držitelů hodinek, který ovlivňuje výšku hlavičky. V základním stavu (pro 1 až 2 držitele) se do této sekce vejde 46 řádků záznamů.

Poslední část obsahuje komentář a údaje o platbách spojených s danou specifikací.

Kapitola 4

Implementace

V rámci bakalářské práce došlo k implementaci funkcionalit na základě funkčních požadavků viz. kapitola 3.4.1.

Byla přidána knihovna **React Hook Form**, pro tvorbu formulářů, díky které byl nově vzniklý kód kratší a přehlednější oproti předchozí knihovně **React Redux**. K manipulaci s daty byla využita knihovna **React Query**, kvůli zmenšení množství kódu potřebného pro komunikaci s databází a možnosti tvorby vlastních hooků. Z aplikace také bylo odstraněno větší množství modálů, které byly zachovány pouze pro potvrzení odstranění.

4.1 Defaultní atributy

Při tvorbě uživatelského rozhraní pro defaultní atributy se vycházelo z požadavků zadavatele na co nejjednodušší a přehlednou tvorbu přidávání, úpravy a odstranění defaultních atributů jednotlivých typů komponent.

Na straně defaultních atributů ve webové aplikaci jsou zobrazeny v tabulce všechny typy komponent, dotaženy pomocí vlastního hooku. Implementováno bylo filtrování na straně frontendu, které při každém renderu porovná hodnotu v input fieldu se všemi záznamy typů komponent. Na základě vyhledaného výrazu dojde k vytvoření objektu se záznamy odpovídajícími vyhledávání, který je následně předán jako nová data tabulky. Jelikož záznamů typů komponent nebudou více než desítky, aplikace nebude nijak zpomalována. Po kliknutí na libovolný řádek tabulky, dojde k přesměrování na úpravu defaultních atributů vybraného typu komponenty.

Před zobrazeních defaultních atributů vybraného typu komponent jsou nejdříve dotaženy z backendu, po úspěšném dotažení je ke každému záznamu přidán `boolean readOnly` s hodnotou `false`, který rozhoduje, zda daný záznam bude v módu úpravy nebo ho nebude možné měnit. Ke každému záznamu je dynamicky vytvořen prostřednictvím **React Hook Form** 2.1.6 samostatný formulář, obsahující:

- input field pro název atributu
- input field pro anglický název atributu
- input field pro hodnotu atributu
- checkbox označující, zda má být atribut zahrnut v tisknutelném reportu
- tlačítko tužky/zaškrtnutí sloužící k přepínání mezi úpravou a náhledem
- tlačítko popelnice sloužící k odstranění atributu z typu komponenty

Navíc je vygenerován prázdný řádek, který slouží k přidávání nových atributů. Ke generování dochází za pomoci `useEffect`u, který je zavolán na změnu proměnné `data` a `newAttribute`. Hook porovná hodnotu v proměnné `newAttribute` s velikostí pole obsahujícího jednotlivé atributy. Pokud jsou hodnoty stejné, dojde k přidání nového prázdného atributu.

Symbol zaškrtnutí zavolá funkci `onSubmit` vybraného formuláře, která v sobě volá funkci `closeEditAttribute`, které jsou předány formulářová data `attributeOrder`, `name`, `enName`, `value`, `includeInExport`.
Ve funkci `closeEditAttribute` nejprve dojde k nalezení indexu, na kterém se nachází měněný objekt za pomoci předdefinovaných funkcí `map` a `indexOf`, které projdou všechny objekty reprezentující atributy a porovnají pořadí s `attributeOrder`. Po úspěšném nalezení atributu jsou data všech atributů překopírována do nového pole a upravovanému atributu jsou přepsány proměnné dle parametrů funkce. Pokud index měněného atributu odpovídá hodnotě uložené v proměnné `newAttribute`, je nastavena nová hodnota `newAtrribute` odpovídající velikosti nově vytvořeného pole se všemi atributy. Výsledné pole objektů je uloženo do proměnné `data`.

```
1    const closeEditAttribute = (attributeOrder: number,
2    name: string, enName: string, value: string,
3    includeInExport: boolean) => {
4      const editIndex = data
5      .map(function (item) {
6        return item.order;
7      })
8      .indexOf(attributeOrder);
9
10     const tmpData = [...data];
11     tmpData[editIndex].readOnly = !data[editIndex].readOnly;
12
13     if (!isNil(newAttribute) && editIndex === newAttribute) {
14       setNewAttribute(tmpData.length);
15     }
16
17     tmpData[editIndex].name = name;
18     tmpData[editIndex].enName = enName;
19     tmpData[editIndex].includeInExport = includeInExport;
20     tmpData[editIndex].value = isEmpty(value) ? null : value;
```

```
21
22        setData(tmpData);
23    };
```

Zdrojový kód 4.1: Funkce ukončení úprav atributu

Funkce pro úpravu `startEditAttribute`, skrývající se pod tlačítkem tužky, je implementována obdobně jako funkce `closeEditAttribute`. Dochází ke změně dat atributu pouze u proměnné `readOnly`, která je nastavena na `true`.

Tlačítko popelnice odstraní vybraný atribut z dat v proměnné `data`, odstraněním se nemění data na backendu. Samotné odstranění je potřeba potvrdit v modalu, který se zobrazí po kliknutí na symbol popelnice, aby nedocházelo k odstranění dat kvůli nepozornosti. Po potvrzení smazání je volána funkce `deleteAttribute`, která odstraní, na základě parametru `attributeOrder`, zvolený atribut z objektu a přepočítá následujícím atributům pořadí (dojde ke zmenšení hodnoty o jedna). Ke změně dat na backendu dojde až ve chvíli, kdy je celý formulář uložen a odeslán na server.

Jednotlivé záznamy jsou generovány v komponentě `SortableElement` a zapouzdřeny komponentou `SortableContainer` z knihovny **react-sortable-hoc**[4]. Do `SortableContainer`u je navíc přidán `useDragHandle`, který umožní přesouvat vytvořený záznam pouze při uchopení ikony `DragHandle` a funkce `onSortEnd` se dvěma vstupními parametry: `oldIndex` reprezentující pozici, na které se nachází posunutý záznam v poli a `newIndex` udávající pozici na kterou byl záznam přesunut.

4.2 Komponenty

Přehled komponent byl implementován formou tabulky obdobně jako defaultní atributy. Při kliknutí na záznam v tabulce nedochází k přesměrování na jinou stránku, ale dojde k označení řádku a zobrazení detailu pod tabulkou. Kliknutím je spuštěn hook, který ověří zda jsou dotažena aktuální data komponent a následně pro vybraný řádek najde objekt komponenty. Dojde k nastavení proměnné `selected`, která barevně označí vybraný řádek v tabulce a do proměnné `selectedComponent` je vložen celý objekt komponenty, který je později využit při vykreslovaní detailu nebo modalu pro smazání.

```
1     /*
2     objekt vybraneho atributu s pocatecni hodnout nastavenou
3     na undefined, jinak nabyva hodnotu totoznou s interface
4     IComponentGet
5     */
6     const [selectedComponent, setSelectedComponent] =
7     useState<IComponentGet | undefined>(undefined);
8     /*
9     ciselna hodnota oznacujici zvoleny radek tabulky
10    s pocatecni hodnotou -1
```

[4] https://github.com/clauderic/react-sortable-hoc

```
11      */
12      const [selected, setSelected] = useState<number>(-1);
13
14      useEffect(() => {
15        if (selected !== -1 && components.data) {
16          const selectedComponent = components?.data?.find((component)
                => component.id === selected);
17          setSelectedComponent(selectedComponent);
18        }
19        /*
20        efekt je spusten pokud dojde ke zmene dotazenych dat nebo
21        je vybran zaznam v tabulce
22        */
23      }, [components, components.data, selected]);
```

Zdrojový kód 4.2: useEffect nastavující vybranou komponentu

K vykreslení detailu komponenty dojde po jejím vybrání v tabulce. Pro detail je implementována funkční komponenta `ComponentDetail`, která má povinný vstup ve formě objektu komponenty. Pro obecné informace o komponentě jsou předpřipravena textová pole „jen pro čtení“ a pro atributy jsou generována textová pole dynamicky, na základě počtu objektů v poli `attributes`. Ve spodní části detailu se dále vyskytuje tabulka s výpisem všech specifikací, ve kterých je komponenta využita.

```
1       export interface IComponentGet {
2         id: number;
3         name: string;
4         enName?: string;
5         code: string;
6         description?: string;
7         componentType?: IComponentType;
8         attributes?: IAttribute[];
9         sample: boolean;
10        derivedComponent?: { id: number; name: string };
11        componentParts?: IComponentPart[];
12        store?: IStore;
13        specifications?: IBasicSpecification[];
14      }
```

Zdrojový kód 4.3: Interface komponenty dotažené z backendu

Pro vytváření nové komponenty byla vytvořena funkční komponenta `AddCont`, která si nejdříve dotáhne data komponent, data typů komponent a data součástek za pomoci vlastních hooků `useComponents`, `useComponentTypes` a `useComponentParts`. Následně dochází k úpravě dotažených dat do formátu pro select fieldy, kterým je dvojice id a název.

Prázdný formulář obsahuje:

- input field pro název komponenty
- input field pro anglický název komponenty

- input field pro kód komponenty
- select field typů komponent
- checkbox označující zda se jedná o vzorovou komponentu
- select field odvozované komponenty
- input field pro popis
- input fieldy atributů
- input fieldy součástek

Při vybrání typu komponenty v select fieldu, dojde ke změně proměnné `selectedComponentTypeId`, která je vytvořena za pomoci funkce `watch`, sloužící ke sledování vybraného prvku aplikace na základě přiděleného jména, která je dostupná z `formMethods` z knihovny React Hook Form. V tomto případě se jedná o select field obsahující všechny typy komponent. Při vybrání typu komponenty dojde ke změně hodnoty `selectedComponentTypeId`, následně dochází ke spuštění `useEffectu`, který změní stav select fieldu na „jen pro čtení“ a proměnné `componentTypeId` přiřadí id vybraného typu komponenty. Dochází k vyfiltrování dat komponent na základě shodujícího se id typu komponenty, která jsou zobrazena v select fieldu pro odvozovanou komponentu.

Změna hodnoty `componentTypeId` vyvolá hook `useDefaultAttributesForComponent(componentTypeId)`, který na základě parametru id typu komponenty dotáhne z databáze defaultní atributy, které jsou následně zobrazeny ve formuláři.

Poslední částí formuláře jsou součástky reprezentované pomocí hooku `useFieldArray`, který se využívá při práci s dynamicky generovanými záznamy. Hook je definovaný pomocí rozhraní záznamu (`name` a `keyName`) a metodou `control`, sloužící k registraci vygenerovaného záznamu do formuláře.

4.3 Specifikace

Prvotně byl formulář tvorby/úpravy/kopírování specifikace implementován pomocí knihovny **React-Hook-Form** 2.1.6. Formulář obsahoval obecné informace o skupiny ve formě input a select fieldů a jednotlivé skupiny komponent jako roztažitelné komponenty. V každé skupině komponent, byly roztažitelné komponenty reprezentující typy komponent, ve kterých se po vybrání komponenty vygenerovaly atributy.

Při vytváření nové specifikace jsou vždy na začátku vypsány všechny skupiny, aby uživatel nemusel skupiny přidávat manuálně po jedné, jelikož ve specifikaci je většinou využívána většina skupin. Skupinu je možná smazat kliknutím na symbol popelnice. Po smazaní skupiny ze specifikace je přesunuta do select fieldu, ze kterého je možné smazané skupiny opět přidat. Skupina se

tedy vždy nachází buď ve specifikaci nebo v select fieldu pro přidání skupiny. Díky tomuto nastavení nemůže dojít k duplikaci skupin ve specifikaci.

Jednotlivé skupiny jsou vyobrazeny v pořadí, které je nastaveno na webové stránce skupin a uvnitř obsahují přiřazené typy komponent. Konkrétní komponentu je možné vybrat v select fieldu uvnitř typu komponenty, po jejíž vybrání dojde k zobrazení atributů. Zmáčknutím symbolu popelnice u typů komponent dojde k vymazaní zvolené komponenty, ale roztažitelná komponenta typu komponenty stále ve skupině zůstává.

Vytvořený formulář byl pro knihovnu moc rozsáhlý a samotné jeho vykreslení trvalo nepřiměřené množství času. Při výběru komponenty docházelo k počtu renderů v rozmezí mezi 200 a 600 (závislost na počtu vykreslovaných skupin), což způsobovalo sekavé načítání a data byla zobrazena až po několika sekundách. Prvotní implementace byla v praxi nepoužitelná, proto byl stejný formulář finálně vytvořen za pomoci knihovny **Redux Form** 2.1.7, čímž došlo ke snížení počtu renderů.

4.3.1 Roztažitelná komponenta

V rámci implementace specifikace došlo k vytvoření univerzální, znovupoužitelné roztažitelné funkční komponenty `ExpandableComponent` za pomoci knihovny **react-animate-height**[5]. Vstupními parametry je možné ovlivnit barvu srolované komponenty a okrajů (`color`), barvu písma (`fontColor`), pozadí rozbalené komponenty (`background`), zobrazený název (`name`), zda bude možné komponentu rozbalit (`enabled`), tohoto se využívá např. v případě, že vybraná skupina neobsahuje žádný typ komponenty a funkci (`delete-Component`), která v rámci specifikací smaže skupinu, nebo vymaže vyplněná data v komponentě.

4.4 Ukládání souborů

Zadavatel požadoval mít možnost přidávat libovolné soubory k zakázkám a specifikacím. K tomuto byla implementována funkční komponenta `File-Manager` sloužící pro zobrazení souborů dané specifikace/zakázky na základě id, které je bráno z URL a možnosti nahrání nového souboru na server.

V databázi byla vytvořena tabulka pro ukládání jednotlivých souborů. Tabulka obsahuje sloupce pro název souboru, typ souboru, id specifikace, id zakázky a data, která jsou uložena jako `byte array`. Ukládání dat v tomto formátu je efektivnější pro redukování potřebné paměti. [28]

Jelikož z databáze dojde na frontend soubor jako `byte array`, není tedy ve formátu, ve kterém by ho uživatel mohl stáhnout, proto byla implementována funkce `downloadFile`. Vstupními parametry funkce jsou název, typ souboru

[5] `https://www.npmjs.com/package/react-animate-height`

a data, ze kterých dojde k vytvoření `Blobu` (rozhraní TypeScriptu reprezentující soubor). Na nově vzniklý soubor je vytvořen odkaz a do webové aplikace je přidán neviditelný element, sloužící jako tlačítko pro stáhnutí souboru. V dalším kroku dojde ke kliknutí na tlačítko samotnou aplikací a stažení souboru, po dokončení stahování je neviditelný element z webové aplikace odstraněn.

4.5 Generování PDF

Pro generování PDF byla použita knihovna **iText** 2.2.4. Samotné generování dokumentu je rozděleno na tři části. V první části dojde v vytvoření záhlaví dokumentu, tvořené tabulkou naplněnou daty, následně je vypočítána její výška. Dalším krokem je vytvoření tabulky zápatí a posléze je vypočítána její výška. Rozměry tabulek se mění v závislosti na datech, kterými jsou plněny, například výška záhlaví může být ovlivněna délkou názvu specifikace, kvůli které dojde k rozšíření výšky řádku, aby byl zobrazen celý název.

Po vypočítání výšek záhlaví a zápatí, jsou vypočítány dvě výšky stránky: při vykreslení záhlaví i zápatí na jedné straně a při vykreslení pouze záhlaví. Jelikož zápatí obsahuje informace o ceně, je vykresleno pouze na poslední straně výsledného dokumentu, kdežto záhlaví je vykresleno na všech stranách.

Poslední částí tvorby dokumentu je vkládání atributů. Na základě vypočítaných výšek je zjištěno kolik řádků atributů se vejde na stranu se zápatím a kolik na stranu bez něj. Dochází k plnění tabulky daty, pokud dojde k dosažení vypočítaného maxima, je vygenerována strana bez zápatí. Další možností je, že tabulka nebyla vyplněna na maximum možných záznamů a specifikace již neobsahuje další atributy. Poté se rozhoduje na základě množství atributů ve vyplněné tabulce, pokud je menší nebo rovno maximálnímu počtu atributů na straně se zápatím, jsou data vygenerována na straně se zápatím, v opačném případě je vygenerována strana s atributy a následně strana se zápatím.

Kapitola 5

Testování

Testování aplikace bylo náročné z důvodu častých úprav aplikace na základě pravidelných konzultací se zadavatelem. Proto bylo rozděleno do dvou částí na testování uživatelem, které bylo prováděno panem Janem Prokopem z firmy Prokop & Brož a na testování bez uživatele ve formě heuristické analýzy.

5.1 Uživatelské testování

V průběhu implementace docházelo k testování aplikace Janem Prokopem, který na základě pravidelných schůzek testoval nově implementované části systému. Simuloval reálný chod systému, při kterém otestoval všechny nově implementované funkcionality. Po testování sepsal report obsahující nalezené chyby, včetně situací při kterých chyby vznikly.

Výstupem uživatelského testování je tabulka nalezených chyb a možných vylepšení aplikace viz. 5.1.

5.2 Nielsenova heuristická analýza

Nielsenova heuristická analýza se využívá v době vývoje aplikace/softwaru a zaměřuje se na uživatelské rozhraní. Úkolem této analýzy je odhalení problémů s použitelností uživatelského rozhraní a ověření, zda výsledný produkt bude pro uživatele co nejjednodušší a intuitivní. [29]

Analýza obsahuje deset pravidel, kterými jsou:

- **Viditelnost stavu systému** - viditelnost stavu systému je v aplikaci implementována formou rotujícího kolečka, které znázorňuje čekání na data nebo vyřizování požadavku.
- **Propojení systému a reálného světa** - systém je přizpůsoben, aby práce v něm byla co nejblíže práci v reálném světě. Pro mazání záznamů je použit symbol koše, pro editaci symbol tužky B.8.

- **Uživatelská kontrola a svoboda** - systém umožňuje uživateli se vrátit do předchozího stavu, nebo zrušit zvolenou akci. Tohoto je dosaženo tlačítkem „zpět“ pro navrácení a tlačítkem „zrušit“ pro zrušení mazání záznamu.
- **Konzistence a standardy** - systém je vytvořen v neutrálním designu, tím pádem je v souladu se všemi platformami. Ovládání a popisky jsou v systému konzistentní, konkrétní akce v rámci aplikace jsou vždy stejně pojmenované (ukládání, vytváření atd.).
- **Prevence chyb** - chyby jsou v systému zachycovány a následně dochází k upozornění uživatele. Například při odeslání formuláře, dojde k upozornění na nevyplněná povinná pole, nebo na chybný formát pole (do pole pro telefon byla přidána nečíselná hodnota). Při mazání vzorové komponenty je uživatel upozorněn, že se chystá smazat vzorovou komponentu formou modálu B.6.
- **Rozpoznání místo vzpomínání** - v systému je využito stránkování, pro zmenšení počtu zobrazovaných záznamů. Select fieldy, které ovlivňují generaci navázaných dat jsou po vybrání hodnoty deaktivovány, aby nedocházelo k přepisování dat. Pokud select neobsahuje žádnou hodnotu, nebude vykreslen.
- **Flexibilní a efektivní použití** - systém je intuitivní a jednoduchý na použití pro pracovníky zadavatelské firmy. Usnadňuje uživateli vyplňování formulářů pomocí předvyplněných hodnot.
- **Estetický a minimalistický** - systém je rozdělen do několika sekcí, kde se každá zabývá specifickou částí systému. Na jednotlivé stránky jsou vytvořeny odkazy v menu, zobrazeném na straně aplikace. Méně často používané funkcionality se nenachází v hlavní části menu, ale jsou v sekci „nastavení“.
- **Pomoc uživatelům pochopit, poznat a vzpamatovat se z chyb** - systém upozorní uživatele při chybném vyplnění formuláře formou chybové hlášky u špatně vyplněného pole. Jedná se například o nevyplnění povinné hodnoty nebo špatný formát vstupu B.6.
- **Nápověda a návody** - systém poskytuje nápovědy při vyplňování formulářů a tvorbě objektů hodinek formou textových zpráv u vyplňovaného pole.

5.3 Výsledek testování

Při uživatelském testování bylo objeveno několik chyb (viz. tabulka 5.1), které byly následně opraveny a znovu otestovány. Jednotlivá pravidla heuristické

Tabulka 5.1: Tabulka chyb nalezených uživatelským testováním

Popis problému	**Řešení**
Chybí možnost označit specifikaci za vzorovou	Přidán checkbox pro označení
Chybí upozornění při mazání vzorové specifikace	Existující upozornění bylo rozšířeno o upozornění, že se jedná o vzorovou specifikaci
Chybí upozornění při tvorbě vzorové specifikace	Přidáno upozornění, že se vytváří nová vzorová specifikace
Neúspěšné uložení kopie komponenty	Kopii bylo odebráno id, čímž došlo k úspěšnému vytvoření nového objektu v databázi
Omezená délka řetězce u telefonních čísel a typu přílohy	Navýšena délka čísla v databázi
Chybějící možnost překladu pro cizince	Přidáno pole pro překlad
Textové pole musí být schopno pojmou všechny typy telefonních čísel	Změněn typ čísla v databázi a zvětšena velikost
Zobrazení zaměstnanců a klientů na základě pořadí, ve kterém byli přidání do databáze	Upraveno na řazení dle příjmení

Tabulka 5.2: Tabulka výsledků Nielsenovy heuristické analýzy

Pořadí pravidla	**1**	2	3	4	5	6	7	8	9	10
Hodnocení	5	5	5	4	5	4	4	5	4,5	4

analýzy byla ohodnocena body 0-5, kde 0 znamená nesplnění pravidla a 5 úplné splnění. Výsledkem heuristické analýzy je tabulka 5.3, ze které vyplývá, že nedošlo k zásadnímu porušení žádného z pravidel.

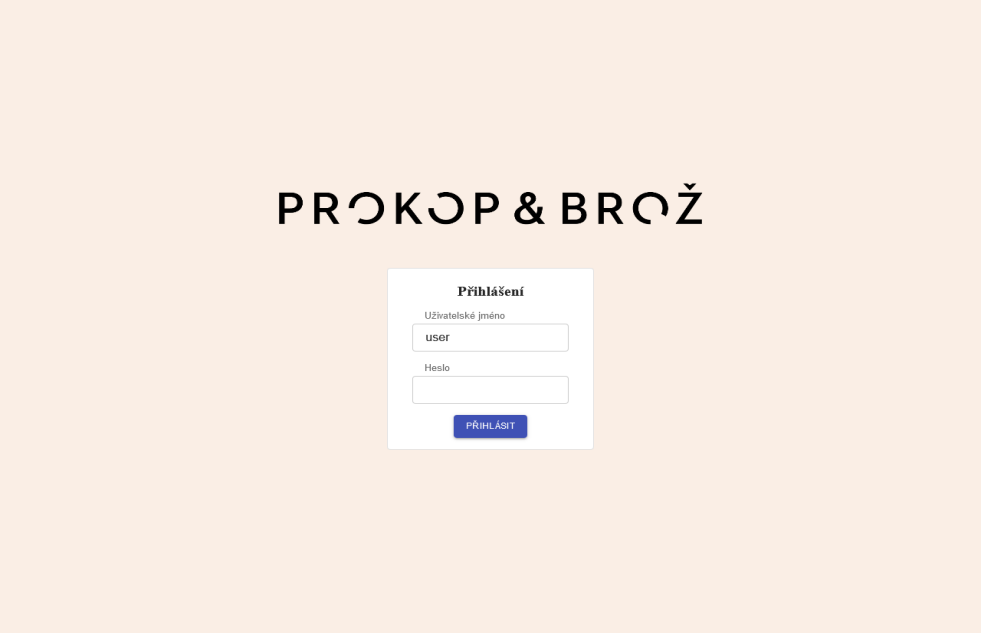
Kapitola 6

Vzhled, dokumentace a nasazení aplikace

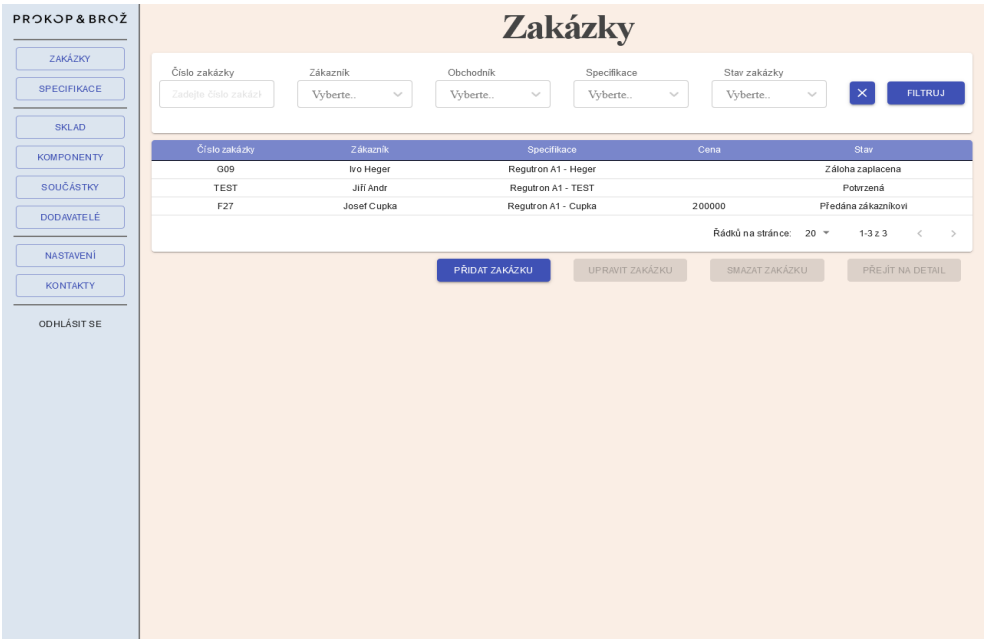
6.1 Vzhled

Po načtení aplikace se uživatel vyskytuje na přihlašovací stránce (dostupné v testovacím prostředí na `https://dev.app.prokop-broz.cz/login`). Informační systém je přístupný jen pro přihlášené uživatele.

Úspěšně přihlášenému uživateli se na úvodní straně zobrazí soupis veške-



Obrázek 6.1: Úvodní strana aplikace



Obrázek 6.2: Soupis zakázek

rých zakázek, jelikož se jedná o nejdůležitější část systému společně se specifikacemi. Pokud je uživatel přihlášen na počítači, v levé části okna se mu zobrazí menu. Uživateli přihlášenému na mobilním zařízení je zobrazeno menu srolované, které je reprezentováno třemi čarami v pravém horním rohu aplikace (viz. příloha B.1), po kliknutí na tlačítko je menu maximalizováno.

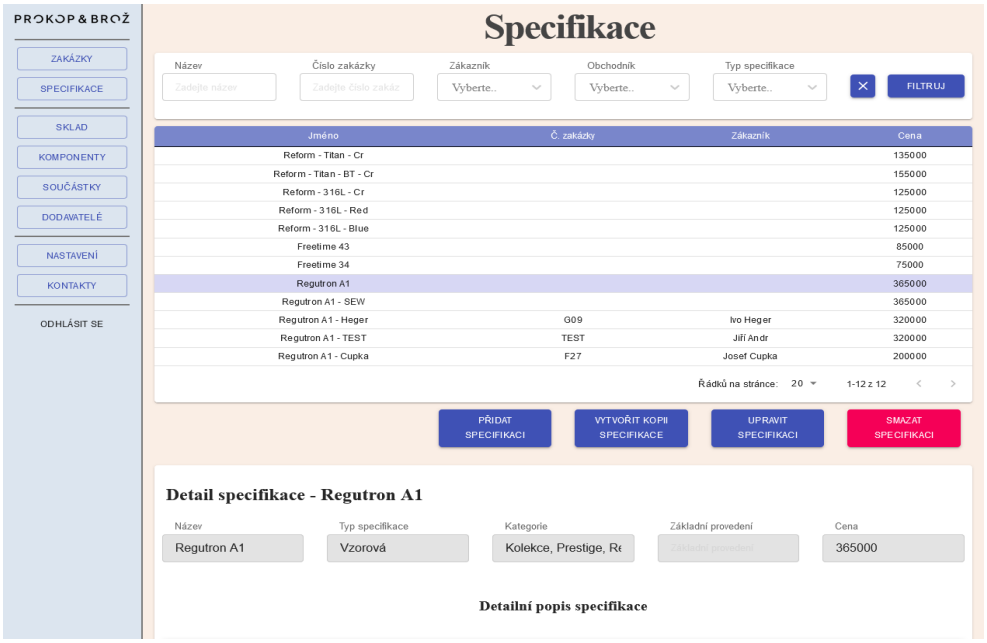
Strana se specifikacemi obsahuje základní informace o specifikacích zobrazených v tabulce, společně s možností filtrování B.2. Kliknutím na záznam tabulky, dojde k načtení detailu, který je zobrazen pod tabulkou. V detailu se navíc vykresluje soupis příloh vybrané specifikace B.3.

Tvorba nové specifikace je implementována pomocí formuláře, který obsahuje rozšiřitelné komponenty pro každou skupinu. Uvnitř skupiny jsou jednotlivé typy komponent, které jsou její součástí.

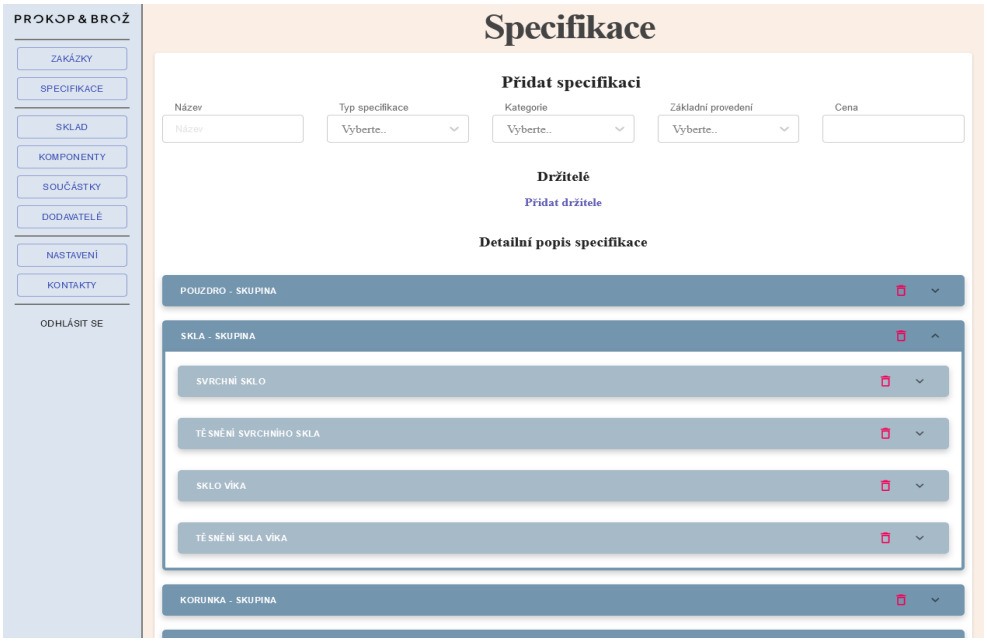
V nastavení se nachází číselníky, které jsou využívány jen zřídka. Jsou zde například defaultní atributy, které jsou vykresleny v přesouvatelných komponentách, které je možné uchopit za `dragHandle` v levé části komponenty a změnit pořadí, ve kterém se vykreslují v daném typu komponenty.

6.2 Dokumentace

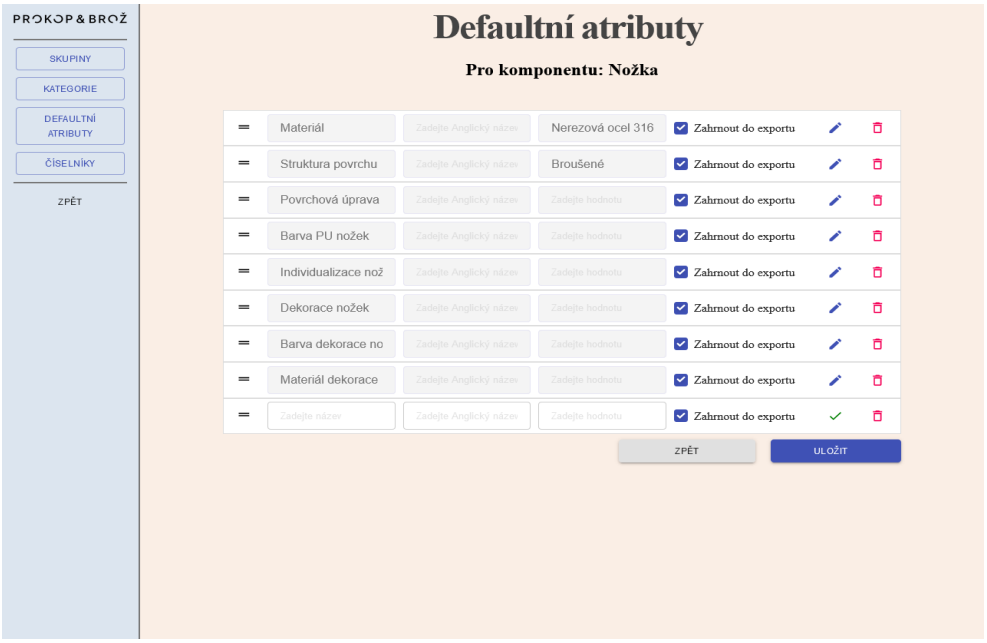
V rámci implementace aplikace docházelo ke komentování metod a funkcí pomocí referenčních značek, které se využívají při tvorbě aplikací psané v jazyce Java. Byly využívány názvy funkcí a proměnných, aby při prvním pohledu



Obrázek 6.3: Tvorba nové specifikace



Obrázek 6.4: Tvorba nové specifikace



Obrázek 6.5: Defaultní atributy nožky

bylo jasné, k čemu mají sloužit. Pokud se v rámci například funkce vyskytovala část kódu, která by mohla být čtenáři nejasná, byla doplněna o krátký komentář, popisující danou část kódu detailněji.

Dokumentace pro backend byla vygenerována pomocí **Javadoc Tool**, který vytvoří API dokumentaci projektu ve formátu HTML.

6.3 Nasazení

Zadavatel se rozhodl využít svého poskytovatele IT služeb pro zajištění serveru a domény, na které má být aplikace nasazena. Z tohoto důvodu došlo ke zdržení a výsledný produkt bude nasazen po odevzdání bakalářské práce. Nasazení bude odprezentováno v době obhajoby práce.

Závěr

Bakalářská práce byla zaměřena na rozšíření webové aplikace pro výrobu luxusních hodinek. Tato aplikace je v současné době testována ve vývojovém prostředí pro reálné využití ve firmě Prokop & Brož.

Cílem bylo seznámit se s předchozím řešením, zanalyzovat ho a následně na něj navázat. V závislosti na analýze došlo k návrhu uživatelského rozhraní specifikací a návrhu nového doménového modelu, kterým by došlo k zefektivnění ukládání dat na straně databáze. Dále se práce soustředila na úpravu uživatelského rozhraní, které bylo v původní verzi příliš často rozšířeno o modály, kterými byla práce s aplikací prodlužována. Dalšími cíli bylo generování tisknutelných reportů specifikace a možnost práce se soubory.

V průběhu implementace aplikace docházelo k pravidelným schůzkám s představitelem firmy, který následně testoval nově implementované funkcionality. Výsledky testování byly společně s připomínkami zapracovány do aplikace pro její zefektivnění a správný chod. Druhou testovací metodou byla Nielsenova heuristická analýza, která v deseti pravidlech testuje použitelnost uživatelského rozhraní. Po jejím vyhodnocení bylo zjištěno, že nově vytvořené uživatelské rozhraní není v rozporu s předepsanými pravidly. Všechny nastavené cíle byly v bakalářské práci splněny.

K tvorbě webové aplikace byly využity moderní technologie, kterými jsou React od společnosti Facebook, Spring a PostgreSQL.

Při vytvářených úpravách se vycházelo z požadavků zadavatele. Tyto požadavky byly plně zapracovány tak, aby přínos aplikace byl co nejefektivnější. Zároveň zde byla snaha, aby aplikace mohla být případně v budoucnosti rozšířena o další funkcionalitu, která vyplyne z reálného provozu.

Literatura

[1] O'REILLY: *Working with React Native - Learning React Native*. [online], [cit. 2021-04-19]. Dostupné z: https://www.oreilly.com/library/view/learning-react-native/9781491929049/ch02.html

[2] POLCAR, V.: *IS pro podporu společnosti zabývající se výrobou luxusních hodinek*. Diplomová práce, České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

[3] CHRISTENSSON, P.: *Frontend Definition*. [online], [cit. 2021-04-19]. Dostupné z: https://techterms.com/definition/frontend

[4] MOZILLA: *What Is JavaScript? - Learn Web Development | MDN*. [online], [cit. 2021-04-19]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript

[5] MÁCA, J.: *Lekce 1 - Úvod do React*. [online], [cit. 2021-04-16]. Dostupné z: https://www.itnetwork.cz/uvod-do-react

[6] *React Archives*. [online], [cit. 2021-04-17]. Dostupné z: https://programmingwithmosh.com/category/react/

[7] MOZILLA: *Document Object Model (DOM) - Web APIs*. [online], [cit. 2021-04-19]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model

[8] RAVICH, A.: *React Virtual DOM Explained in Simple English*. [online], [cit. 2021-04-19]. Dostupné z: https://programmingwithmosh.com/react/react-virtual-dom-explained/

[9] TYPESCRIPTLANG: *Documentation - TypeScript for the New Programmer*. [online], [cit. 2021-04-20]. Dostupné z: https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html

[10] KAMMALAWATTA, O.: Getting Started With useQuery (React Query). [online]. Dostupné z: https://medium.com/swlh/getting-started-with-usequery-react-query-9ea181c3dd47

[11] TANSTACK: *Queries*. [online], [cit. 2021-04-20]. Dostupné z: https://react-query.tanstack.com/guides/queries

[12] TANSTACK: *useQuery*. [online], [cit. 2021-04-20]. Dostupné z: https://react-query.tanstack.com/reference/useQuery

[13] *useMutation*. [online], [cit. 2021-04-20]. Dostupné z: https://react-query.tanstack.com/reference/useMutation

[14] JÖCH, D.: *Functional vs Class-Components in React*. [online], [cit. 2021-04-20]. Dostupné z: https://djoech.medium.com/functional-vs-class-components-in-react-231e3fbd7108

[15] FACEBOOK INC.: *Introducing JSX – React*. [online], [cit. 2021-04-21]. Dostupné z: https://reactjs.org/docs/introducing-jsx.html

[16] *React Hook Form*. [online], [cit. 2021-04-21]. Dostupné z: https://react-hook-form.com/

[17] *Redux Form*. [online], [cit. 2021-05-11]. Dostupné z: https://redux-form.com/8.3.0/docs/gettingstarted.md/

[18] PASTORINO, M.: *Frontend vs Backend: What's The Difference?* [online], [cit. 2021-04-27]. Dostupné z: https://www.pluralsight.com/blog/software-development/front-end-vs-back-end

[19] *Spring Framework*. [online], [cit. 2021-05-07]. Dostupné z: https://spring.io/projects/spring-framework

[20] POSTGRESQL: *PostgreSQL: About*. [online], [cit. 2021-04-27]. Dostupné z: https://www.postgresql.org/about/

[21] LIQUIBASE: *Liquibase Online Help*. [online], [cit. 2021-05-04]. Dostupné z: https://docs.liquibase.com/home.html

[22] ITEXT: *iText 7 Java*. [online], [cit. 2021-05-07]. Dostupné z: https://itextpdf.com/en/resources/api-documentation/itext-7-java

[23] JAN KUŽNÍK, J. L.: *Řídí Hodiny Na Václaváku, Vyrobí Hodinky i Podle Lokomotivy. Jediný Pražský Hodinář*. [online], [cit. 2021-04-27]. Dostupné z: https://www.idnes.cz/technet/reportaze/ridi-hodiny-na-vaclavaku-vyrobi-hodinky-i-podle-lokomotivy-jediny-prazsky-hodinar.A110920_175405_tec_reportaze_kuz

[24] PROKOP & BROŽ: *Zakladatelé*. [online], [cit. 2021-04-19]. Dostupné z: http://www.prokop-broz.cz/zakladatele.html

[25] PROKOP & BROŽ: *Příběh značky*. [online], [cit. 2021-04-16]. Dostupné z: http://www.prokop-broz.cz/pribeh-znacky.html

[26] NOVOTNÝ, O.: *Mistři komplikací. Jak malé hodinářské značky z Česka prorazily až do světa*. [online], [cit. 2021-04-27]. Dostupné z: https://procne.ihned.cz/c1-66829230-mistri-komplikaci-jak-male-hodinarske-znacky-z-ceska-prorazily-az-do-sveta

[27] ERIKSSON, U.: *Functional vs Non-Functional Requirements - Understand the Difference*. [online], [cit. 2021-04-26]. Dostupné z: https://reqtest.com/requirements-blog/functional-vs-non-functional-requirements/

[28] EUGENE, P.: *What Is a Byte Array?* [online], [cit. 2021-05-03]. Dostupné z: http://www.easytechjunkie.com/what-is-a-byte-array.htm

[29] NIELSEN, J.: *10 Usability Heuristics for User Interface Design*. [online], [cit. 2021-05-11]. Dostupné z: https://www.nngroup.com/articles/ten-usability-heuristics/

Seznam použitých zkratek

API Application Programming Interface.

DOM Document Object Model.

HTML HyperText Markup Language.

JS JavaScript.

MVC Model-View-Controller.

PDF Portable Document Format.

SQL Structured Query Language.

TS TypeScript.

URL Uniform Resource Locator.

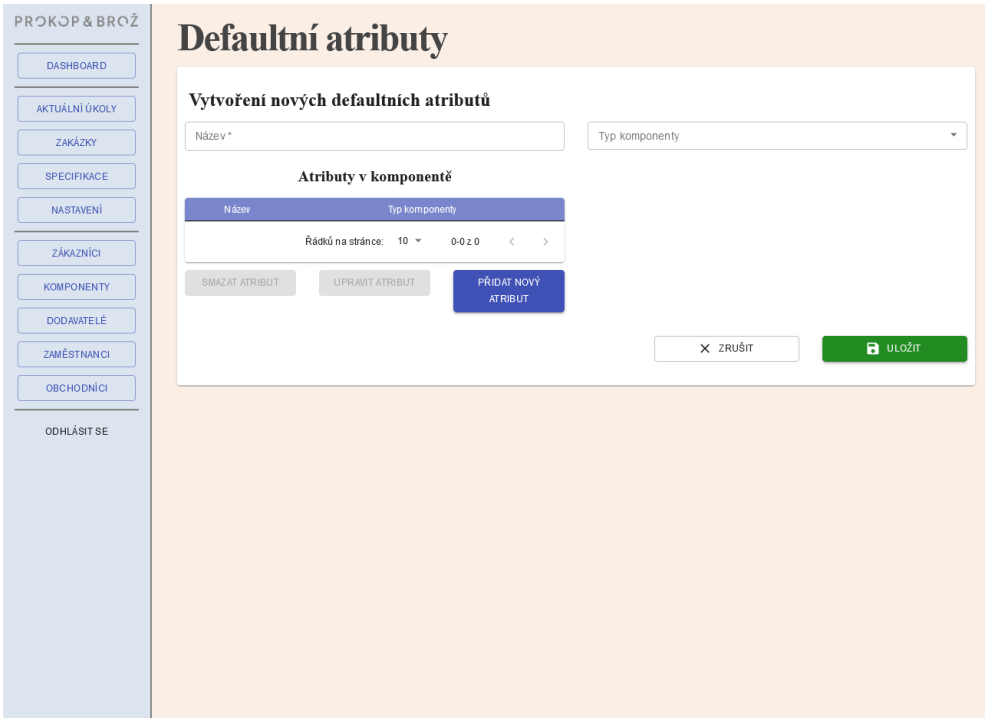
VDOM Virtual Document Object Model.

XML Extension Markup Language.

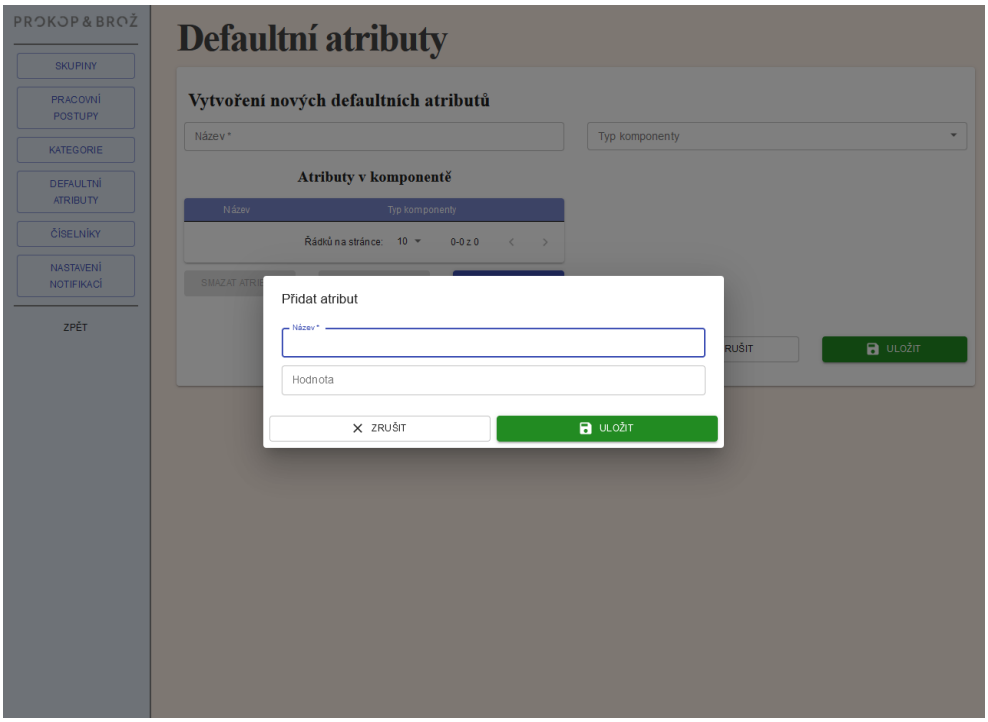
Příloha A

Původní aplikace

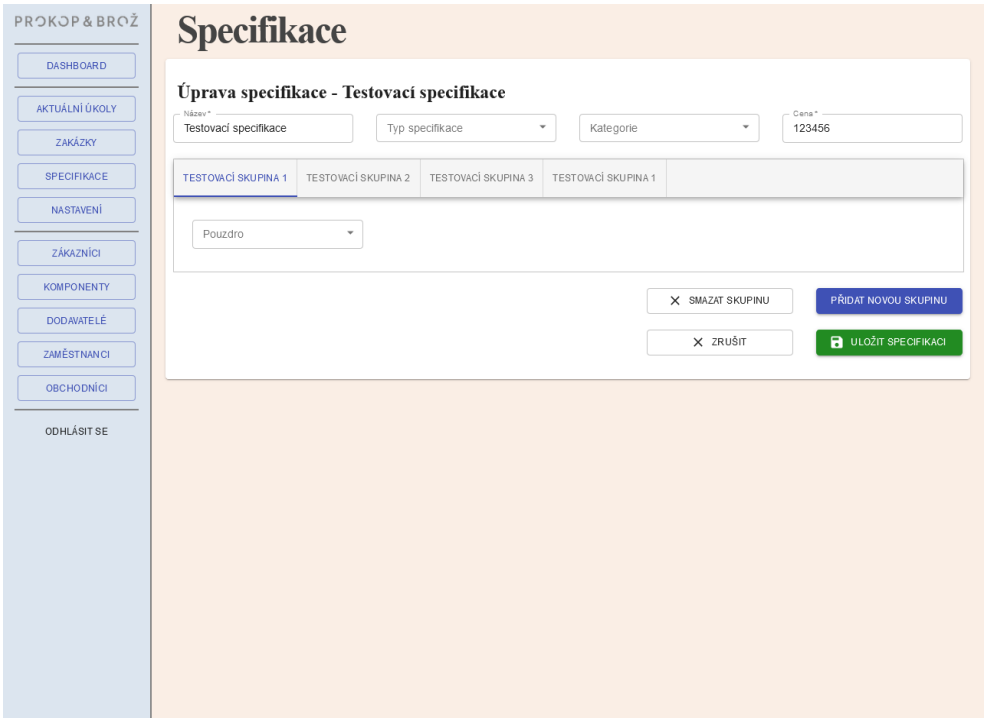
Obrázek A.1: Přehled skupin defaultních atributů, přiřazeným k typům komponent



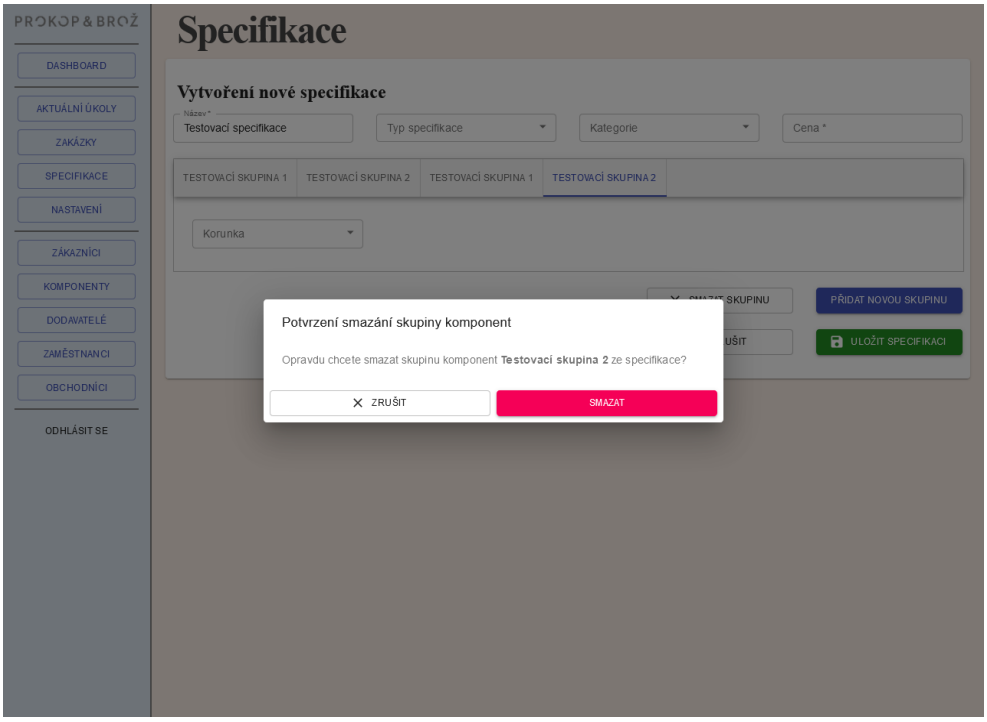
Obrázek A.2: Tvorba nových defaultních atributů



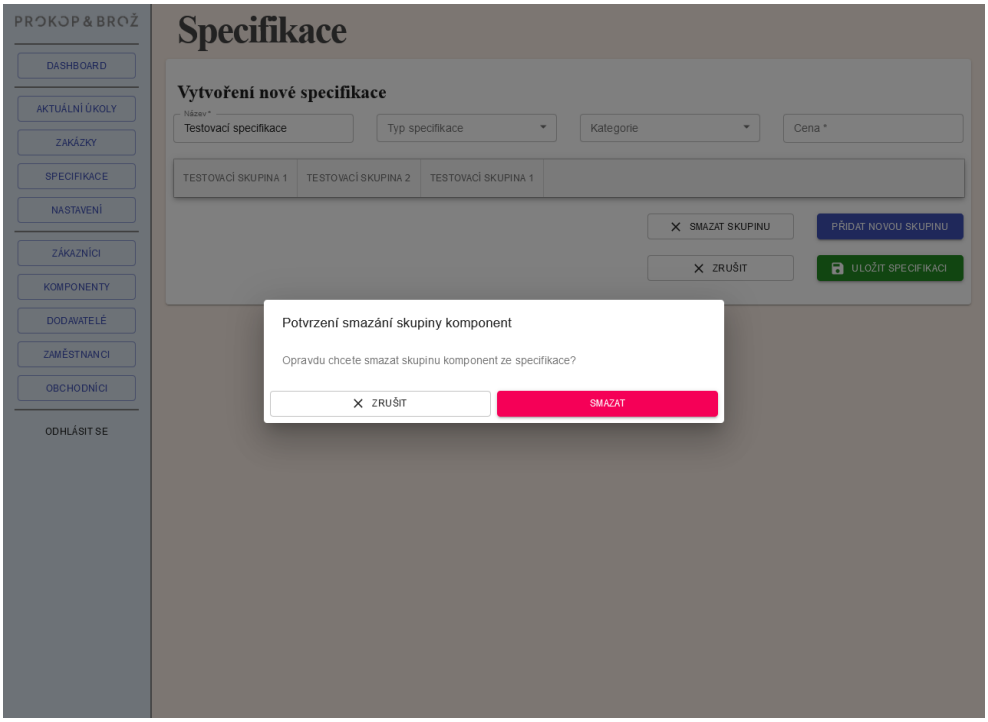
Obrázek A.3: Přidání nového atributu do skupiny defaultních atributů



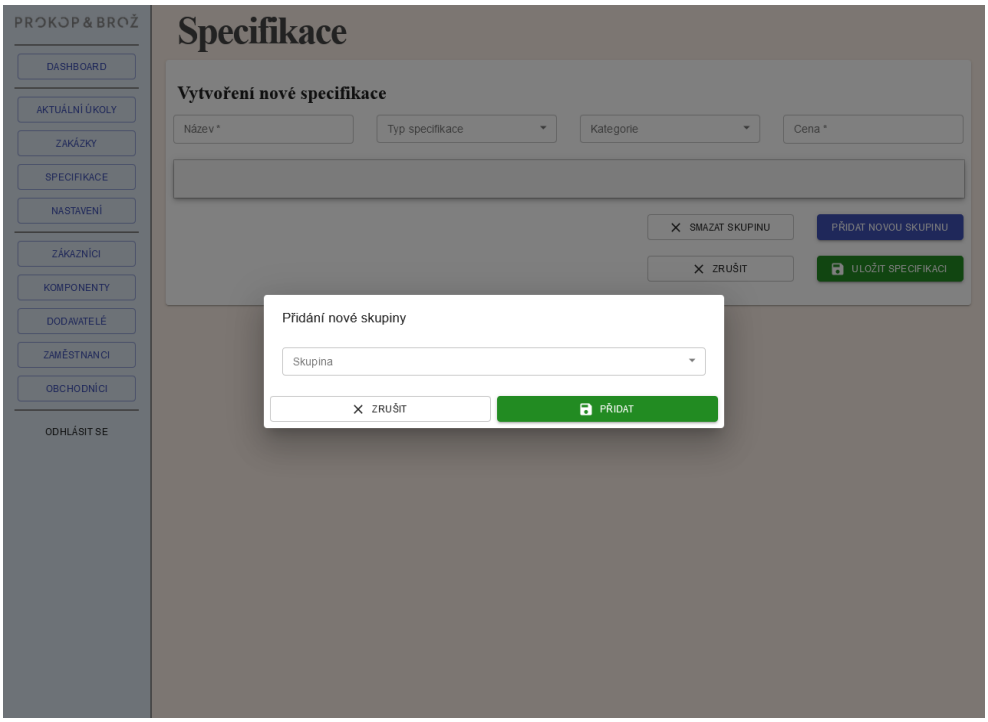
Obrázek A.4: Tvorba nové specifikace



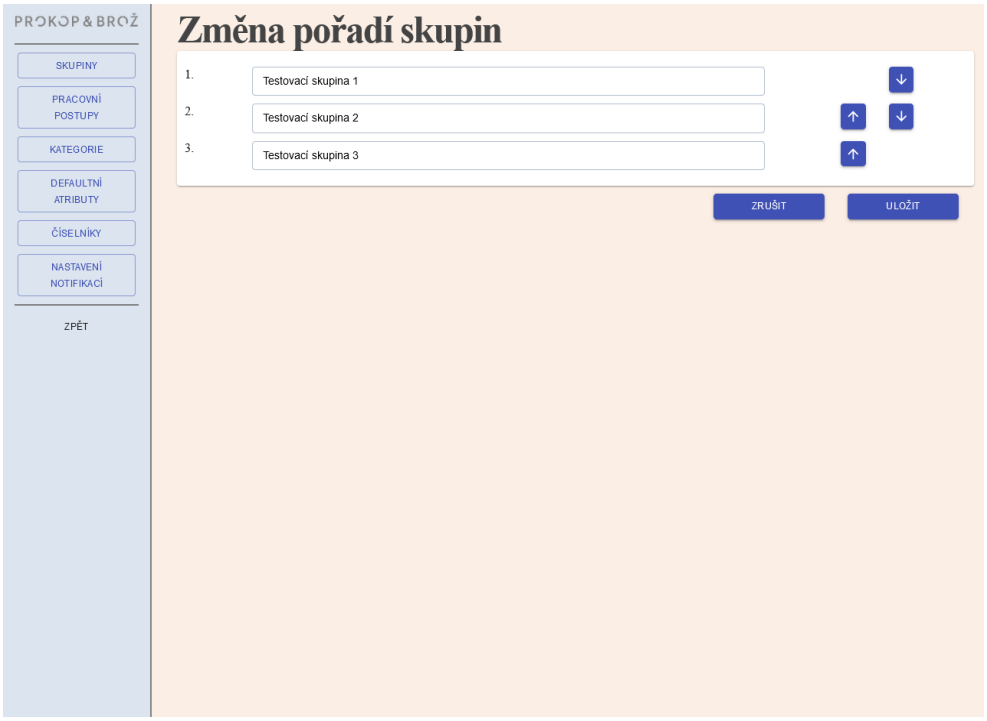
Obrázek A.5: Mazání skupiny komponent ze specifikace



Obrázek A.6: Mazání náhodné skupiny komponent



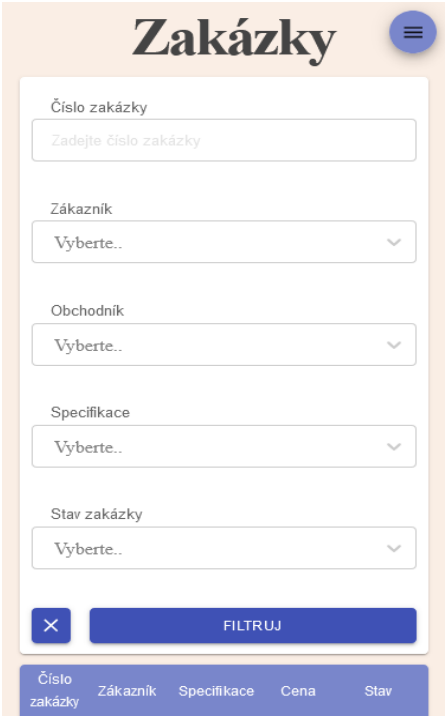
Obrázek A.7: Přidání nové skupiny do specifikace



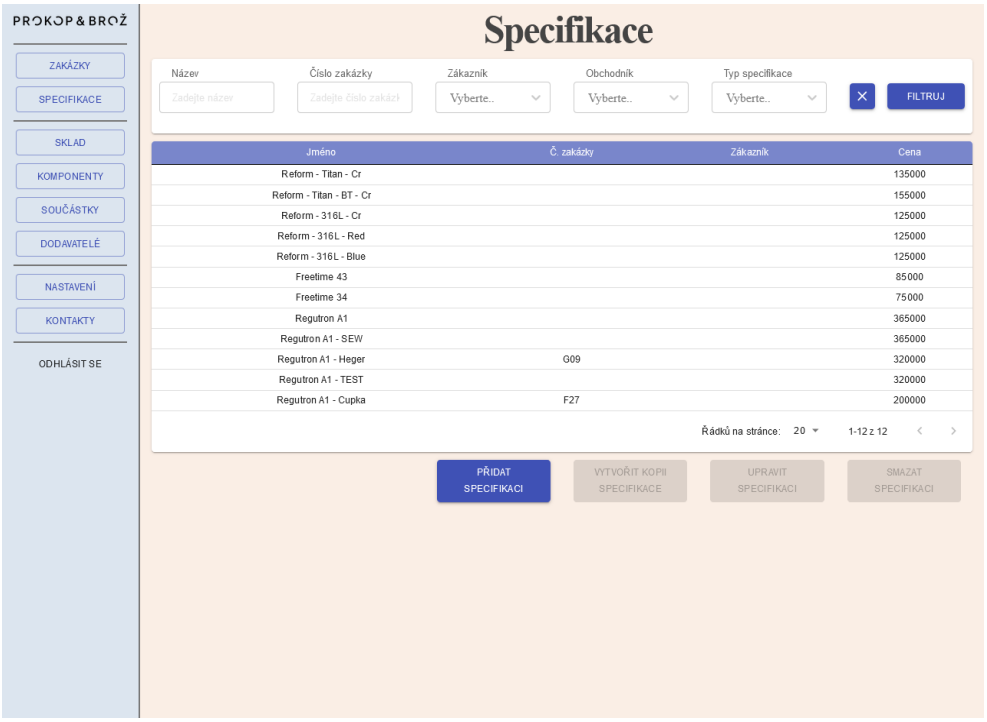
Obrázek A.8: Změna pořadí skupin

Příloha B

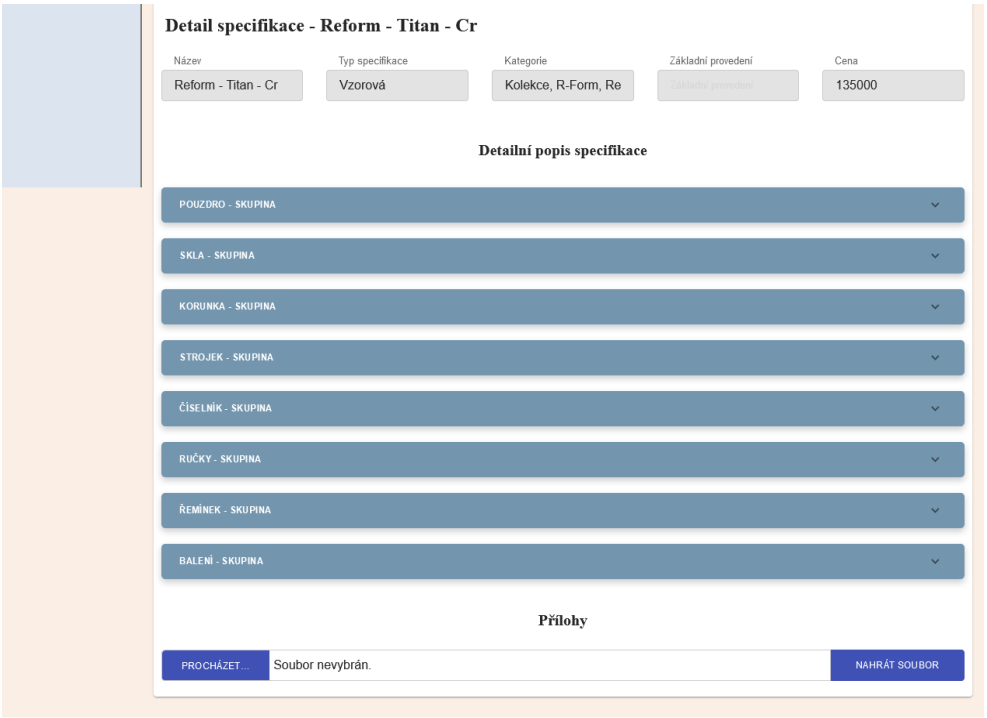
Rozšířená aplikace



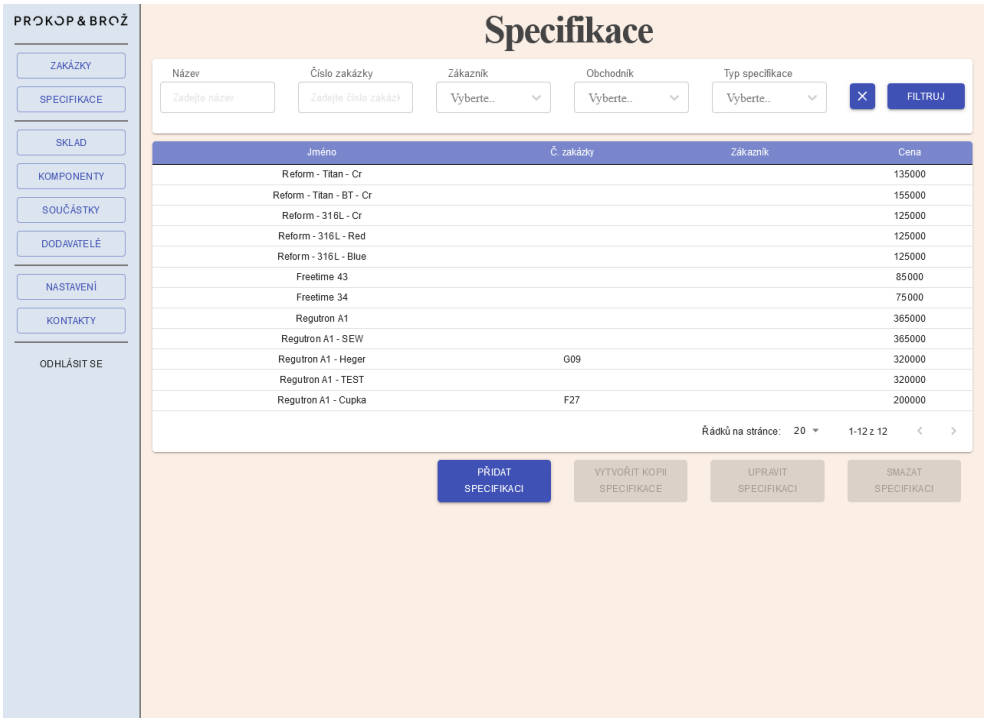
Obrázek B.1: Úvodní strana aplikace na mobilním zařízení



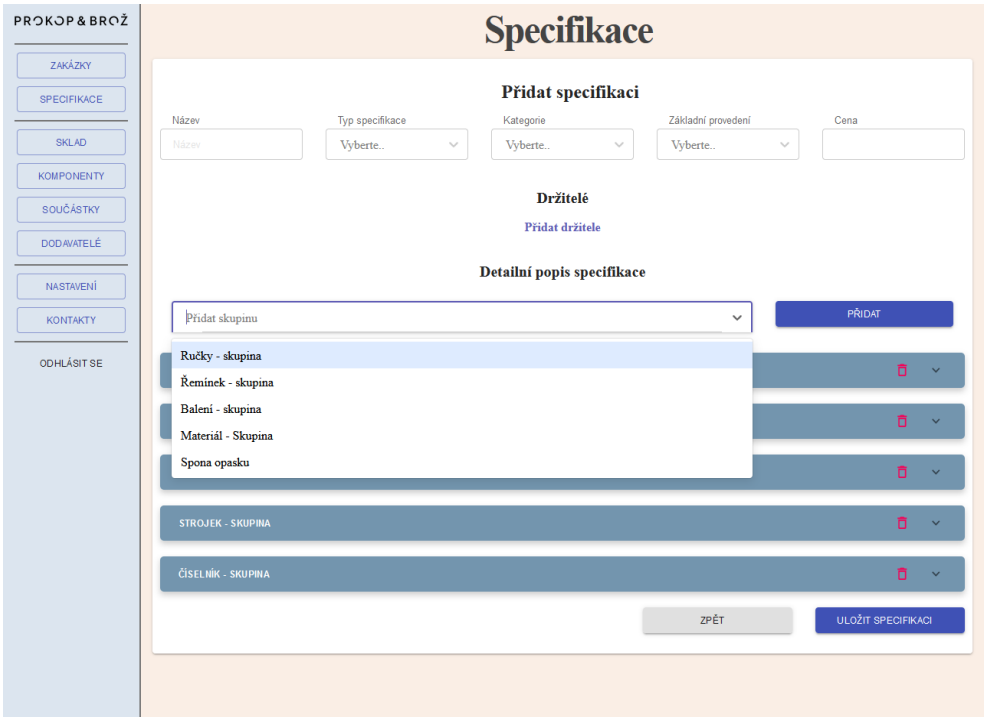
Obrázek B.2: Soupis specifikací



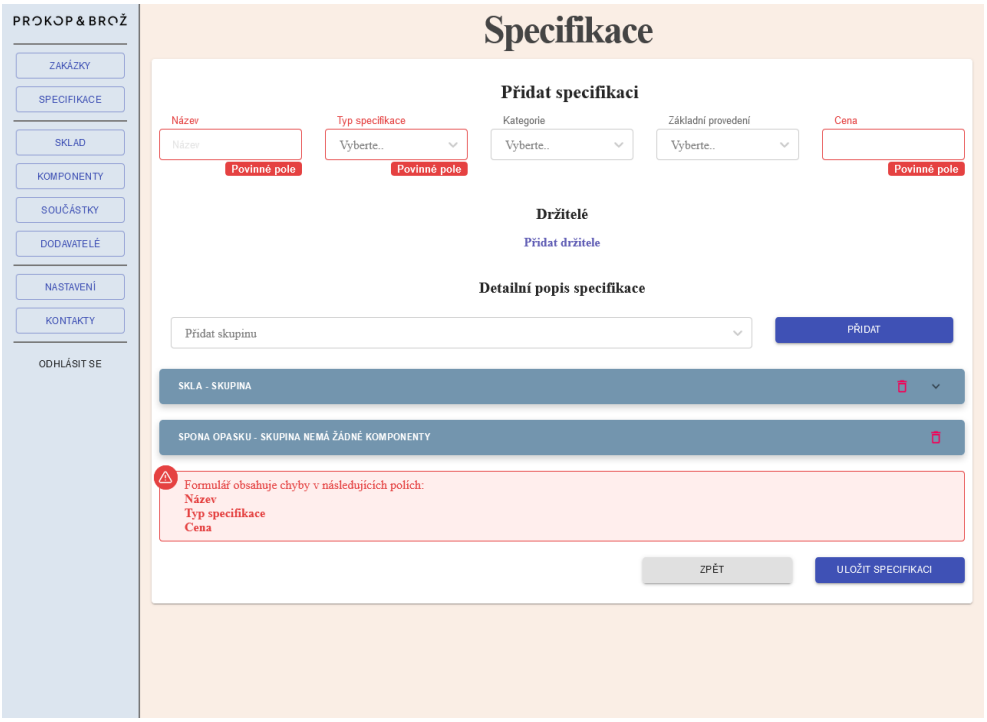
Obrázek B.3: Detail specifikace s přílohami



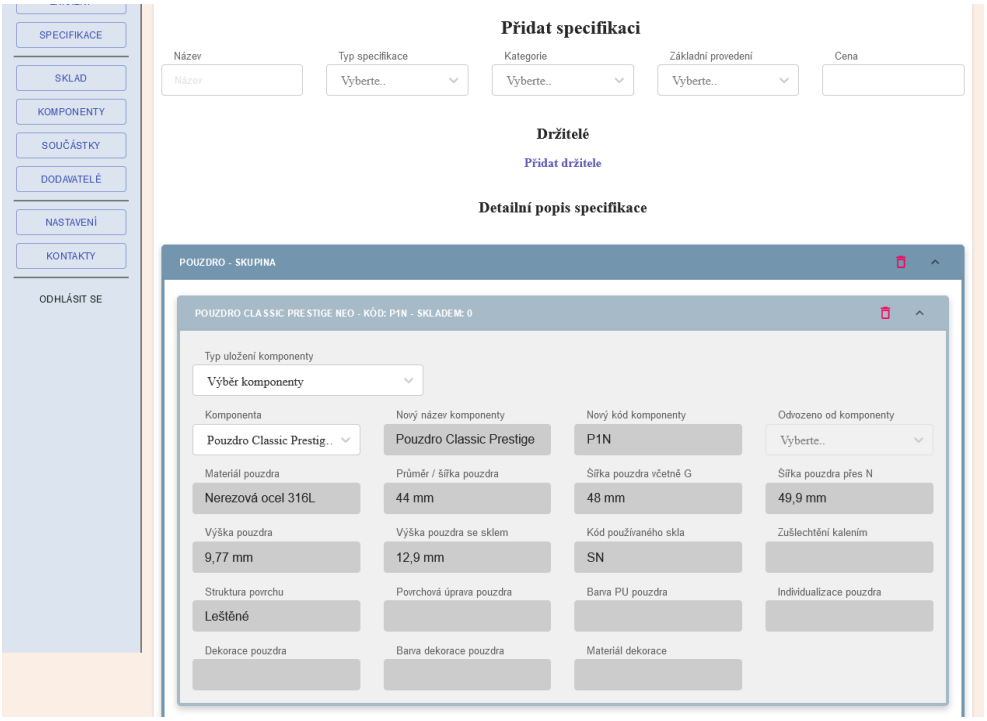
Obrázek B.4: Potvrzení odstranění specifikace



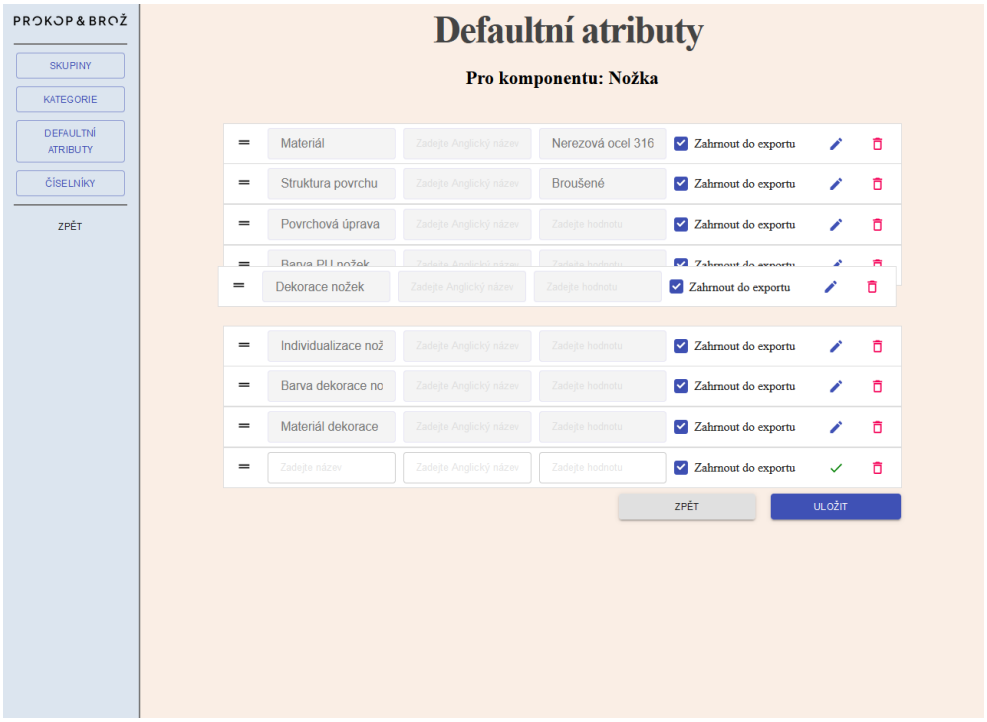
Obrázek B.5: Select se skupinami při tvorbě specifikace



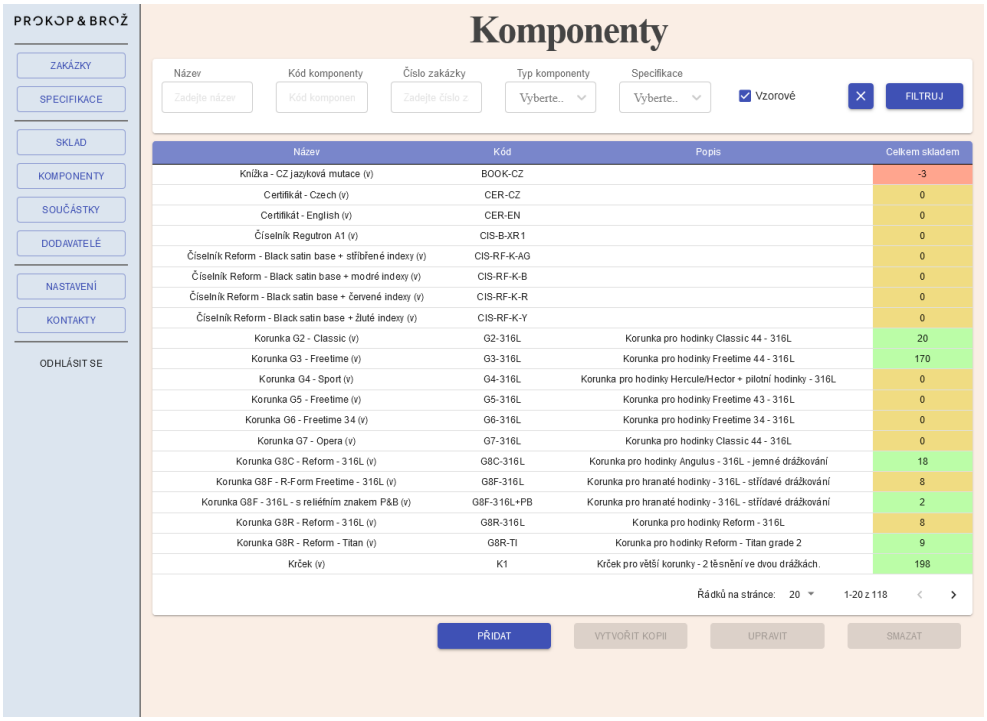
Obrázek B.6: Validace formuláře specifikace



Obrázek B.7: Roztažitelné komponenty specifikace



Obrázek B.8: Změna pořadí atributů



Obrázek B.9: Soupis komponent

Příloha C

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `src`
  - `impl` ..... zdrojové kódy rozšířené aplikace
  - `impl-before` ..... zdrojové kódy předchozího stavu aplikace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
- `documentation` ..... dokumentace backendu