Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra informatiky a kvantitativních metod

Systém pro správu majetku v Angular frameworku

Bakalářská práce

Autor: Milan Knop
Studijní obor: Aplikovaná informatika

Vedoucí práce: Ing. Pavel Kříž, Ph.D.

Hradec Králové duben 2021

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedených zdrojů.

V Hradci Králové dne 26.4.2021 Milan Knop

Poděkování:

Děkuji vedoucímu bakalářské práce Ing. Pavlu Křížovi, Ph.D. za metodické vedení práce, směrování a cenné rady při zpracování bakalářské práce. Také bych chtěl poděkovat své rodině, za velikou trpělivost.

Anotace

Práce je zaměřena na vývoj systému evidence majetku ve frameworku Angular pro organizaci s vysokou fluktuací aktiv a velkým počtem uživatelů. Aplikační logika systému je rozdělena na stranu klienta, tvořena ve frameworku Angular a na stranu serveru, tvořenou frameworkem NestJS.

Tvorbě vlastní aplikace předchází průzkum dostupných aplikací pro evidenci majetku, představení frameworků Angular a NestJS. U každého frameworku jsou popsány základní stavební prvky a knihovny využité pro danou část aplikace.

Praktickou část tvoří systém evidence majetku ve frameworku Angular, ve kterém jsou implementovány funkce, které usnadňují práci v případě velkého počtu položek či častých úprav majetku. Zejména se jedná o funkce hromadných úprav, tvorby pohledů, ukládání sestav pro pozdější využití a podpůrné funkce pro knihovnu ag-Grid.

Title: Assets management system in Angular framework

Annotation

This Bachelor Thesis is focused on assets management development in framework Angular for an organization with a lot of transfers of assets and also a large number of users. The application logic of the system is separated into the client-side, made with Angular framework, and for server-side, developed in NestJS framework.

There are checks on common systems for assets evidence, show up frameworks Angular and NestJS. Also, there described the basic techniques and additional libraries used in the development, for each framework and for all parts of the application.

The practical part starts by describing the assets management system in Angular, where the main functionality is implemented. Developed functionality and own adjusted business logic make the whole application flexible and enable it to make a large number of changes on a large number of assets, very quickly and effectively. Implementation is designed to make effective working with multiple changes, creating the views, list management, and some supporting functions for ag-Grid.

Obsah

Seznam obrázků

1 Úvod

V dnešní době se využívají počítače téměř ke všem administračním úkonům. Málokdo si dnes už dokáže představit, že by jakoukoli agendu vedl pouze v listinné podobě. I když v této oblasti došlo k výraznému pokroku, spousta lidí již neřeší, zda software, který využívají, by nemohl fungovat lépe, či zda by nešlo dělat něco jinak. Vzhledem k osobním zkušenostem, z takovéhoto přechodu z listinné podoby do evidence pomocí softwaru v oblasti evidence „oběžného“ majetku a spotřebních předmětů (dále jen majetku), je tato práce zaměřena na vývoj a implementaci takovéhoto softwaru.

Webová aplikace řešená v rámci této bakalářské práce slouží jako základ pro rozsáhlejší aplikaci správy majetku. Má sloužit jako pomocný software správcům majetku, pro přehled a evidenci v rámci společnosti, kde je velký počet uživatelů a pohyb majetku. Je tvořena s důrazem na hromadné úpravy tak, aby ušetřila co nejvíce času jejím uživatelům.

V aplikaci je možné vytvářet uživatele, nastavovat jim práva, díky kterým je možné definovat určité role v rámci aplikace. Vkládat majetek a přiřazovat jej do kategorií, které jsou tvořeny stromovou strukturou a v případě nutnosti jej rozřazovat do sekcí, resp. jednotek společnosti, které jsou také tvořeny stromovou strukturou. V reálném provozu probíhá import uživatelů, majetku a kategorií pomocí automatizovaných operací.

2 Analýza

V této kapitole je určení cílů, funkčnosti programu a zmapování současné situace v oblasti aplikací pro evidenci majetku.

2.1 Cíl bakalářské práce

Cílem práce je vytvořit webovou aplikaci pro správu majetku postavenou na frameworku Angular. Aplikace bude umožňovat správu uživatelů a majetku jim přiděleného, vytvářet předávací protokoly, karty přiděleného majetku osobě, operace s majetkem v dočasných sestavách, vyhledávání nad databází majetku a bude uchovávat jeho historii. Dále bude umožňovat jednotlivým uživatelům tvorbu jejich sestav majetku a vytváření poznámek k jednotlivým položkám.

2.2 Požadavky na aplikaci

V této sekci jsou stanoveny hlavní funkční požadavky na aplikaci.

2.2.1 Přístupnost

Systém je možné provozovat na oddělené síti, bez možnosti přístupu na internet. Přístup k němu by měl být prostřednictvím webového prohlížeče tak, aby nebyla vyžadována žádná další instalace na pracovní stanici. Přístup pomocí webového prohlížeče zajistí i nezávislost na operačním systému. Zároveň si jej bude moci každý uživatel v síti zobrazit a tím i vyhledat majetek jemu přidělený. Vzhledem k charakteru aplikace a provozu na interní síti není vyžadována optimalizace pro mobilní zařízení, či barvoslepost.

2.2.2 Kategorizace

Systém umožňuje procházení majetku přes omezení kategorií pomocí stromové struktury minimálně do 4. úrovně.

2.2.3 Kapacita

Kapacita pro počet záznamů by měla být neomezená, minimálně však 20000 položek majetku a 300 uživatelů, resp. osob, kterým je svěřen majetek a přístup k systému, při nejmenším z hlediska náhledu na vlastní majetek a editace položky vlastní poznámka.

2.2.4 Uživatelské účty a oprávnění

V systému by měly existovat minimálně 4 druhy účtů a to: administrátor, správce majetku, lokální správce majetku (dále jen lokální správce) a uživatel. V ideálním případě je možné každému uživateli nastavovat práva individuálně.

2.2.4.1 Administrátor

Administrátorský účet může tvořit stromovou strukturu majetku, vkládání majetku do systému a přiřazování jej jednotlivým správcům pomocí požadavku na přijetí majetku do evidence, vytváření, editace a odstraňování správců majetku hlavních kategorií a uživatelů. Přiřazovat uživatelské účty pod správce majetku.

2.2.4.2 Správce majetku

Účet s těmito právy může spravovat majetek jemu přiřazený, resp. jeho úseku. Dále může zasílat požadavky na přeřazení majetku mezi správci majetku a zároveň je potvrzovat a tím přijímat majetek do svého úseku, vytvářet lokální správce majetku a přiřazovat jim uživatele a majetek do užívání.

2.2.4.3 Lokální správce

Úkolem lokálního správce je sledovat pohyb majetku mezi uživateli jemu svěřenými. U majetku může editovat pouze informativní položky jako jsou umístění, uživatel, poznámky, přiložené soubory a obrázky. Dále může zadávat požadavek na přesun majetku z jeho lokální oblasti pod jiného správce majetku či lokálního správce, požadavek na vyřazení majetku z evidence, který přijde na potvrzení správci majetku.

2.2.4.4 Uživatel

Uživatel může majetek vyhledávat, zobrazit a vytvářet si k němu vlastní poznámky. Vytvářet vlastní sestavy majetku pro pozdější použití.

2.2.5 Hromadné funkce a vyhledávání

V aplikaci je možné vyhledávat podle veškerých zadaných údajů, zejména inventární číslo, výrobní číslo, uživatel.

Vzhledem k vysoké fluktuaci majetku systém umožňuje hromadné operace nad vybranými položkami majetku, zejména hromadná změna uživatele, zadání požadavku na převod či odstranění, resp. vyřazení záznamu. Vybrané položky systém umí uložit do pracovních sestav pro pozdější využití, jakožto jejich načtení, archivaci a úpravu.

Majetkovým správcům umožňuje tisknutí předávacích protokolů, seznamů majetku na místnosti a informativních sestav.

2.3 Průzkum dostupných aplikací

V této kapitole jsou popsány dostupné aplikace pro evidenci majetku. Většina z nich má evidenci majetku jako součást širší aplikace a není přímo soustředěna na evidenci majetku. Pokud není uvedeno jinak, byl software testován autorem práce.

2.3.1 SW Klid

Obrázek 1, logo SW Klid, zdroj: https://www.swklid.cz/

„SW KLID je databázový systém přístupný přes webový prohlížeč, který je určený pro efektivní správu majetku, infrastruktury a pracovního prostředí." (EASY FM, © 2014-2020). Software je provozován výhradně online a veškerá data jsou uložena u společnosti EASY FM.

Evidence majetku je zde koncipována především pomocí umístění majetku v objektech (areály, budovy, patra, místnosti). Není zde možné přidávat k majetku odpovědnou osobu, resp. svěřit majetek přímo osobě, ale pouze zadat jeho umístění.

K majetku vloženému v evidenci je možné vkládat soubory jako přílohy, ale chybí zde položky jako výrobní číslo, inventární číslo a odpovědná osoba. Tyto položky je možné vložit do komentáře, ale systém v komentáři nevyhledává, tudíž není možné dohledat věc podle výše zmíněných údajů.

V systému není možné operovat nad více položkami najednou, což v případě velké fluktuace majetku velice prodlužuje jakoukoli činnost. Systém je navíc v nejvyšší verzi licence omezen na 3800 záznamů a 25 uživatelů.

2.3.2 Onesoft Connect

Obrázek 2, Onesoft logo, zdroj: https://www.onesft.com/cs

„Aplikace slouží jako jednoduchá evidence firemního majetku, který si můžete členit do jednotlivých kategorií majetku a přiřazovat k zaměstnancům, kteří ho mají ve správě.“ (Onesoft Connect Inc., 2020).

Řešení od Onesoft Connect přináší nejen evidenci majetku, ale i její údržbu, přehled o zaměstnancích dodavatelích, kontaktech, smlouvách i evidenci elektronické pošty. Jedná se taktéž o výhradně online řešení, takže veškerá data jsou uložena na cloudových serverech externí společnosti.

Majetek se zde vkládá do předpřipravených kategorií a není možnost si zde vytvořit kategorii vlastní, je zde ovšem možné si ke každé položce definovat vlastní informace, kde se zadá název a typ buňky jako text, číslo, datum a jiné.

Základní vyhledávací filtr hledá pouze v názvech majetku. Pokud chceme hledat majetek s konkrétním výrobním číslem, musíme použít detailní filtr.

Aplikace umí taktéž provádět hromadné změny, ale ne u přidělení majetku uživateli. Je možné v ní mít i více majetkových správců, ale není již možné jim přiřadit konkrétní skupinu majetku či uživatelů. Majetkový správci zde operují nad veškerým majetkem a neumožňuje přiřazení konkrétních osob pod určité správce.

2.3.3 Evidei

Obrázek 3, logo Evidei, zdroj: https://evidei.com/evidei-evidencia-majetku/

„Tento typ aplikace je určen pro pracovní stanice s operačním systémem Windows. Je hlavním rozhraním pro správu a evidenci majetku. Umožňuje vkládat a upravovat majetek, sledovat jeho různé stavy, umístění a klasifikaci. Registruje historii a všechny změny, půjčky majetku třetím stranám nebo evidenci cizího majetku. K dispozici je též sledování příslušenství, dodavatelů, pořizovacích cen, sériových čísel, nechybí

fotografie a další přílohy. Údaje jsou bezpečně ukládané na Microsoft SQL Server. " (Evidei, 2020).

Jedná se o desktopovou aplikaci určenou pro operační systémy Windows, pro přístup k ní je tedy vyžadována instalace na každé stanici, ze které chcete k systému přistupovat.

Není vyžadován přístup na internet, a je možné tedy systém provozovat na interní síti společnosti. V případě provozu na interní síti jsou tedy data uložena u objednatele. Pokud by aplikace byla provozována online, umožňuje uložení dat u poskytovatele.

Ukládání majetku do stromové struktury je zde řešeno obdobně jako u aplikace SW Klid a to především pomocí umístění (lokalita, oddělení, místnost).

Systém má omezení dat, resp. karet majetku dle zakoupené licence. Ale počet uživatelů v rámci multilicence nikoli.

Umožňuje svěřovat osobám majetek do užívání i nahlížení uživatelům na něj. Neumožňuje tvorbu více správců majetku, tak aby každý správce odpovídal za majetek a uživatele jemu svěřené.

Aplikace umožňuje hromadné úpravy poznámek, umístění, změny osob, které je majetek přidělen, uložení sestav pro pozdější využití. Také je možné k majetku přidávat vlastní soubory (obrázky, smlouvy a další).

Výše uvedené informace o systému byly získány dle dotazníku, který byl vyplněn zástupcem firmy poskytující software, viz. 7.2.

2.3.4 Fusio

Obrázek 4, Ants Fusio, zdroj: https://www.fusio.cz/

Systém evidence Fusio je také výhradně desktopová aplikace. Takže veškerý přístup k ní je třeba řešit pomocí instalace aplikace na pracovní stanici.

Data jsou uchovávána u objednatele a je možné ji provozovat na interní síti společnosti.

Majetek zde může být řazen pomocí stromové struktury, ovšem jedná se o individuální nastavení systému pro konkrétní realizaci. Umožňuje vyhledávání majetku pomocí omezení nadřazeného ve stromové struktuře.

Počet uživatelů ani počet majetku zde není limitován. Systém umožňuje přiřadit majetek osobám do užívání i náhled na majetek jim přiřazený.

V systému je možné mít více správců majetku, kteří mohou operovat pouze nad majetkem a osobami jim přidělených.

K majetku je možné nahrávat vlastní soubory jako jsou obrázky, smlouvy a jiné. V individuální realizaci je možné přidat i hromadné úpravy umístění, poznámky, změny přidělené osoby a odstranění či vyřazení majetku. Není možné ukládat sestavy pro pozdější využití.

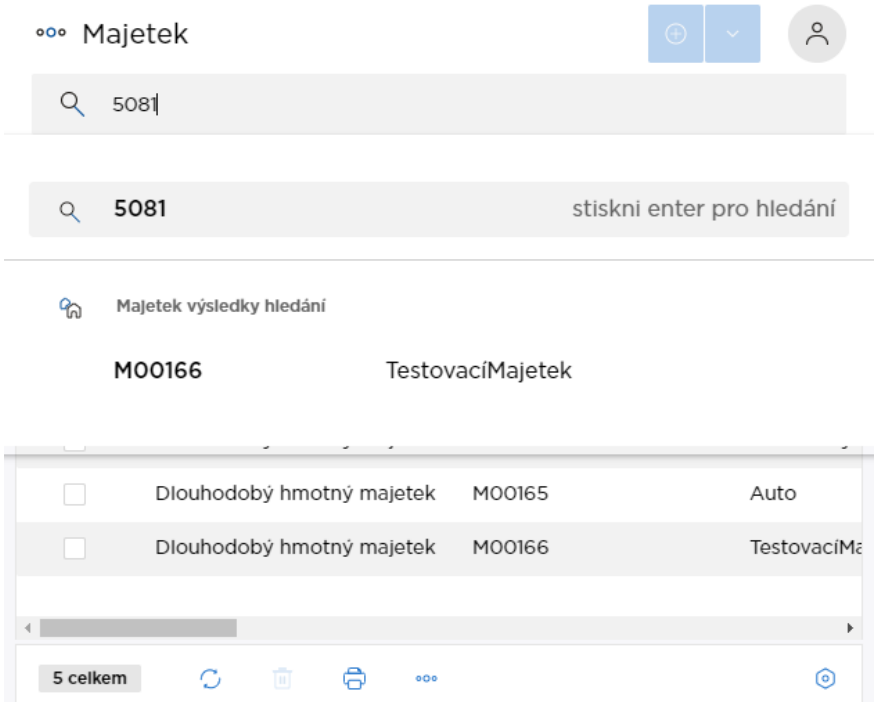
Výše uvedené informace o systému byly získány dle dotazníku, který byl vyplněn zástupcem firmy poskytující software, viz. 7.2.

2.3.5 ABRA Flexi

Obrázek 5, ABRA, zdroj: https://www.abra.eu/

Společnost ABRA Software a.s. nabízí software, který je možné provozovat jak v cloudu, tak u objednatele. Správa majetku je zde jako jeden z mnoha modulů, které je možné využít. Nabízí možnost ovládání pomocí nainstalovaného klienta či pomocí webového rozhraní. Webové rozhraní, ale neobsahuje tolik funkcí jako nainstalovaný klient, např. není v něm možné přidávat majetek do evidence a ani jej editovat i přesto, že se v klientu nachází tlačítko editace, veškeré položky jsou zakázané pro editaci.

Při vyhledávání majetku pomocí vyhledávacího okna zobrazuje majetek, který obsahuje vyhledávaný výraz, ovšem již neznázorňuje, v jaké položce byl výsledek nalezen, viz. Obrázek 6



Obrázek 6, ABRA - vyhledávání majetku, vlastní tvorba

Při stisku tlačítka „enter“ nabídne seznam majetku, ve kterém hledaný výraz nalezl. Software ani v tomto seznamu neznázorní vyhledaný výraz. To lze obejít funkcí prohlížeče pro vyhledání výrazu (ctrl+f), aby uživatel viděl, kde byl výraz vyhledán. Vyhledávání zde bere v potaz diakritiku, takže pokud by bylo do pole pro vyhledávání zadáno: „testovac**i**“ a ne „testovac**í**“, tak by majetek nebyl nalezen.

Kategorizace majetku je tvořena pomocí definovaných kategorií, které není možné do sebe zanořovat a jsou všechny na stejné úrovni.

Přiřazení majetku odpovědné osobě je možné pouze přes instalovaného klienta a není možné jej dohledat pomocí vyhledávacího formuláře. Nahrávání souborů k položce je možné.

2.3.6 Závěr k průzkumu dostupných aplikací

V dnešní době je stále málo software pro správu majetku, který by umožňoval provoz aplikace u objednatele a pokud ano, tak většinou s instalovaným klientem. Kategorizace majetku je převážně jednoúrovňová nebo daná již předdefinovanými kategoriemi.

Vyhledávání v majetku je převážně řešeno filtry, jejich nastavení zabere častokrát spoustu času a pokud není výsledek vidět již při nastavování, tak uživatel často neví, do doby potvrzení nastavení všech filtrů, výsledek svého hledání.

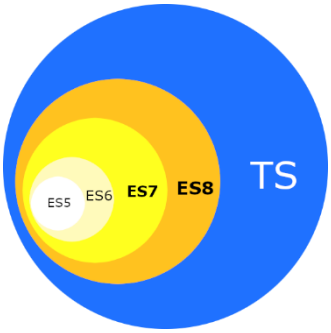
V případě, že má společnost méně položek nebo malý pohyb, tak jsou aplikace v oblasti evidence majetku vyhovující, což je i ovlivněno potřebami většiny potenciálních zákazníků. Ovšem pokud by měl uživatel denně provádět více úkonů, tak začne v aplikacích narážet na zdlouhavé vyhledávání majetku pomocí nastavování filtrů metodou „pokus-omyl“, často pro jeden účel. Dohledání majetku v případě nejasného identifikátoru by při vysokém počtu položek bylo velice obtížné a zdlouhavé.

3 Technologie použité pro tvorbu aplikace

V této kapitole jsou představeny použité technologie a jejich hlavní části pro tvorbu aplikace a knihovny implementované v projektu.

3.1 TypeScript

TypeScript je programovací jazyk a balíček nástrojů pro generování JavaScriptu zároveň. Navrhl jej Andres Hejlsberg ze společnosti Microsoft a jedná se o open-source projekt, který má pomáhat vývojářům v psaní kódu v týmech.



Obrázek 7, TypeScript, vlastní tvorba

Standardizovaný JavaScript se označuje jako ECMAScript a číslo označující edici. (Rozentals Nathan, 2017).

TypeScript přináší do vývoje JavaScriptových aplikací typovou kontrolu. Dále využívá tzv. „dekorátorů“, což je velmi silný nástroj, jak přidat metadata k deklaraci třídy. Vytvořením dekorátoru se definuje speciální anotace, která definuje způsob, jakým se třídy, metody či funkce chovají. (Deeleman Pablo, 2016)

3.2 Angular

Obrázek 8, Angular logo, zdroj: https://angular.io/

Angular je jeden z předních frameworků pro tvorbu tzv. „single page apps“ (dále jen SPA). Hlavní rozdíl mezi SPA a standardní webové stránce je, že pokud uživatel klikne na odkaz, tak se načte celá nová stránka, kdežto u SPA se načte pouze požadovaný obsah. (Lim Greg, 2017).

3.2.1 Obecná struktura projektu

Každý projekt obsahuje aspoň jeden hlavní modul. Ten může obsahovat další moduly. U menších projektů postačí pouze hlavní modul.

V modulech se nachází komponenty, služby, které zajišťují sdílení dat a operace napříč aplikací či zajišťují komunikaci se server-side aplikací a další potřebné třídy k chodu daného modulu.

3.2.2 Hlavní prvky frameworku Angular

V této sekci jsou uvedeny „stavební kameny“ frameworku. Nejsou zde uvedeny třídy či rozhraní, které reprezentují objekty. Obecně se v Angularu využívají s příznakem „.model.ts“, ale toto je věc čistě individuální.

3.2.2.1 Moduly

Moduly jsou třídy, které obsahují @NgModule dekorátor. Ten obsahuje objekt s několika parametry viz. Obrázek 9. Tyto parametry jsou dále rozebrány v jednotlivých sekcích.

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Obrázek 9, ngModule, vlastní tvorba

declarations

Vlastnost declarations definuje pole komponent, direktiv a tzv. pipes, které náleží modulu. Každá deklarovaná třída musí být obsažena pouze v jednom modulu napříč celým projektem.

imports

Vlastnost imports obsahuje kolekci importovaných modulů. Tímto způsobem je možné využít v konkrétním modulu exportované prvky z modulu importovaného.

providers

Vlastnost providers je kolekce objektů, které je možné využívat napříč modulem, jedná se převážně o služby, které zajišťují výměnu dat mezi komponentami a zajišťují funkční logiku aplikace.

bootstrap

Vlastnost bootstrap definuje pole tzv. „vstupních komponent“, které se načtou imperativně ihned při načtení modulu a není možné je vložit pomocí reference v jiné šabloně komponenty. (Google, 2010-2020a)

3.2.2.2 Komponenty

Komponenty jsou hlavním stavebním kamenem Angularu. Jsou to třídy označené pomocí @Component dekorátoru, který obsahuje opět objekt s parametry.

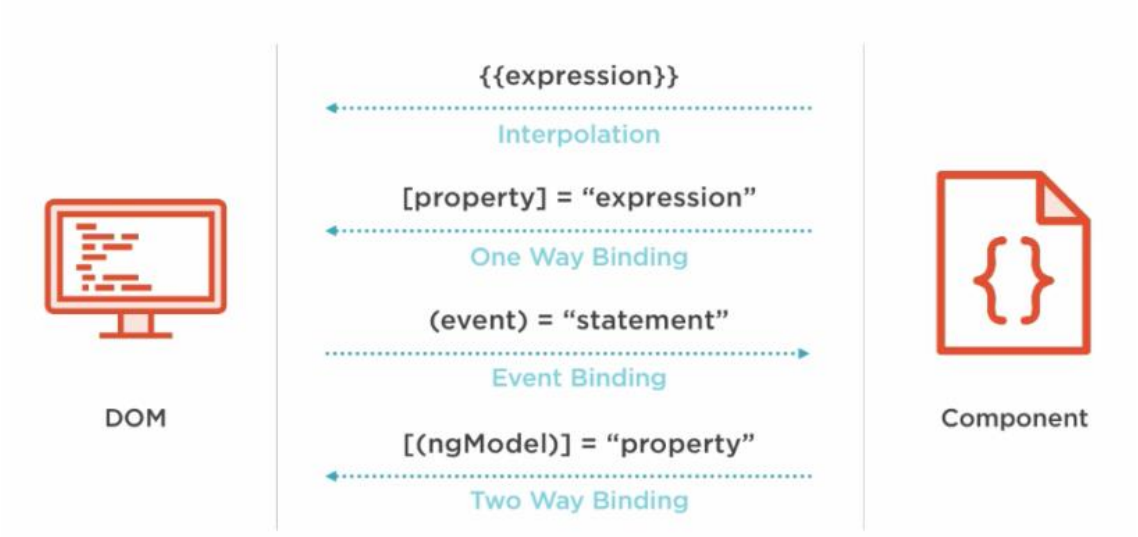
Nejčastěji se v tomto objektu nachází vlastnost ‚selector‘, která určuje značku, pomocí které se umísťuje komponenta do šablony.

Následuje vlastnost „template“ či „templateUrl“. V prvním případě se zapisuje html šablona přímo jako hodnota. Pokud je uvedeno „templateUrl“, tak je šablona načtena z url, která je v hodnotě vlastnosti.

Poslední typická vlastnost je styleUrls, případně styles, která umožňuje připojit ke komponentě kaskádové styly. Analogicky jako v předchozím případě, buď přímo jako hodnota, či soubor na uvedené url. Zde je možné využívat i preprocesory, záleží vždy na nastavení aplikace.

Sdílení dat mezi šablonou a funkční vrstvou komponenty

V anglickém názvosloví se jedná o „property bindings“. Angular využívá čtyř typů sdílení dat viz. Obrázek 10



Obrázek 10, Property Bindings, Hoang Ha, zdroj: https://hahoangv.wordpress.com/2016/05/30/display-data-in-angular-2/

Prvním typem je interpolace. Ta slouží čistě pro zobrazení dat. V šabloně se označuje dvěma za sebou jdoucími znaky složených závorek. Příklad použití viz. Obrázek 11 a Obrázek 12, atribut „title“. Atribut title bude vytvořen na elementu i v případě, že hodnota v titleText nebude zadána.

Další typ, který slouží k jednostrannému sdílení dat z funkční vrstvy do šablony je tzv. „One Way Binding“. Kde se vlastnost vyváří a nastavuje v případě, že existuje ve funkční vrstvě. Jak je vidět na Obrázek 11 v atributu „src“, ten se vytvoří a nastaví až v případě, že hodnota imagePath existuje.

```
<img [src]="imagePath" title="{{titleText}}" (click)="enlarge()">
<input type="text" [(ngModel)]="titleText">
```

Obrázek 11, Property Bindings - šablona, vlastní tvorba

```
export class AppComponent {
  title = 'empty';
  titleText: any;
  imagePath: any;

  enlarge(): void {
    // zvetseni
  }
}
```

Obrázek 12, Property Bindings - komponenta, vlastní tvorba

Opačným typem sdílení je tzv. „Event binding“, je označen kulatými závorkami a slouží pro vyvolání akce na základě události. Ve výše uvedeném příkladu slouží k zavolání metody „enlarge“ při kliku na element.

Posledním typem je oboustranné sídlení, které je označeno hranatými a kulatými závorkami zároveň. Slouží pro oboustranné sdílení dat. Ve výše uvedeném příkladě, pokud začneme psát text do textového pole, tak se bude měnit zároveň i atribut „title“, který má interpolovanou hodnotu nastavované proměnné.

Veškeré sdílení informací jde aplikovat i mezi komponentami, pokud tedy existuje v šabloně jedné komponenty, komponenta druhá, mohou mezi sebou komunikovat pomocí tohoto sdílení informací.

3.2.2.3 Služby

Služby jsou speciální třídy, které jsou označeny jako @Injectable. Pomocí vlastnosti provideIn s hodnotou root se zajistí, že takto označená třída bude tzv. „singleton“ (pouze jedna instance této třídy v rámci aplikace).

Služby slouží především pro sdílení informací mezi komponentami, či moduly a pro komunikaci s externími zdroji. Pro přístup ke službě je třeba službu vložit do konstruktoru třídy.

3.2.2.4 Pipes

Pipes jsou nástroje, které umožňují přetvářet řetězce, např. formát data či pro čísla na řetězec s počtem desetinných míst v dané zemi. V Angularu existují některé předdefinované pipes, které slouží k nejběžnějším transformacím. Jsou to: DatePipe, UpperCasePipe, LowerCasePipe, CurrencyPipe, DecimalPipe, PercentPipe. Použití pipes se vyvolá znakem „|“ a názvem použité pipy.

```
<nb-toggle [(checked)]="right.selected" (click)="changeRights(right)"
           *ngFor="let right of rights |
           filter: category.relatedTo: propName: 'relatedTo' ">
  {{right.name}}
</nb-toggle>
```

Obrázek 13, Použití filter pipe, vlastní tvorba

Pipes je možné i definovat vlastní. Typickým příkladem je filter pipes, uvedená na Obrázek 14.

```
@Pipe({
  name: 'filter'
})

export class FilterPipe implements PipeTransform {

  transform(value: any, filterString: string, propName: string): any {
    if (value.length === 0) {
      return value;
    }
    const resultArray = [];
    for (const item of value) {
      if (item[propName] === filterString) {
        resultArray.push(item);
      }
    }
    return resultArray;
  }
}
```

Obrázek 14, Filter pipe, vlastní tvorba

3.2.2.5 Direktivy

Direktivy jsou speciální prvky, které ovlivňují a upravují elementy. Zapisují se uvnitř html elementu a dělíme je do dvou skupin a to buď „Attribute directive“ nebo „Structural directive“.

Attribute directives

Atributové direktivy mění vzhled či funkční stránku DOM elementu, na kterém jsou umístěny. Je možné aplikovat více těchto direktiv na jeden DOM element. Vestavěné atribut direktivy jsou např. ngStyle a ngClass.

```
<div fxFlex fxLayout="column" fxLayoutGap="1em"
     [ngClass]="{'editMode': !editMode}">
  <div fxLayout="row" fxLayoutAlign="start center" fxLayoutGap="1em">
    <label for="name">Jméno</label>
    <input nbInput id="name" value="Milan" formControlName="name">
```

Obrázek 15, ngClass direktiva, vlastní tvorba

Na Obrázek 15, ngClass direktiva, vlastní tvorba je vidět ukázka použití direktivy ngClass. V případě, že je nastavena hodnota editMode na pravdu, tak je „divu“ přirazena třída „editMode“, v opačném případě se atribut „class“ na divu vůbec nevyskytuje.

Structural directives

Strukturální direktivy se používají pro manipulaci s DOM strukturou, především umožňují prvky přidávat či odebírat. Stejně jako ostatní direktivy jsou aplikovány na element, který modifikují. Není složité je poznat, jelikož začínají na „*“ viz. Obrázek 16. Mezi nejběžnější zabudované strukturální direktivy patří: ngIf, ngFor, ngSwitch. (Google, 2010-2020b)

```
<nb-option *ngFor="let unit of units"
           [value]="unit">
  {{unit.name}}
</nb-option>
```

Obrázek 16, ngFor direktiva, vlastní tvorba

3.2.3 Využité knihovny v projektu

V této kapitole jsou představeny knihovny pro framework Angular využité v rámci aplikace.

3.2.3.1 Nebular

Nebular je modulární knihovna obsahující komponenty pro tvorbu uživatelského rozhraní pro framework Angular.

Je založena na Eva Design Systému. Tím je umožněno jednoduše tvořit vlastní barevná témata pomocí generátoru na webu, zároveň i nativně využívá sadu ikon z tohoto systému.

3.2.3.2 ag-Grid

Ag-Grid je zástupce knihovny, která zobrazuje data pomocí tabulkového systému. Je inspirována programy jako jsou Microsoft Excel, Google Sheets, LibreOffice a další jím podobné. Známější zástupce této kategorie je např. handsontable či datatables.

Tato knihovna je rozšířená nejen pro Angular, ale i další frameworky (React a Vue), nemá žádné závislosti a je možné ji využít i v projektu za použití pouze JavaScriptu.

Tato knihovna byla volena, jelikož umí dobře zobrazovat stromovou strukturu spojenou s tabulkovým výpisem a umožňuje využívat vlastní komponenty v rámci knihovny. Např. pro vlastní vykreslení obsahu buňky, filtry postranního panelu a spoustu dalších.

3.3 NestJS

Obrázek 17, NestJS logo, zdroj: https://nestjs.com/

NestJS je framework využívající platformu node.js pro tvoření „server-side“ aplikací a stejně jako Angular, tak i NestJS využívá TypeScript.

3.3.1 Obecná struktura projektu

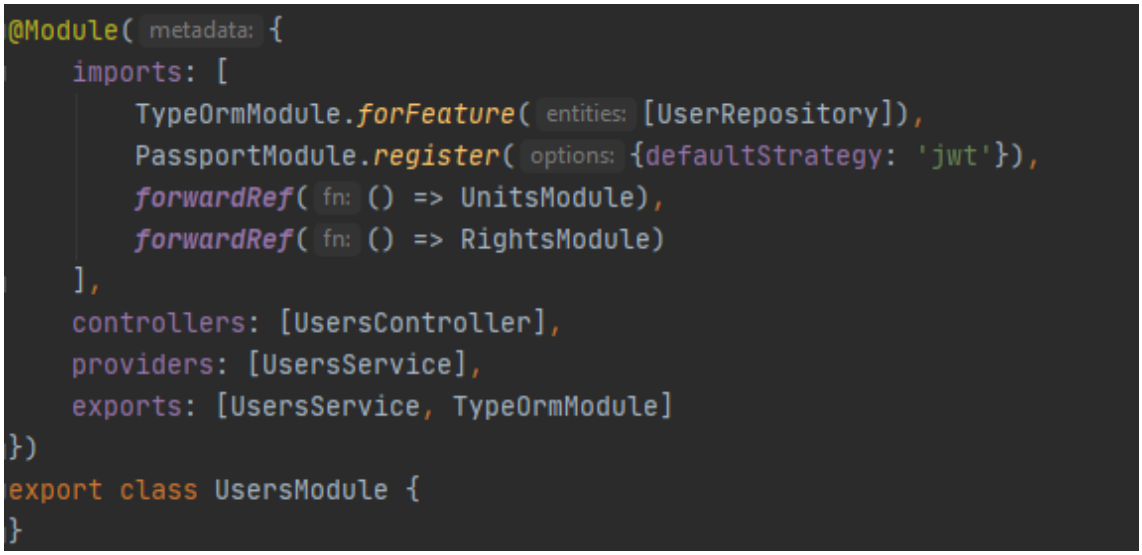
Struktura projektu je téměř totožná projektu vytvořeném ve frameworku Angular. Je rozdělená do několika modulů, které obsahují prvky k sobě vztažené. Jako hlavní stavební kameny nejsou v NestJS komponenty, ale controllery.

3.3.2 Hlavní prvky frameworku NestJS

V této kapitole jsou popsány hlavní prvky frameworku NestJS.

3.3.2.1 Moduly

Moduly jsou třídy s dekorátorem @Module, který obsahuje opět objekt s vlastnostmi: imports, providers, exports. Vlastnosti providers a exports jsou totožné s vlastnostmi modulu ve frameworku Angular, proto zde nejsou zmíněny. V objektu se již nevyskytují deklarace, ale místo této vlastnosti je zde vlastnost controllers.



Obrázek 18, NestJS modul, vlastní tvorba

controllers

Vlastnost controllers obsahuje pole kontrolerů, jenž jsou součástí modulu.

imports

Vlasnost imports zahrnujte importy modulů pro chod modulu. Na Obrázek 18, je vidět využití reference v NestJS, která odkazuje na ještě nedefinovanou třídu, (Mysliwiec, 2017-2020a) čímž je možné zajistit využití modulu, který vyžaduje pro svoji funkčnost modul, který jej importuje. V tomto případě se jedná o ukázku modulu „users.module.ts“, který ke své funkčnosti potřebuje, mimo jiné, moduly: „UnitsModule“ a „RightsModule“. Tyto moduly ke své funkčnosti potřebují ale i modul: „UsersModule“ a proto je třeba využít importu tímto způsobem.

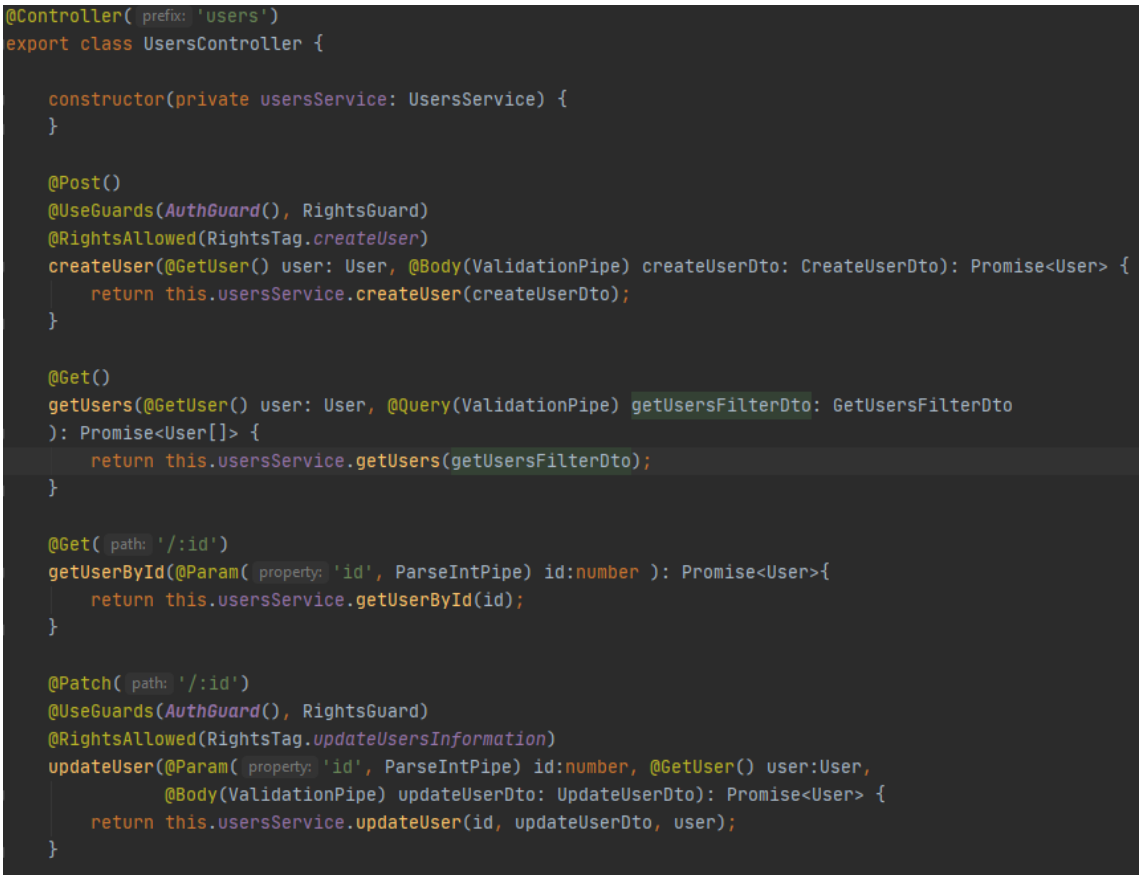
Pokud by nebylo potřeba využití modulů vzájemně, bylo by možné zapsat import bez funkce forwardRef(), ale byl by zde uveden pouze název modulu.

3.3.2.2 Controllers

Jako jsou hlavním stavebním prvkem v Angularu komponenty, tak u NestJS to jsou Controllers. Jedná se o třídy obsahující dekorátor @Controller, který zpravidla obsahuje string, definující společnou cestu pro metody uvnitř třídy. V ukázce

(**Chyba! Nenalezen zdroj odkazů.**) je znázorněna část Controlleru, který je pro c estu „/users“, jak je patrné z řetězce uvnitř @Controller. Metody uvnitř Controlleru jsou zpravidla předcházeny různými dekorátory, zejména pak těmi, které určují metodu http:

- @Post() – pro vložení záznamu
- @Put(), @Patch() – pro změnu dat
- @Get() – pro získání dat
- @Delete() – pro smazání dat



```
@Controller( prefix: 'users')
export class UsersController {

    constructor(private usersService: UsersService) {
    }

    @Post()
    @UseGuards(AuthGuard(), RightsGuard)
    @RightsAllowed(RightsTag.createUser)
    createUser(@GetUser() user: User, @Body(ValidationPipe) createUserDto: CreateUserDto): Promise<User> {
        return this.usersService.createUser(createUserDto);
    }

    @Get()
    getUsers(@GetUser() user: User, @Query(ValidationPipe) getUsersFilterDto: GetUsersFilterDto
    ): Promise<User[]> {
        return this.usersService.getUsers(getUsersFilterDto);
    }

    @Get( path: '/:id')
    getUserById(@Param( property: 'id', ParseIntPipe) id:number ): Promise<User>{
        return this.usersService.getUserById(id);
    }

    @Patch( path: '/:id')
    @UseGuards(AuthGuard(), RightsGuard)
    @RightsAllowed(RightsTag.updateUsersInformation)
    updateUser(@Param( property: 'id', ParseIntPipe) id:number, @GetUser() user:User,
               @Body(ValidationPipe) updateUserDto: UpdateUserDto): Promise<User> {
        return this.usersService.updateUser(id, updateUserDto, user);
    }
```

Obrázek 19, Ukázka Controlleru, vlastní tvorba

Uvnitř dekorátorů definujících http metody se může také nacházet další string, který nastavuje další část url na parametr s názvem za dvojtečkou. Ve výše uvedeném případě se jedná např. o metodu: Get(‘/:id‘). Pokud je v dekorátoru tento zápis, definuje cestu ,/users/hodnotaParametru“ a při volání této cesty je možné

v metodě, uvedené pod dekorátorem, k hodnotě přistoupit, přes dekorátor @Param, kde na prvním místě definujeme název parametru (*id*), který mohou a zpravidla následují jedna či více pipes (*ParseIntPipe*), jenž hodnotu z parametru převedou, či kontrolují. Jelikož „hodnotaParametru“ je typu string, existují v NestJS pipes, které výstup přetransformují.

3.3.2.3 Služby

Stejně jako v případě frameworku Angular, jsou zde služby třídy označené dekorátorem Injectable. Pokud není uveden objekt v decorátoru obsahující vlastnost scope (analogie s provideIn vlastností v Angularu), je vytvořena pouze jedna instance třídy napříč aplikací.

3.3.2.4 Pipes

NestJS obsahuje šest zabudovaných pipes, které vstup transformují či kontrolují. Z transformačních to jsou ParseIntPipe, ParseBoolPipe, ParseArrayPipe, ParseUUIDPipe.

K nastavení výchozí hodnoty slouží DefaultValuePipe a poslední pipe je ValidationPipe, která validuje objekt na základě definovaného typu, tzv. DTO („Data transfer object“).

Pipes je možné definovat i vlastní. Jedná se o třídy s dekorátorem @Injectable a musí implementovat rozhraní PipeTransform. V níže uvedeném příkladě (Obrázek 20) se jedná o pipe převádějící celý řetězec na velká písmena.

```
import {ArgumentMetadata, Injectable, PipeTransform} from '@nestjs/common';

@Injectable()
export class UpperCasePipe implements PipeTransform {
    transform(value: any, metadata: ArgumentMetadata): any {
        return value.toUpperCase();
    }
}
```

Obrázek 20, UpperCasePipe, vlastní tvorba

3.3.2.5 Guards

Jedná se o třídu s dekorátorem @Injectable a implementací CanActivate rozhraní. Tyto třídy mají pouze jedinou zodpovědnost, a to rozhodnout, zda je možné přistoupit k danému přístupovému bodu na základě různých podmínek (např., práv, rolí) či nikoli (Mysliwiec, 2017-2020b).

3.3.2.6 Dekorátory

NestJS nabízí několik dekorátorů, které je možné použít a které reprezentují prosté Express či Fastify objekty, např. @Param(param?: string) je req.params, potažmo req.params[param] (Mysliwiec, 2017-2020c).

3.3.3 Využité knihovny v projektu

Tato kapitola obsahuje základní informace o knihovnách využitých v rámci serverové aplikace.

3.3.3.1 JSON Web Tokens

„*JSON Web Token (dále jen JWT) je otevřený standard (RFC 7519), který definuje bezpečný způsob přenosu informací jako JSON objekt. Tato data je možné ověřit a lze jim důvěřovat, jelikož jsou digitálně podepsána*.“ (Auth0, 2013).

Tato knihovna je využita pro vytváření tokenů a validaci tokenů na straně serveru (NestJS), ale i na straně klienta (Angular) pro extrakci informací z vytvořených tokenů.

3.3.3.2 Swagger

Swagger je velmi silný nástroj pro tvorbu a vývoj API dokumentace. Po přidání do aplikace, knihovna zobrazuje end pointy, včetně požadovaných argumentů a návratových hodnot. Zároveň poskytuje i tzv. „playground“, kde si vývojář může zkoušet dotazy na server a vidět návratové hodnoty.

4 Návrh a implementace

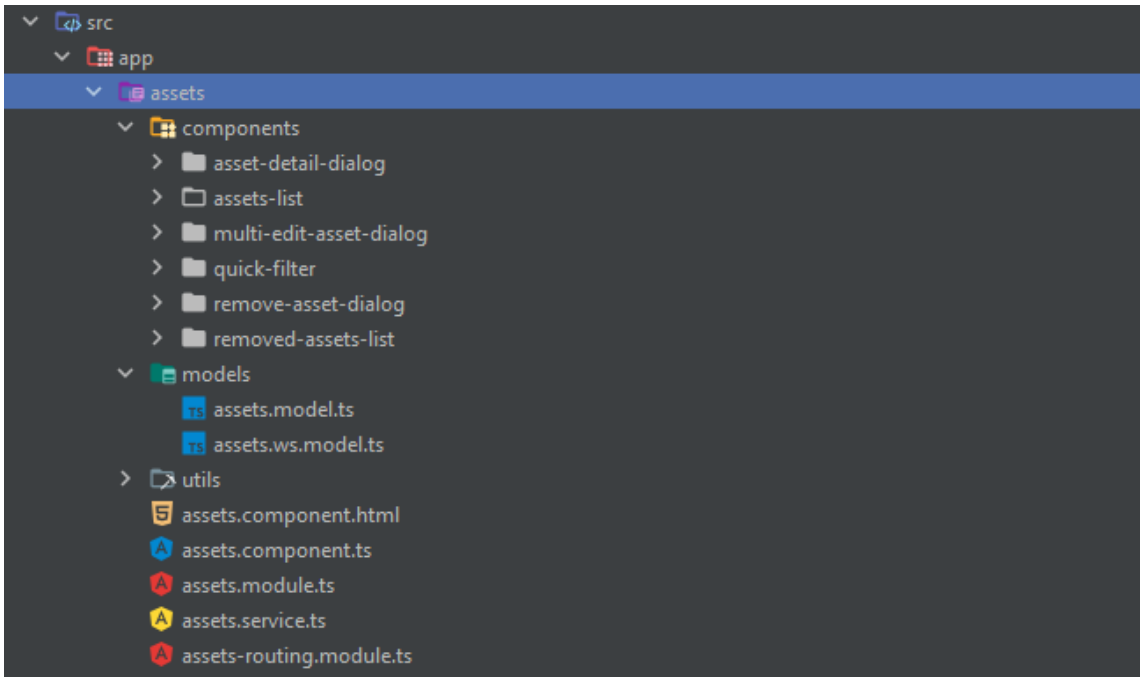
Kapitola obsahuje návrh entit, resp. datových struktur, samotné aplikace a její implementace.

4.1 Struktura projektů

V této kapitole jsou popsány implementace Angularu a NestJS. Jsou zde vysvětleny principy a myšlenky projektu. Jejich implementace, odhalené nedostatky a zmíněny oblasti kudy projekt dále rozvíjet.

4.1.1 Frontend – Angular

Struktura projektu se drží obecné struktury projektu popsané v bodě 3.2.1 (Obecná struktura projektu Angular). V kořenové složce projektu se nachází app.module.ts ve kterém jsou importovány ostatní moduly. Na Obrázek 21 je zobrazen jeden modul na kterém budou vysvětleny principy.



Obrázek 21, modul assets, vlastní tvorba

Modul obsahuje složky: components, models a utils. Veškeré komponenty náležící k modulu se nacházejí ve složce components.

4.1.1.1 Rozhraní

Ve složce models se nacházejí enumy a rozhraní, které se v TypeScriptu chovají odlišněji než např. v Javě. V Javě rozhraní popisuje metody, jejich vstupy a výstupy, kdežto v TypeScriptu pomocí rozhraní definujeme vzhled objektu. Na Obrázek 22 je ukázka, jak rozhraní vypadá. Rozhraní **IAssetUsetExt**, definuje vzhled objektu, který obsahuje vlastnosti: id, typu number, name a surname, oboje typu string, reachable, typu boolean a unit, která je typu jiného rozhraní.

```
export interface IAssetUserExt {
  id: number;
  name: string;
  surname: string;
  reachable: boolean;
  unit: Unit;
}

export enum AssetNoteSetTypeEnum {
  start = 0,
  end = 1,
  replace = 2
}

export enum AssetChangeEnum {
  assetUser = 'assetUser',
  assetSerialNumber = 'assetSerialNumber',
  assetInventoryNumber = 'assetInventoryNumber',
  assetEvidenceNumber = 'assetEvidenceNumber',
  assetIdentificationNumber = 'assetIdentificationNumber',
  assetName = 'assetName',
  assetNote = 'assetNote'
}
```

Obrázek 22, ukázka rozhraní a enums, vlastní tvorba

Dále můžeme na Obrázek 21 vidět použití dvou enumerátorů. Jeden je definován s číselným předpisem a druhý s řetězcovým. Hlavní rozdíl v těchto typech je ten,

že enumerátor s číselným předpisem je možné použít na templates (html část komponenty), kdežto s řetězcovým předpisem nikoli.

V souboru s ws v názvu, je enumerátor (Obrázek 23) definující operace websocketu.

```
export enum AssetsWs {
  assetsUpdate = 'assetsUpdate',
}
```

Obrázek 23, enumerátor definující operaci pro ws, vlastní tvorba

Ve složce **utils** se poté nacházejí pomocné funkce, či komponenty, které jsou využity prostřednictvím komponent.

4.1.1.2 Routing

Soubor **assets-routing.module.ts**, obsahuje definici cest v rámci modulu a zobrazení jednotlivých komponent.

```
const routes: Routes = [
  {path: '', component: ListsComponent, children: [
      {path: '', component: ListsListComponent},
      {path: 'working-list', component: WorkingListComponent,
        data: {loadWorkingList: true}},
      {path: 'list/:id', component: ListDetailComponent},
    ]},
];

@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})

export class ListsRoutingModule {
}
```

Obrázek 23, routing, vlastní tvorba

Na Obrázek 23 je vyobrazen routing pro sestavy. Je zde import a export cest, kde v importech se využívá metoda „forChild“ pokud se jedná o lazyLoaded modul. Samotné nastavení cest je pak pomocí konstanty **routes**, která je typu Routes.

Obsahuje pole definovaných cest. V tomto případě, pokud je načtena cesta pro modul sestav (lists), tak se zobrazí ListComponenta, která obsahuje router-outlet a pokud není další parametr v cestě, tak se načte seznam sestav (ListsListComponent), pokud cesta obsahuje ještě „working-list“ zobrazí se pracovní/dočasná sestava a pokud je adresa ve tvaru: „list/:id“, zobrazí se detail sestavy (ListDetailComponent), kde „:id“ značí parametr identifikátoru sestavy.

Pro porovnání je na Obrázek 24 zobrazen app-routing.module.ts, kde je vidět použití metody forRoot() a u cest metoda loadChildren(), pro lazyLoading daného modulu.

```
const routes: Routes = [
  {path: '', pathMatch: 'full', component: HowToComponent},
  {
    path: 'units',
    loadChildren: () => import ('./units/units.module').then(m => m.UnitsModule)
  },
  {
    path: 'lists',
    loadChildren: () => import('./lists/lists.module').then(m => m.ListsModule)
  },
  {path: 'categories'...},
  {path: 'users'...},
  {path: 'assets'...},
  {path: 'locations'...},
  {path: 'history'...},
  {path: 'protocols'...},
  {path: '**', component: PageNotFoundComponent}
];
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule {
}
```

Obrázek 24, app-routing.module.ts, vlastní tvorba

Dále je zde pak cesta definovaná s hodnotou: „**“, ta slouží pro veškeré nerozpoznané cesty a zobrazí komponentu PageNotFounComponent.

4.1.1.3 Služby

V každém modulu se pak nachází service (služba), která se stará o manipulaci a sdílení dat napříč komponentami. Entity jako jsou majetek a uživatelé ukládá do „Behavior subjectu“ a ty pak dále vystavuje formou observables pro ostatní komponenty. Většina těchto service jsou s nastavením provideIn: „root“. Tím je docíleno, že je pouze jedna instance použita napříč celým projektem. Úskalím může být, pokud bude služba uvedena v poli providers v modulu. Pokud nastane tato situace, tak se vytvoří dvě instance této služby a záleží na čase vytvoření ostatních služeb a komponent ke které instanci služby se přihlásí. Toto může způsobit dlouhé odhalování logické chyby, jelikož kompilátor na toto neupozorní.

V projektu je použita i služba, u které je žádoucí, aby byla vytvořena ve více instancích.

```
@Injectable({
  providedIn: 'any'
})
export class AgGridService {
  private showOnlySelectedRows$: BehaviorSubject<boolean> = new BehaviorSu
  private fitColumns$: BehaviorSubject<boolean> = new BehaviorSubject<bool
```

Obrázek 25, AgGridService, vlastní tvorba

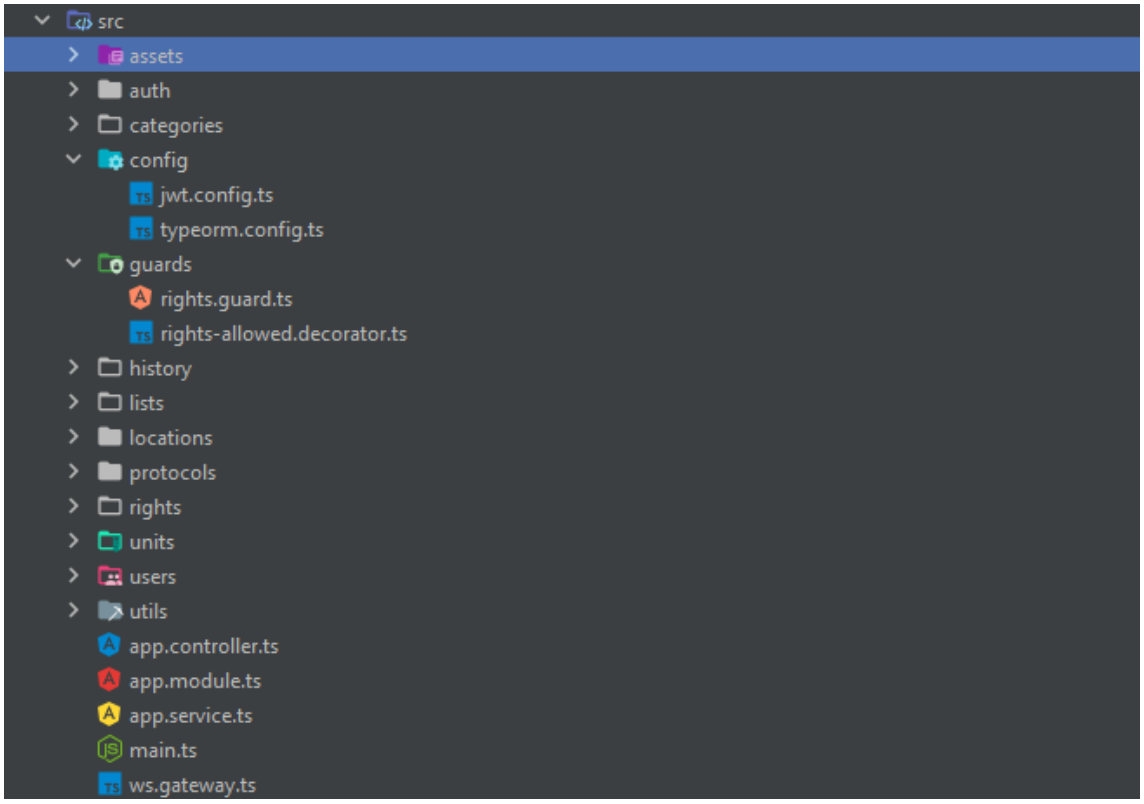
Jedná se o službu, která manipuluje s knihovnou ag-grid a zde je potřeba, aby každá tabulka měla vlastní službu na její ovládání. Jak je vidět na Obrázek 25, je vlastnost provideIn, nastavena na „any“. Tímto nastavením se zajistí, že služba bude spuštěna ve více instancích.

4.1.2 Backend – NestJS

Struktura projektu NestJS vychází ze struktury popsané v bodě 3.3.1 (Obecná struktura projektu NestJS), jak je patrné z Obrázek 26. V kořenové složce jsou složky modulů, které jsou importovány v hlavním modulu app.module.ts. Dále jsou složky config, guards a utils. Složka utils obsahuje podpůrné funkce, které využívají ostatní části napříč moduly.

Ve složce **guards** je definovaný guard a jeho dekorátor, pro hlídání uživatelských oprávnění a ve složce **config**, se nachází nastavení pro tokeny (jwt.config.ts), kde je nastaven secret a doba expirace tokenů.

Dále je zde soubor typeorm.config.ts, v souboru je nastavení pro připojení na databázi, v projektu je využita databáze Postgres. Během vývoje byla testována i změna na MySQL, změna trvala asi 5 minut. Je potřeba doinstalovat ovladač pro připojení k databázi pro TypeORM a změnit některé nekompatibilní sloupce.



Obrázek 26, struktura projektu NestJS, vlastní tvorba

4.1.2.1 Struktura modulu NestJS

Jak je vidět na Obrázek 27, struktura modulu obsahuje složky dto, models a repositories. Ve složce dto (data transfer objects) se nachází předpisy pro příchozí data.

Ve složce models se pak nachází soubory model a entity. Ze souborů „entity“ jsou tvořeny tabulky v databázi.

Složka repositories obsahuje soubory, ve kterých jsou upravené metody pro komunikaci s databází.

Obrázek 27, struktura modulu NestJS, vlastní tvorba

4.1.2.2 Data transfer objects

Data transfer objects jsou třídy, které využívají knihovny class-validator a transforms a definují vzhled příchozích dat, která jsou na základě dekorátorů kontrolována. V případě, že data nesplňují podmínky, je vrácena chyba se zprávou. Na Obrázek 28 je vidět předpis pro data na tvorbu kategorie. Kontroluje se, zda příchozí objekt obsahuje vlastnost name, typu řetězec a zda je jeho délka v rozmezí

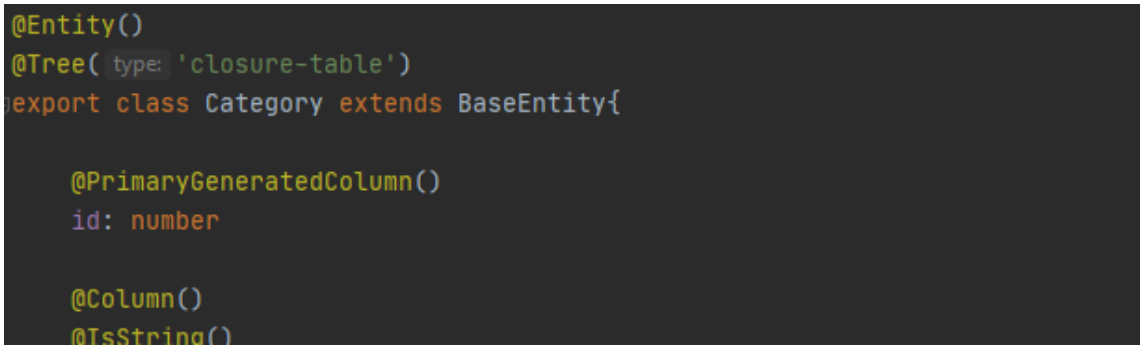
od 1 do 50 znaků. Následuje vlastnost code, která není povinná, ale pokud je, tak musí být typu řetězec s maximální délkou 20 znaků. Poslední vlastnost, co je možné v objektu zadat, je taktéž volitelná položka, a to identifikátor rodiče, pod který se případná kategorie zanoří. Zde je použitá knihovna tranforms, která v případě, že hodnota je ve číslo ve stringu, tak jej převede na typ number.

```
export class CreateCategoryDto {

    @IsString()
    @MinLength( min: 1)
    @MaxLength( max: 50)
    name: string

    @IsOptional()
    @IsString()
    @MaxLength( max: 20)
    code?: string

    @IsOptional()
    @IsNotEmpty()
    @Transform( transformFn: value => Number(value.value))
    @IsInt()
    parent?: number
}
```

Obrázek 28, CreateCategoryDto, vlastní tvorba

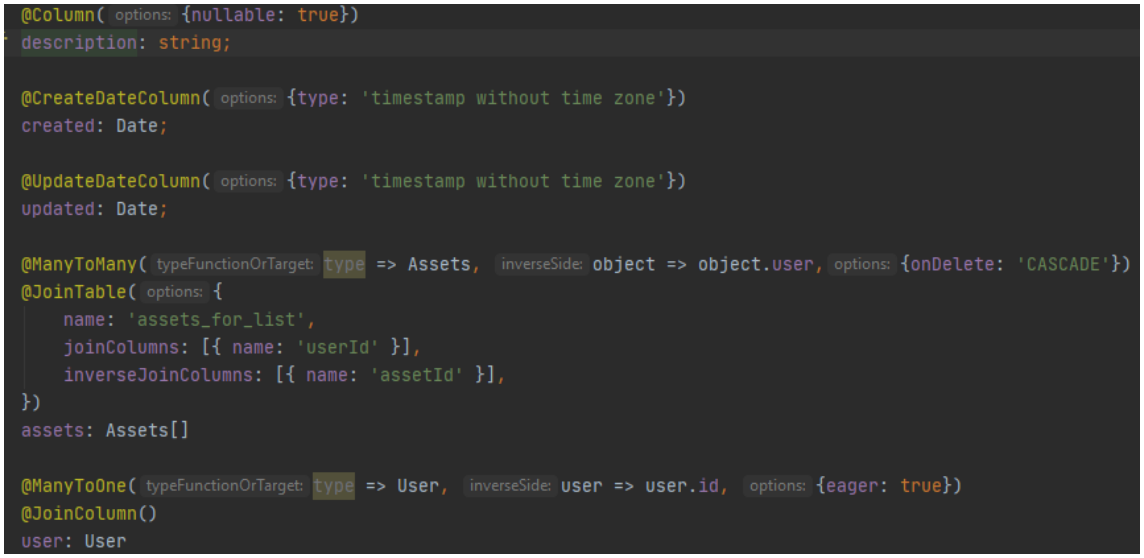
4.1.2.3 Entity

Entity jsou třídy, kterým předchází dekorátor @Entity() a rozšiřují BaseEntity z knihovny TypeORM. Tyto třídy definují vzhled tabulek v databázi pomocí dekorátorů a vlastnostech v nich nastavených. Na Obrázek 29 je ukázka entity Category, která je uložena ve stromové struktuře typu „closure-table“.



Obrázek 29, Entity Category, vlastní tvorba

Pomocí dekorátorů je možné specifikovat nejen sloupce typu „primary generated“, ale i třeba sloupce obsahující datum úpravy, vytvoření, verze či další relační vztahy. Např. pro vztah N:1 slouží dekorátor @ManyToOne(), pro M:N, ManyToMany(). Implementace těchto vztahů je předvedena na Obrázek 30.



Obrázek 30, ListEntity, vlastní tvorba

4.1.2.4 Repositories

Repositories představují vrstvu pro komunikaci s databází pro danou entitu. Aby byly repositories přístupné pro TypeORM, je nutné je zaregistrovat v modulu, viz. Obrázek 31

```
@Module( metadata: {
    imports: [
        TypeOrmModule.forFeature( entities: [CategoriesRepository, CategorySettingsRepository]),
```

Obrázek 31, registrace repositories, vlastní tvorba

V případě nutnosti je možné v nich upravovat či definovat metody. To je užitečné v případě, pokud je potřeba využít složitější dotaz. Na následujícím Obrázek 32, je předefinovaná metoda find().

```
@EntityRepository(Assets)
export class AssetsRepository extends Repository<Assets>{

    async find(): Promise<Assets[]>{
        const query = await this.createQueryBuilder( alias: 'assets');

        query.leftJoinAndSelect('assets.user', 'users')
            .leftJoinAndSelect('assets.category', 'categories')
            .leftJoinAndSelect('users.unit', 'units')
            .leftJoinAndSelect('assets.removingProtocol', 'protocols')
            .select([
                'assets',
                'protocols',
                'users.id', 'users.name', 'users.surname',
                'categories.id', 'units.id', 'units.name'])
        return query.getMany();
    }

}
```

Obrázek 32, AssetsRepository, vlastní tvorba

4.1.3 Komunikace mezi Angularem a NestJS

Komunikace mezi klientem a serverem probíhá převážně pomocí http dotazů a websocketu. V případě změn důležitých informací jako jsou změny u uživatelů, majetku či kategorií jsou změny řešeny pomocí websocketu, tak aby byly data aktuální na všech stanicích.

U běžných webových aplikací, které využívají pouze http dotazů, se změny dat projevují až v případě vyvolání nového http dotazu ať změnou webové strany, či

pomocí volání metody. V případě využití websocketu, je mezi klientem (Angular) a serverem (NestJS) otevřený komunikační kanál, na kterém klient „poslouchá“ zprávy od serveru a na základě jejich typu provádí akce.

Pokud tedy uživatel upraví u sebe v kanceláři majetek, změny se projeví i u ostatních uživatelů, bez toho, aniž by museli jakkoli vyvolat akci na načtení nových dat.

4.1.4 Nedostatky

V této sekci jsou popsány zjištěné chyby, popsané jejich možné řešení a věci, které v projektu nejsou ještě implementovány.

4.1.4.1 Vyskakující okna

Při vývoji bylo zjištěno, že knihovna Nebular v případě tvorby NbWindow či NbDialog komponent vytváří v DOM speciální wrapper s třídou „cdk-global-overlay-wrapper“ určený pro daný typ komponent, který v DOM zanechává. Tím je docíleno, že uživatel může mít více vyskočených oken na obrazovce. Ovšem pokud uživatel otevře okno typu A, zavře jej. Následně otevře okno typu B a z něj okno typu A, tak se okno typu A zobrazí pod oknem B.

Je možné, že v budoucnu bude tato chyba v knihovně vyřešena. Nyní jsou tři možnosti řešení:

1. Chybu neřešit
   Nedochází k ní často a pokud ano, spíše než-li se snažit opravit chybu v projektu, tak se snažit o opravu chyby tvůrcem Nebularu.

2. Nalezení obalujícího wrapperu a zvýšení hodnoty Z-index na nejvyšší
   V tomto případě se ale dostanou veškerá okna v tomto wrapperu, před okno, ze kterého bylo okno vyvoláno.

3. Zavření wrapperu v krajních případech

   U oken, u kterých je předpoklad, že se budou otevírat z druhé či třetí úrovně zajistit vyjmutí wrapperu z DOM. Zde ovšem dojde k uzavření i ostatních oken ve vyjmutém wrapperu.

V projektu, vzhledem k rychlosti řešení, byla zvolena možnost třetí, zavření wrapperu při zavření detailu majetku.

4.1.4.2 Oblasti pro dodělání

Oproti návrhu nejsou v projektu ještě implementovány **lokace**. Jsou zmíněny u majetku v detailu, ale není možné je jakkoli vytvářet ani upravovat.

U majetku je třeba dodělání přidávání **poznámek od uživatelů**. V kartě majetku je jeho použití nastíněno, ale ne zcela implementováno. Je zde třeba myslet na logiku úprav a zachování, resp. nezachovávání poznámek.

K majetku by bylo určitě vhodné implementovat možnost **přidání fotografií či dalších souborů** jako jsou smlouvy či manuály.

4.2 Entity

V této sekci jsou zmíněné hlavní a logické vlastnosti navržených entit a jejich úloha v rámci aplikace. Funkce s nimi související, včetně ukázky GUI z front-endové aplikace.

4.2.1 Uživatelské účty

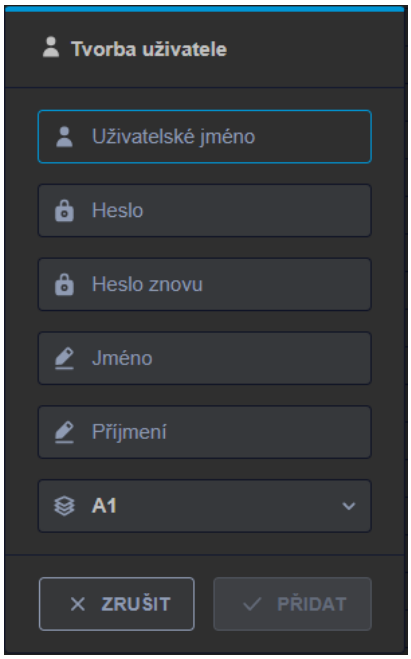
Aby bylo možné ovlivnit a nastavit různá práva, je třeba aby aplikace věděla, kdo k ní přistupuje.

S ohledem na budoucí vývoj nejsou v aplikaci předem definované role, ale je možné každému uživateli nastavit různá oprávnění, tím je zajištěna dostatečná diverzita a pro budoucí vývoj takový návrh umožňuje jednodušší přidávání omezení, případně přidávání dalších funkcí pro uživatele.

4.2.1.1 Tvorba uživatelů

Tvorba uživatelů probíhá prostřednictvím ostatních uživatelů, kteří mají oprávnění vytvářet uživatele, a to přes formulář viz. Obrázek 33, formulář tvorba uživatele, vlastní tvorba.

Je zde určité omezení a to, že každý nově vytvořený uživatel je přiřazen do jednotky, ve které se nachází jeho tvůrce, či v jednotce jí podřízené dle stromové struktury. Více o jednotkách viz. 4.2.3 Jednotky.



Obrázek 33, formulář tvorba uživatele, vlastní tvorba

4.2.1.2 Přihlašování uživatelů

Přihlašování uživatelů probíhá pomocí formuláře vyvolaného pomocí tlačítka v pravém horním rohu aplikace, kde uživatel zadá přihlašovací údaje. Ty jsou odeslány na server, který po ověření zadaných údajů vystaví JWT token obsahující údaje jako jsou unikátní číslo uživatele, jeho jméno, oprávnění a další.

Po ověření tokenu si jej klient uloží do prohlížeče do localStorage, pro případné příští načtení, např. při opětovném načtení prohlížeče.

4.2.1.3 Autentizace a autorizace

Při každém požadavku ze strany klienta na stranu serveru se pomocí tzv. Http Interceptoru (Obrázek 34) přidá do hlavičky požadavku autorizace s hodnotou tokenu.

```
@Injectable()
export class AuthInterceptor implements HttpInterceptor {
  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    const authJwtToken = localStorage.getItem( key: 'authJwtToken');

    if (authJwtToken) {
      const cloned = req.clone( update: {
        headers: req.headers
          .set('Authorization', `Bearer ${authJwtToken}`)
      });
      return next.handle(cloned);
    } else {
      return next.handle(req);
    }
  }
}
```

Obrázek 34, AuthInterceptor, vlastní tvorba

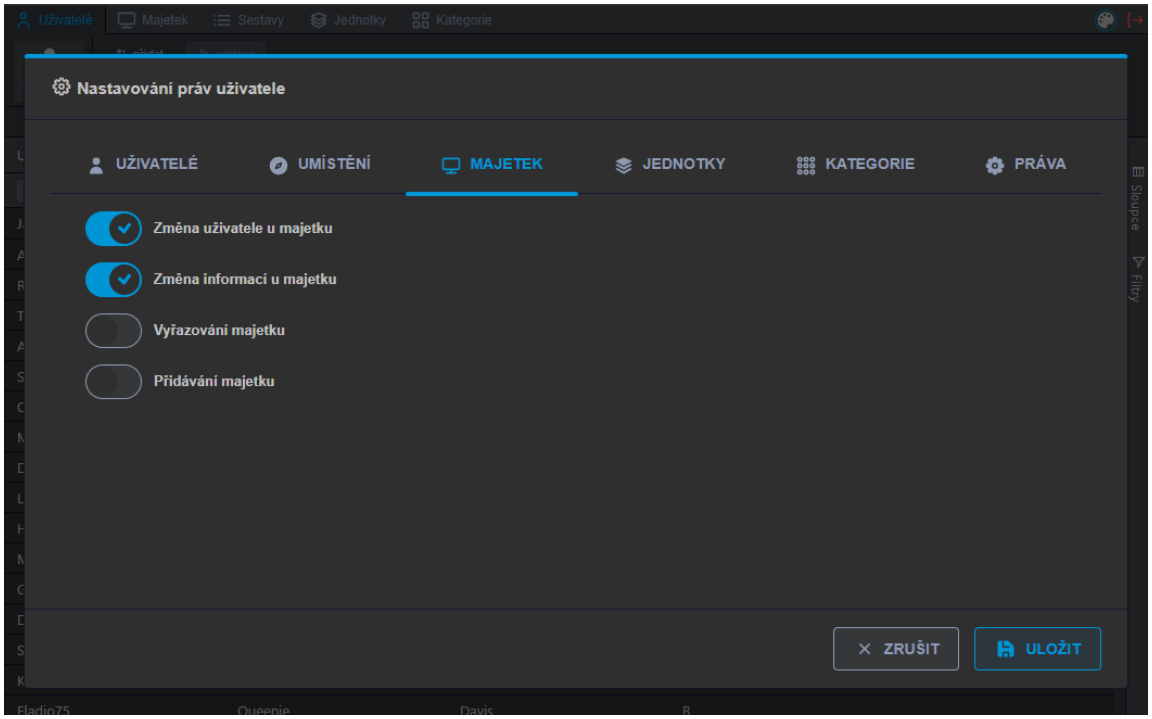
Takto doplněný požadavek je na straně serveru nejprve ověřen, zda je token validní, a poté je s jeho pomocí načten uživatel a jeho oprávnění. Následuje autorizace, kde je pomocí tzv. Guardu, zkontrolováno, zda má uživatel práva pro provedení akce. Na následující ukázce (Obrázek 35), je vidět použití dvou guardů: AuthGuard, RightsGuard, kde AuthGuard slouží pro ověření uživatele a RightsGuard pro autorizaci. Pomocí dekorátoru @RighstAllowed je specifikováno oprávnění, které je vyžadováno pro použití metody, v tomto případě pro tvorbu jednotky.

```
@Post()
@UseGuards(AuthGuard(), RightsGuard)
@RightsAllowed(RightsTag.createUnits)
createUnit(@Body(ValidationPipe) createUnitDto: CreateUnitDto, @GetUser() user: User): Promise<Unit> {
    return this.unitsService.createUnit(createUnitDto, user);
}
```

Obrázek 35, Ukázka použití Guardu, vlastní tvorba

4.2.2 Oprávnění

Oprávnění mohou nastavovat uživatelé, kteří mají oprávnění provádět tuto akci. Veškeré nastavení se provádí pomocí formuláře vyvolaného z editace uživatele (viz. Obrázek 36)



Obrázek 36, formulář nastavení oprávněn uživatele, vlastní tvorba

U nastavených oprávnění platí obdobné omezení jako v případě tvorby uživatelů. Uživatelská oprávnění vztahující se k uživatelům, jednotkám a právům, se opět promítnou pouze ve stromové struktuře pod jednotkou, kde je nastavovaný uživatel umístěn, včetně jednotky, ve které je uživatel zařazen.

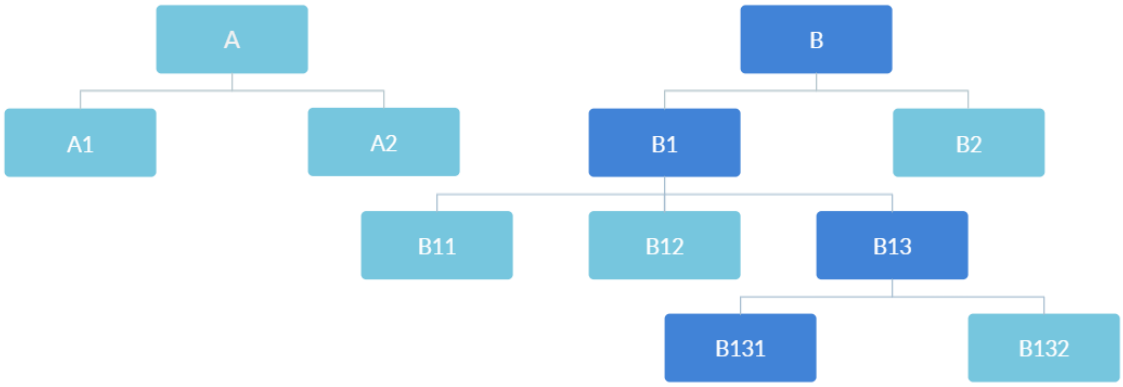
U položky kategorie se nepočítá s širokým počtem uživatelů s tímto právem a je zde žádoucí, aby kategorizace byla napříč aplikací.

Lokace se vztahují k celému stromu a nepředpokládá se zde velký počet uživatelů s tímto oprávněním.

4.2.3 Jednotky

Entita jednotka představuje část organizace ve stromové struktuře a určuje různá omezení pro uživatele a rozsah jeho práv. Jak je vidět na Obrázek 37, jakýkoli obdélník představuje jednu jednotku v hierarchii společnosti. V případě, že je uživatel přiřazen např. do jednotky B13, jeho rozsah práv pro editaci majetku

a uživatelů je pouze v rámci jednotky B13 a jednotek B131 a B132. Tímto mohou manažeři delegovat svoje pravomoci na uživatele níže.



Obrázek 37, Stromová struktura, vlastní tvorba

4.2.4 Kategorie

Tak, aby bylo pokryto široké spektrum společností, je kategorizace navržena jako stromová struktura bez předem daného rozsahu. Nepočítá se s tím, že by v rámci organizace bylo potřeba velké množství lidí s pravomocemi vytvářet kategorie majetku, a proto se tvorba kategorií promítá napříč aplikací.

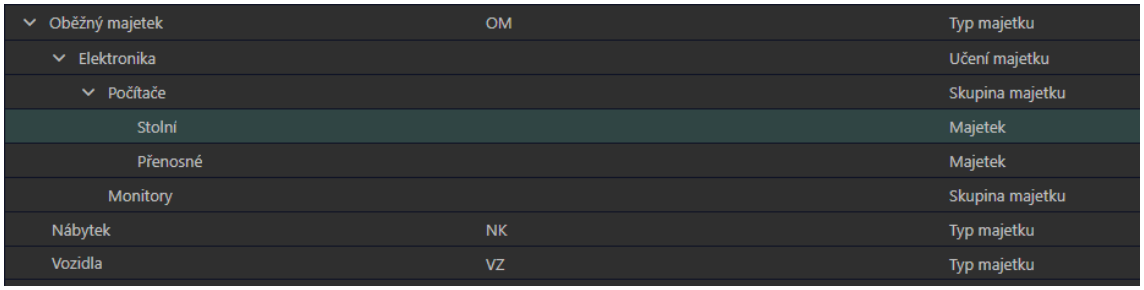
U každé kategorie je možné nastavit její název i kód, a poté v nastavení zobrazení sloupců definovat, zda se kategorie na dané úrovni bude zobrazovat pouze s názvem, či bude v tabulce s majetkem zahrnut i další sloupec, a to s kódovým označením kategorie.

4.2.5 Lokace

Umístění majetku se provádí pomocí lokací. Vzhledem k předpokladu, že nebude velké množství uživatelů, kteří by měli spravovat lokace, je toto právo delegováno v rámci stromu. Viz. Obrázek 37, uživatel přiřazený do jednotky B13, má možnost tvorbu a editaci lokací pro celý strom B, ale již neuvidí a neovlivní umístění ve stromě jednotek A.

4.2.6 Majetek

Majetek jako takový je položka, která může nabývat různých atributů. Záleží na konkrétních požadavcích. Její zařazení je ovšem úzce spjato s kategorizací. Každá položka tedy musí patřit do nějaké kategorie. Jak je vidět na Obrázek 38, Ukázka kategorií, vlastní tvorba tak v označené kategorii je možné vytvořit položku majetku, která bude mít vazbu na danou kategorii.



Obrázek 38, Ukázka kategorií, vlastní tvorba

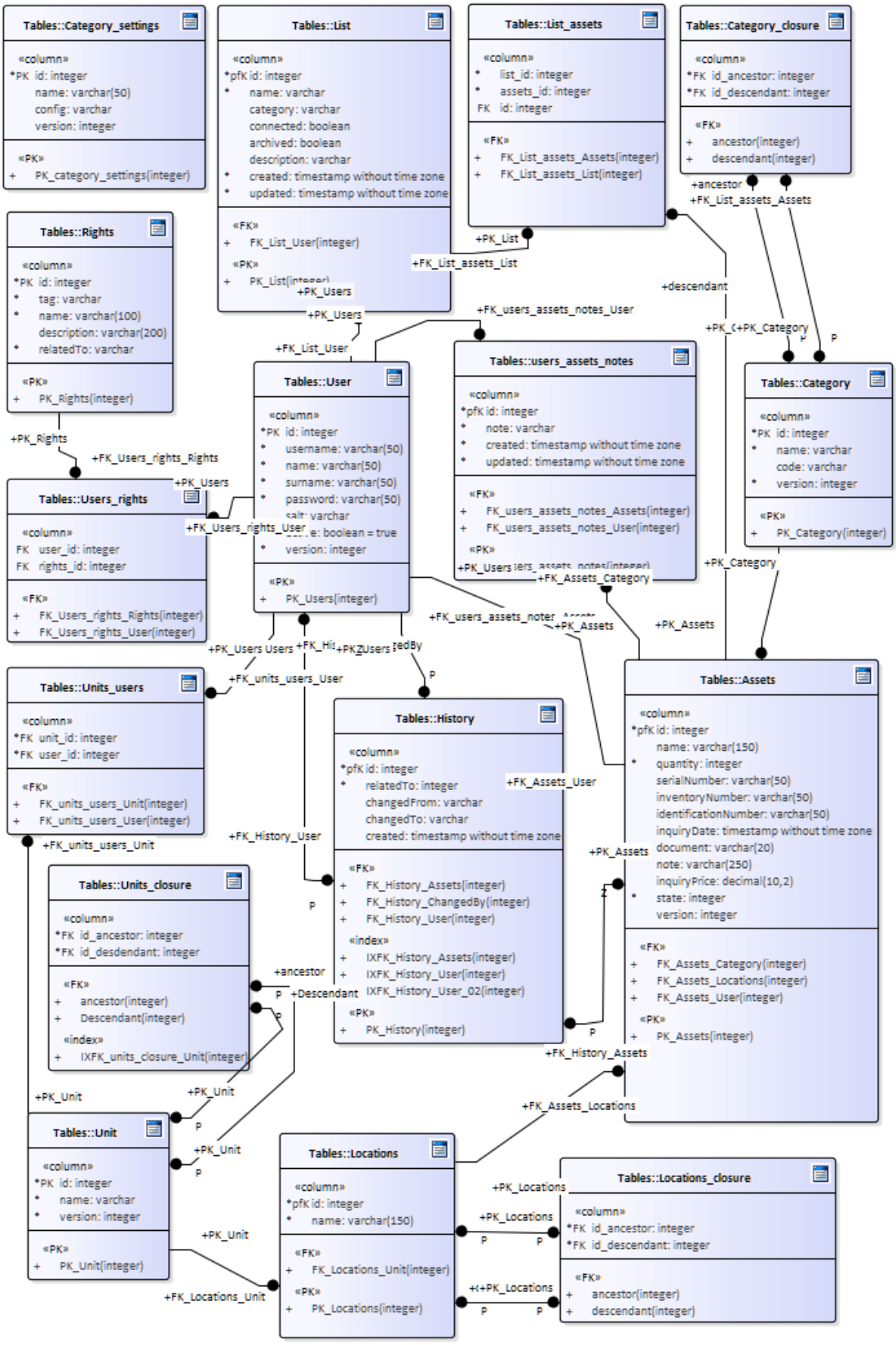
Dále je majetek možné přidávat do užívání jednotlivým uživatelům do správy v kartě majetku, kde má manažer k dispozici předat majetek do užívání osobám, které jsou ve stromové struktuře umístěné ve stejné jednotce jako manažer, případně v jednotkách pod ním.

4.2.7 Sestavy majetku

Sestavy majetku slouží pro ukládání majetku do určitých logických částí, které je možné v budoucnu použít např. pro tisk protokolů nebo pro rychlejší přechod k majetku a jeho hromadným úpravám pro dané položky. Aby mohl být majetek uložen do sestavy je nutné ho vložit do dočasné/pracovní sestavy a tu následně uložit.

4.2.8 Provázání entit

Na Obrázek 39 je znázorněné propojení jednotlivých tabulek v databázi.



Obrázek 39, diagram propojení entit, vlastní tvorba

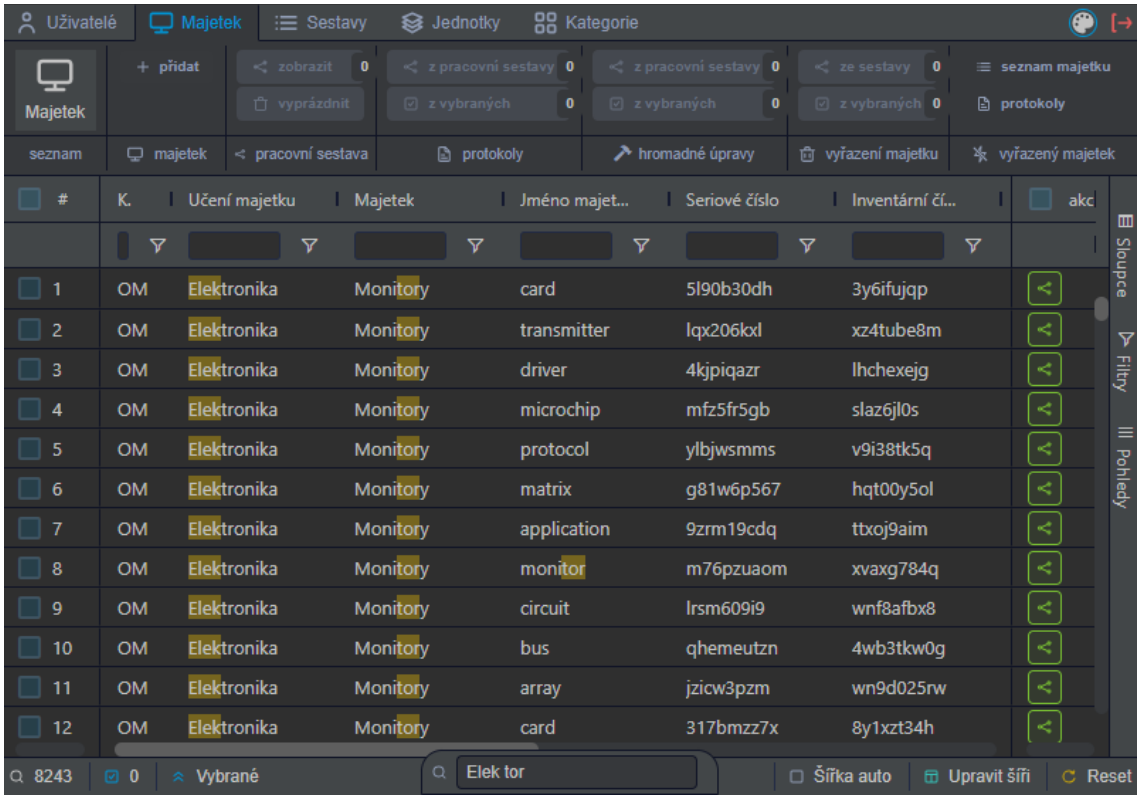
4.3 Vyhledávání majetku

V této kapitole jsou naznačeny možné způsoby vyhledávání majetku a scénáře, ke kterým může dojít a zpravidla nejčastěji dochází.

4.3.1 Vyhledávání majetku pomocí rychlého filtru

Nejčastější scénář vyhledávání majetku je v případě, že chce majetkový správce vyhledat majetek na základě nejběžnějších identifikačních údajů jako jsou výrobní číslo, či jen zobrazení konkrétního majetku přiděleného uživateli.

Tzv. rychlý filtr je umístěný na střed ve spodní části obrazovky (Obrázek 40, element obsahující výraz: „Elek tor“) a je přístupný v rámci celé aplikace. Výraz zapsaný v tomto filtru se aplikuje na veškeré buňky v rámci tabulky a zobrazí se pouze záznamy obsahující zadané výrazy, jako oddělovač slouží mezera.



Obrázek 40, rychlý filtr, vlastní tvorba

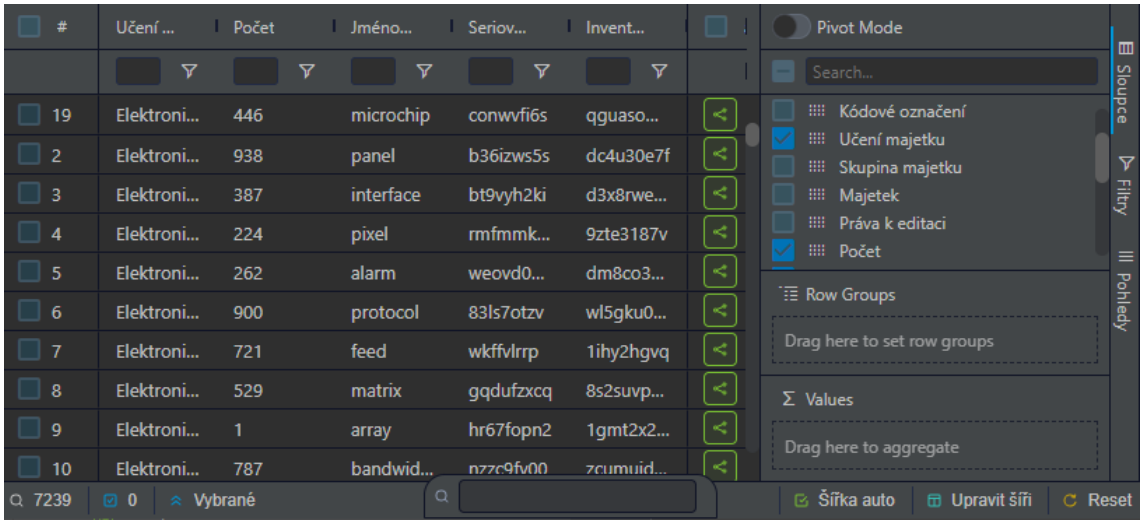
Použití tohoto filtru, vzhledem k flexibilitě a díky tomu, že hledá ve všech veškerých položkách je nejčastější. Často může být důležitá informace vztahující se k věci či uživateli např. v poznámce. Pokud bychom vyhledávali pouze nad jedním sloupcem, tak bychom o tuto informaci mohli přijít.

4.3.2 Filtrování pomocí filtrů ve sloupcích

Další způsob, jak filtrovat majetek je pomocí filtrů, které jsou umístěné nad každým sloupcem hodnot v tabulce viz. Obrázek 40. Tímto filtrem se použije omezení pouze na daný sloupec. Tzv. rychlý filtr a filtr nad sloupci se aplikují současně. Tzn., pokud chceme pouze majetek od jednoho uživatele, vybereme jej v horním filtru, pokud chceme majetek omezit ještě na určitou lokaci, zadáme omezení i pro daný sloupec.

4.3.3 Seskupování kategorií a tvorba vlastních pohledů

Často si každý uživatel volí pořadí sloupců, tak jak mu vyhovuje a na základě toho co momentálně potřebuje s daty vykonávat. Díky knihovně ag-Grid je možné i data agregovat a tím velice podstatně změnit vzhled zobrazených dat a úplně změnit použití zobrazených dat.



Obrázek 41, neagregovaná data, vlastní tvorba

Na Obrázek 41 jsou vidět ukázková data, která nejsou nijak seskupena, ale pomocí funkce „Row Groups“ je možné data agregovat, např. pomocí uživatele viz. Obrázek

42. Zde jsou data již agregována a je vedle uživatele vidět i kolik položek daný uživatel obsahuje. Touto funkcí je možné i agregace zanořovat.

Obrázek 42, agregovaná data, vlastní tvorba

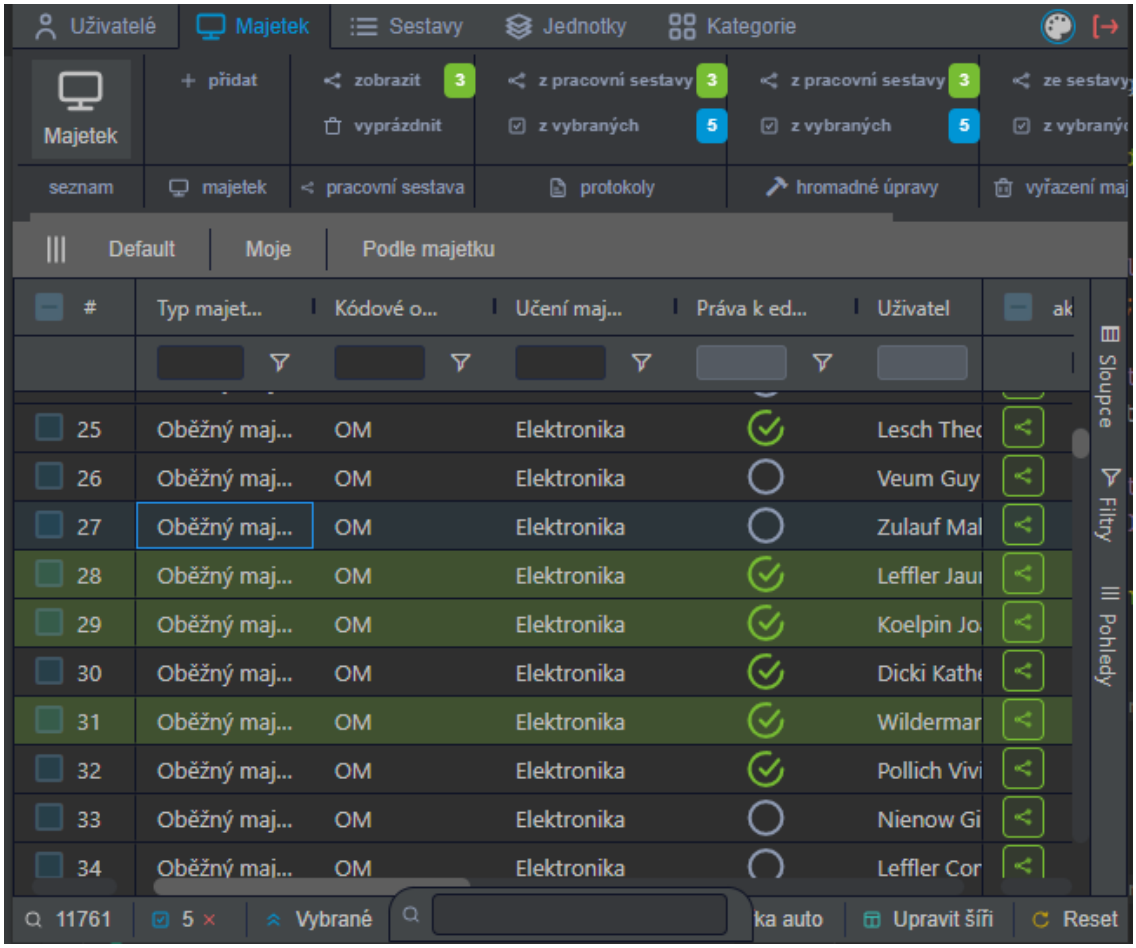
Takto nastavené seskupení, případně různé filtry je možné uložit do tzv. pohledů. Pohledy je možné ukládat a mít k nim přístup i v budoucnu. Tím má uživatel k dispozici předpřipravené pohledy, tak jak si sám definoval a nemusí se spoléhat na optimální nastavení ze strany tvůrce softwaru, který často neví, nad jakými daty uživatel pracuje a co s nimi potřebuje dělat.

4.4 Sestavy majetku

Aby měl správce majetku větší přehled o majetku a mohl s ním disponovat rychleji i v budoucnu, je možné si ukládat majetek do sestav.

4.4.1 Dočasné sestavy

Majetek uložený v této sestavě je uložen pouze v paměti klienta a v případě znovunačtení prohlížeče je sestava ztracena. Jak je vidět na Obrázek 43, tak majetek přidaný v dočasné (pracovní) sestavě je podbarven jinou barvou než majetek, který v pracovní sestavě není.



Obrázek 43, Ukázka sestavy, vlastní tvorba

Dále je možné sestavu zobrazit pomocí tlačítka umístěného v menu „zobrazit“, ve sloupci „pracovní sestava“. Po zobrazení „pracovní“ sestavy je v možné sestavě přiřadit název, popis a kategorii a sestavu uložit pro pozdější použití, a to i v případě běžného uživatele.

4.4.2 Uložené sestavy

Seznam uložených sestav využívá také možnosti tvorby vlastních pohledů a rychlých odkazů nad tabulkou, takže si každý uživatel může navolit vlastní filtry, resp. pohledy pro různá zobrazení sestav. Majetek přiřazený do uložené sestavy je kdykoli možné zobrazit, zaslat ostatním uživatelům či vložit jej do pracovní sestavy.

Sestavy je možné ukládat s názvem i popisem sestavy, případně slovním popisem kategorie pro větší přehled nad sestavami a pro případné filtrování.

U každé sestavy je možné sestavu převést na „Uloženou svázanou sestavu“.

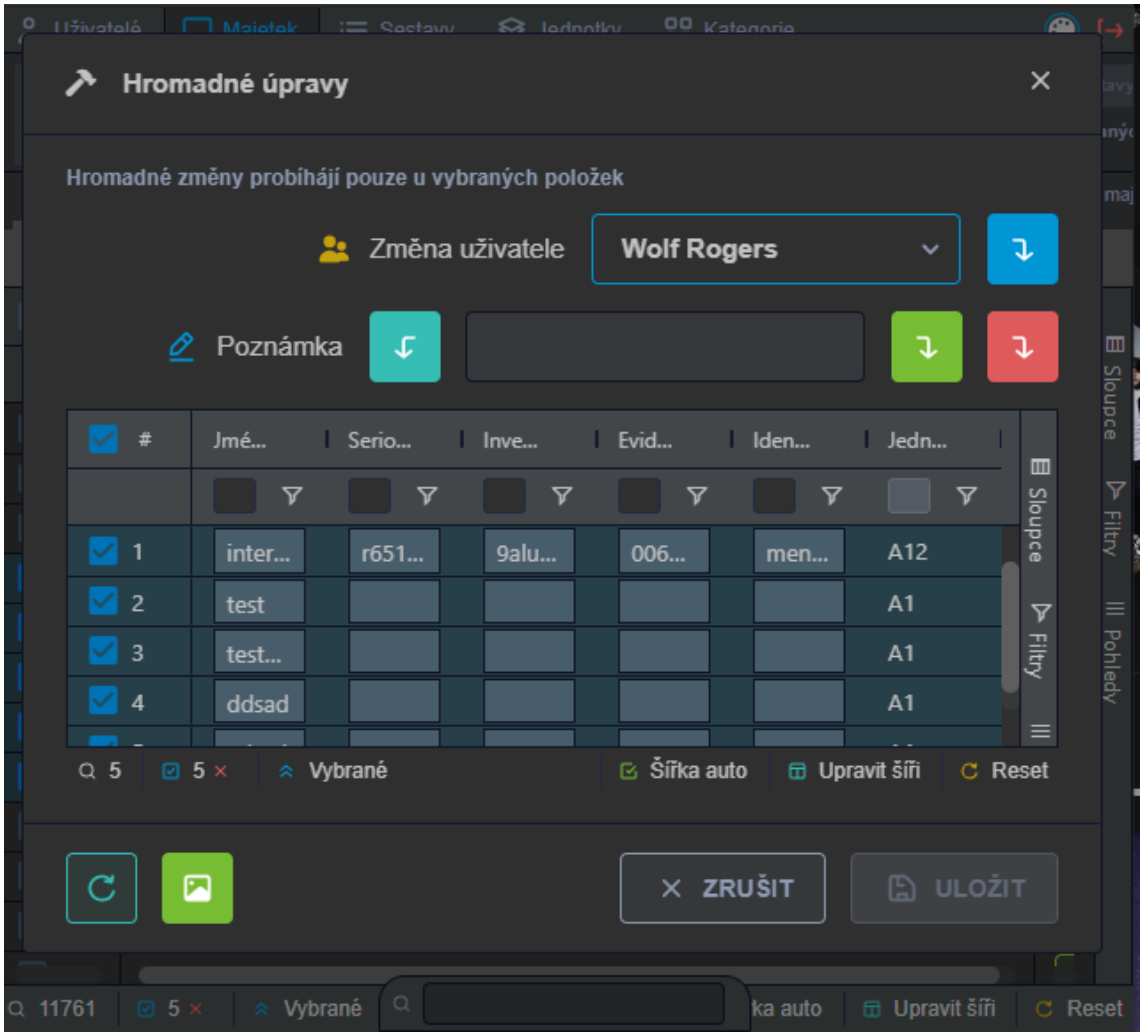
4.4.3 Uložené svázané sestavy

Tento typ je pouze barevně odlišený typ sestavy uložené, tak aby bylo na první pohled jasné, že se jedná o důležitou sestavu a majetek uložený v ní má svůj konkrétní význam. Např. součásti technologií, které je možné použít pouze pospolu a samy o sobě nemají význam.

4.5 Hromadné úpravy

Vzhledem k systému navrhovaném pro společnost s velkým pohybem aktiv, je třeba, aby bylo možné nad majetkem provádět hromadné úpravy. Zejména změnu uživatele, lokace či doplnění nebo změny poznámky u materiálu, tak aby aplikace šetřila čas správce majetku.

Tyto úpravy je možné dělat přes dialog vyvolaný z menu v sekci majetek, hromadné úpravy, a to buď na aktuálně vybraném majetku, nebo na majetku v pracovní sestavě.



Obrázek 44, ukázka hromadné úpravy, vlastní tvorba

Na Obrázek 44, ukázka hromadné úpravy, vlastní tvorba je ukázka dialogu hromadných úprav. Aby nebyl správce limitován prováděním změn nad všemi položkami, je možné ještě v tomto dialogu provádět výběr a veškeré změny se aplikují pouze na vybrané položky. Na ukázce je vidět změna uživatele u vybraných položek a úprava poznámky: přidání na začátek (světle modré tlačítko), přidání na konec aktuální poznámky (zelené tlačítko) a přepsání poznámky (červené tlačítko).

Tento dialog byl spuštěn s účtem s právy úpravy informací o majetku, a proto jsou v tabulce vidět input boxy pro editaci položek jako je jméno.

Veškeré změny je potřeba potvrdit tlačítkem uložit. Změny probíhají odděleně na typu práv. Pokud má správce práva měnit uživatele i informace o majetku jsou

na server odeslány dva požadavky. Jeden na změnu uživatelů i majetku a druhý na změnu informací o majetku. Změny jsou na serveru prováděny v transakcích (buď se provedou všechny úpravy nebo se neprovede žádná). Ukázka implementace na serveru je vidět na Obrázek 45, transakce, vlastní tvorba.

```
async changeUser(assetId: number, userId: number, user: User,
                 managerForTransaction?: EntityManager): Promise<Assets> {
    const toUser: User = await this.usersService.getReachableUser(userId, user);
    const asset: Assets = await this.assetsRepository.findOneOrFail( conditions: {id: assetId})
    const isInTree = await this.unitsService.isManagerInTree(asset.user.unit.id, user);

    if (!isInTree) {
        throw new ForbiddenException('You are not able to do this operation');
    }
    asset.user = toUser;
    if (managerForTransaction) {
        return await managerForTransaction.save(asset);
    }
    return await asset.save();
}

changeUserBulk(changeUserBulkDto: ChangeUserBulkDto[], user: User): Promise<Assets[]> {
    return this.connection.transaction( runInTransaction: async manager => {
        return await Promise.all(changeUserBulkDto.map(async change => {
            return await this.changeUser(change.assetId, change.newUserId, user, manager);
        }))
    });
}
```

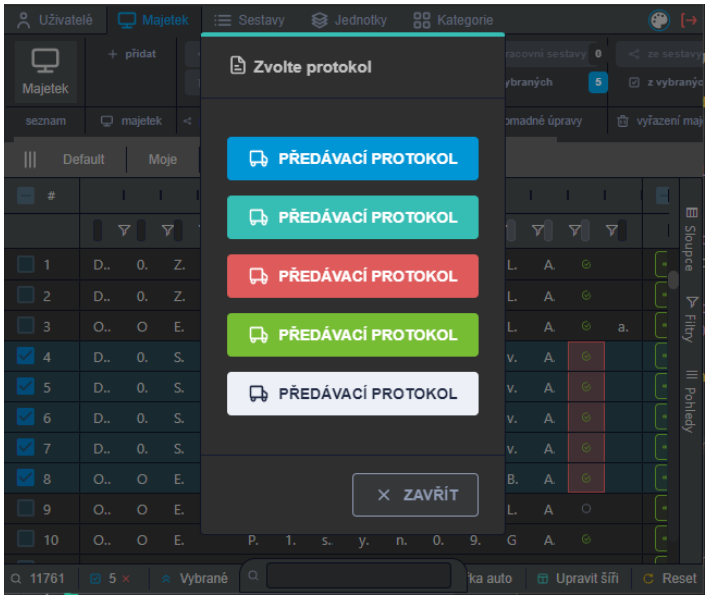
Obrázek 45, transakce, vlastní tvorba

V případě volání metody „changeUserBulk“ je použit EntityManager pro všechna uložení entit. Jakmile se jedna operace neprovede, neprovede se žádná a server vrací chybu.

4.6 Protokoly

Pomocí aplikace je možné pořizovat tiskopisy, které mohou mít různý formát, dle potřeby použití. Mimo základních jako jsou majetek umístěný na dané lokaci (např. místnosti), určený do správy uživatele a předávací protokol, je možné si navrhnout i vlastní šablony a ty jednoduše implementovat do aplikace.

Protokol je možné vytvořit z vybraného majetku pomocí kliknutí na tlačítko v menu v sekci protokoly. Následně je uživateli nabídnut typ protokolu viz. Obrázek 46.



Obrázek 46, ukázka volby protokolu, vlastní tvorba

Po výběru příslušného protokolu dojde k otevření předdefinovaného protokolu s vybranými položkami majetku. Na Obrázek 47 je ukázka jednoduchého protokolu

Doklad.č.....
Ze dne 16. 3. 2021
Jednotka.....

Předávací protokol

#	Věc	Počet	Sériové číslo
1	Monitor Dell 34	1	r651o20hy
2	Židle	1	
3	Stůl	1	
4	PC	1	SE5010
5	Sluchátka	1	

Dne: 16. 3. 2021 Dne: 16. 3. 2021

Předávající Přebírající

Obrázek 47, předávací protokol, vlastní tvorba

5 Závěr

Angular patří mezi přední frameworky pro tvorbu frontendových aplikací, existuje pro něj spousta rozšíření v podobě npm balíčků, které jsou velice snadné na implementaci, ale i přesto tvorba systému pro evidenci majetku není triviální záležitost.

Bez předchozí znalosti problému by tvorba takového systému trvala daleko delší dobu a předcházela by jí spousta analýz, což by značně prodloužilo dobu vývoje systému. Dle nástroje pro měření času integrovaného ve vývojovém prostředí Intellij Idea trval vývoj přiložené aplikace cca 120 hodin na straně backendu a cca 400 hodin na frontendu.

Při hodinové sazbě cca 350 Kč / hod. by náklady na projekt byly ve výši 182 000 kč, k výsledné sumě ještě není započítána částka $750 za licenci ag-gridu. Proto je třeba zvážit před vývojem takovéhoto projektu, jsou-li individuální funkce a rozšiřitelnost systému pro vlastní potřeby natolik důležité a není možné využití jiných nástrojů k podobným účelům, byť s kompromisy v dané oblasti, ale s množstvím dalších funkcí v jiných oblastech.

Jen pro představu ABRA Flexi Basic verze účtu pro provoz na vlastním serveru stojí 3 950 Kč za účet, nahlížení do systému bez účtu je možné. Pokud by bylo potřeba 300 aktivních účtů je výsledná suma 1 185 000 Kč.

Výsledkem práce je systém pro evidenci majetku. V systému je možné vytvářet, měnit a mazat kategorie majetku, jednotky, resp. logické celky společnosti, majetek, uživatele, sestavy majetku a umístění. Aplikace obsahuje funkce pro usnadnění manipulace s majetkem a jeho správou, jako je hromadná úprava poznámek, editace položek z jednoho pohledu a hromadná změna uživatele. Umožňuje jednoduše implementovat různé tiskopisy. Zobrazuje historii majetku a dovoluje běžným uživatelům vytvářet si vlastní sestavy majetku. Systém využívá websocketů, takže změny provedené vidí i ostatní uživatele v reálném čase, bez nutnosti obnovení prohlížeče.

Systém je primárně určen pro interní síť společnosti a předpokládá se využití prohlížečů co websocket podporují a také použití zobrazovacího zařízení s minimálně full HD rozlišením, lépe pak monitor s 34“ a nativním rozlišením. Po specifických úpravách pro konkrétní společnost a zakoupení licence ag-grid je možné systém nasadit v reálném provozu.

Systém je dále možné rozvíjet v oblasti lokací, které by měli být zadávány stromovou strukturou, aby bylo možné jednoduše omezovat majetek podle oblasti. Další velké ulehčení pro evidenci by bylo zajisté převést aplikaci na tzv. „PWA“ aplikaci, tak aby jí bylo možné instalovat na jiná zařízení a naprogramovat část, která bude tvořit a číst QR kódy a tím i urychlit inventury a zpřesnit evidenci možnými požadavky na změnu umístění dle naskenovaných kódů v daných lokacích.

Cíl, vytvořit systém pro evidenci majetku ve frameworku Angular, byl splněn.

6 Seznam zdrojů

6.1 Knižní zdroje

ROZENTALS, Nathan. Mastering TypeScript: Second Edition, Birmingham, Packt Publishing Limited, 2017, 552. ISBN 978-1-78646-871-0

LIM, Greg. Beginning Angular 2 with TypeScript, CreateSpace Independent Publishing Platform

DEELEMAN, Pablo. Learning Angular 2: Your quick, no-nonsense guide to buiding real-world apps with Angular 2, Packt Publishing Limited, 2016, 346. ISBN 978-1-78588-207-4

6.2 Internetové zdroje

EASY FM. Informační systém pro správu majetku SW KLID [online]. Praha: EASY FM, © 2014-2020 [cit. 2020-07-28]. Dostupné z: https://easyfm.cz/informacni-system-pro-spravu-majetku-sw-klid/

Onesoft Connect Inc. Software pro správu, údržbu a evidenci majetku [online]. Redwood city: Onesoft Connect, 2020 [cit. 2020-07-28]. Dostupné z: https://www.onesft.com/cs/asset-management-majetek

Evidei. Kodys [online]. Praha: Kodys, spol. s r.o., 2020 [cit. 2020-07-31]. Dostupné z: https://www.kodys.cz/produkty/software/komplexni-systemy/evidei-evidence-inventarizace-majetku

Google. Angular [online]. Silicon Valley: Google, 2010-2020a [cit. 2020-12-08]. Dostupné z: https://angular.io/guide/entry-components

Google. Angular [online]. Silicon Valley: Google, 2010-2020b [cit. 2020-12-08]. Dostupné z: https://angular.io/guide/structural-directives

MYSLIEWIEC, Kamil. NestJS [online]. Polsko: Mysliwiec, 2017-2020a [cit. 2021-01-03]. Dostupné z: https://docs.nestjs.com/fundamentals/circular-dependency#forward-reference

MYSLIEWIEC, Kamil. NestJS [online]. Polsko: Mysliwiec, 2017-2020b [cit. 2021-01-03]. Dostupné z: https://docs.nestjs.com/guards

MYSLIEWIEC, Kamil. NestJS [online]. Polsko: Mysliwiec, 2017-2020c [cit. 2021-01-03]. Dostupné z: https://docs.nestjs.com/custom-decorators

Auth0, JSON Web Tokens [online]. Washington: Auth0, [2013] [cit. 2021-01-07]. Dostupné z: https://jwt.io/introduction

7 Přílohy

7.1 Příloha č. 1 – elektronická příloha

Obsah elektronické přílohy (source.rar), CD-ROM

- zdrojové kódy pro Angular (složka frontend)
- zdrojové kódy pro NestJS (složka rest)

7.2 Příloha č. 2 – dotazníky

Dotazník pro průzkum softwaru pro evidenci majetku

Název aplikace:

Fusio

Aplikace je provozována jako:

☒ Desktopová aplikace ☐ Webová aplikace

☐ Webová i desktopová

Data jsou uchovávána u

☒ Objednatele ☐ Poskytovatele

Může být provozována na interní síti společnosti?

☒ Ano ☐ Ne

Je majetek řazen v kategoriích pomocí stromové struktury?

☒ Ano ☐ Ne

Pokud **Ano** uveďte prosím maximální úroveň či zda je neomezená

neomezená

Je možné majetek vyhledávat pomocí omezení této struktury?

☒ Ano ☐ Ne

** Např. existují kategorie A, B, C. Kategorie A obsahuje kategorie A1, A2. Kategorie A1 obsahuje A1.1, A1.2 atd. Pokud si vyberu kategorii A uvidím veškerý majetek v této kategorii a kategoriích určených stromovou strukturou, resp. vyberu-li kategorii A1 uvidím majetek zadaný v kategorii A1, A1.1 a A1.2 a vše ostatní co je pod těmito kategoriemi?*

Je počet položek majetku v aplikaci omezený?

☐ Ano ☒ Ne

Pokud Ano jaké je maximální množství položek majetku

Je počet uživatelů omezený?

☐ Ano ☒ Ne

Pokud **Ano** jaké je maximální množství uživatelů

Je možné osobě svěřit majetek do užívání, resp. přiřadit k majetku osobu?

☒ Ano ☐ Ne

Umožňuje aplikace osobám, kterým je majetek svěřen náhled svěřeného majetku?

☒ Ano ☐ Ne

Je možné mít v systému více účtů: „Správců majetku" s tím, aby každý odpovídal pouze za osoby a majetek jemu svěřený včetně úprav?

☒ Ano ☐ Ne

Je možné v aplikaci provádět hromadné úpravy nad vybraným majetkem?

☒ Ano ☐ Ne

Pokud **Ano** jaké z níže uvedených

☒ Změna umístění ☒ Změna osoby, které je majetek svěřen

☒ Změna poznámky ☐ Uložení sestav pro pozdější využití

☒ Odstranění majetku

Je možné k majetku nahrávat přílohy jako jsou obrázky, smlouvy atp.?

☒ Ano ☐ Ne

Dotazník pro průzkum softwaru pro evidenci majetku

Název aplikace:

Evidei

Aplikace je provozována jako:

☒ Desktopová aplikace ☐ Webová aplikace

☐ Webová i desktopová

Data jsou uchovávána u

☒ Objednatele ☒ Poskytovatele

Může být provozována na interní síti společnosti?

☒ Ano ☐ Ne

Je majetek řazen v kategoriích pomocí stromové struktury?

☒ Ano ☐ Ne

Pokud **Ano** uveďte prosím maximální úroveň či zda je neomezená

- *Minimálně 3 fyzické úrovně (Lokalita, Oddělení, Místnost) a možnost dalších několika virtuálních (např. majetek přidělený určité osobě)*

Je možné majetek vyhledávat pomocí omezení této struktury?

☒ Ano ☐ Ne

** Např. existují kategorie A, B, C. Kategorie A obsahuje kategorie A1, A2. Kategorie A1 obsahuje A1.1, A1.2 atd. Pokud si vyberu kategorii A uvidím veškerý majetek v této kategorii a kategoriích určených stromovou strukturou, resp. vyberu-li kategorii A1 uvidím majetek zadaný v kategorii A1, A1.1 a A1.2 a vše ostatní co je pod těmito kategoriemi?*

Je počet položek majetku v aplikaci omezený?

☒ Ano ☐ Ne

Pokud Ano jaké je maximální množství položek majetku

- *Podle zakoupené licence, Aplikace je multilicence z pohledu počtu uživatelů, ale licencuje se počet karet majetku, které můžou být v databázi*

Je počet uživatelů omezený?

☐ Ano ☒ Ne

Pokud **Ano** jaké je maximální množství uživatelů

Je možné osobě svěřit majetek do užívání, resp. přiřadit k majetku osobu?

☒ Ano ☐ Ne

Umožňuje aplikace osobám, kterým je majetek svěřen náhled svěřeného majetku?

☒ Ano ☐ Ne

Je možné mít v systému více účtů: „Správců majetku" s tím, aby každý odpovídal pouze za osoby a majetek jemu svěřený včetně úprav?

☒ Ano ☒ Ne

Je možné v aplikaci provádět hromadné úpravy nad vybraným majetkem?

☒ Ano ☐ Ne

Pokud **Ano** jaké z níže uvedených

☒ Změna umístění ☒ Změna osoby, které je majetek svěřen

☒ Změna poznámky ☒ Uložení sestav pro pozdější využití

☐ Odstranění majetku

Je možné k majetku nahrávat přílohy jako jsou obrázky, smlouvy atp.?

☒ Ano ☐ Ne

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání bakalářské práce

Autor: **Milan Knop**

Studium: I1800495

Studijní program: B1802 Aplikovaná informatika

Studijní obor: Aplikovaná informatika

Název bakalářské práce: **Systém pro správu majetku v Angular frameworku**

Název bakalářské práce AJ: Assets Management System in Angular framework

Cíl, metody, literatura, předpoklady:

Cílem práce je vytvořit webovou aplikaci pro správu majetku postavenou na frameworku Angular. Aplikace bude umožňovat správu uživatelů a majetku jim přiděleného. Aplikace bude vytvářet předávací protokoly, karty přiděleného majetku osobě, operace s majetkem v dočasných sestavách, vyhledávání nad databází majetku a bude uchovávat jeho historii. Dále bude umožňovat jednotlivým uživatelům tvorbu jejich sestav majetku a vytváření poznámek k jednotlivým položkám.

Hlavní osnova:

1. Úvod
2. Analýza
3. Přehled nástrojů a knihoven (představení frameworku Angular a jazyka Typescript)
4. Návrh a implementace aplikace
    1. na straně klienta ve frameworku Angular,
    2. na straně serveru ve frameworku NestJS a platformě node.js
5. Výsledky a doporučení
6. Závěr

https://angular.io/

https://nestjs.com/

Garantující pracoviště: Katedra informatiky a kvantitativních metod, Fakulta informatiky a managementu

Vedoucí práce: Ing. Pavel Kříž, Ph.D.

Datum zadání závěrečné práce: 15.10.2020