Tvorba systému pro plánování kapacit v aplikaci Perm3

Bc. Oliver Balun

Diplomová práce
2020

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně

Fakulta aplikované informatiky

Ústav informatiky a umělé inteligence

Akademický rok: 2019/2020

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Oliver Balun**
Osobní číslo:	**A18246**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Forma studia:	**Prezenční**
Téma práce:	**Tvorba systému pro plánování kapacit v aplikaci Perm3**
Téma práce anglicky:	**The Creation of a Capacity Planning System for the Perm3 Application**

Zásady pro vypracování

1. Popište technologie použité k tvorbě řešení.
2. Navrhněte databázové struktury a aplikační objekty.
3. Dle návrhu zapracujte definiční a číselníkové struktury do aplikace Perm3.
4. Vytvořte výkonnou knihovnu pro aplikační server P3Server.
5. V rámci webové aplikace PermWeb vytvořte rozhraní s využitím výkonné knihovny.
6. Popište možnosti budoucího vývoje.

Rozsah diplomové práce:
Rozsah příloh:
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. CANTU, Marco, WOOD, Peter, ed. Object Pascal handbook. Piacenza (Italy): Marco Cantu, 2014. ISBN-10 1514349949.
2. GRUBER, Martin. Mistrovství v SQL. Praha: Softpress, 2004. ISBN 8086497623.
3. MARTIN, Robert C. Čistý kód: [návrhové vzory, refaktorování, testování a další techniky agilního programování]. Brno: Computer Press, 2009. ISBN 978-80-2512285-3.
4. NICK, Hodges. More coding in Delphi. Nepeta Enterprises, 2015. ISBN 978-1941266106.
5. KOZLOWSKI, Pawel. Mastering Web Application Development with AngularJS. Packt Publishing, 2013. ISBN 978-1782161820.

Vedoucí diplomové práce: **doc. Ing. Jiří Vojtěšek, Ph.D.**
Ústav řízení procesů

Datum zadání diplomové práce: **28. listopadu 2019**
Termín odevzdání diplomové práce: **15. května 2020**

doc. Mgr. Milan Adámek, Ph.D.
děkan

prof. Mgr. Roman Jašek, Ph.D.
ředitel ústavu

Ve Zlíně dne 9. prosince 2019

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 30. 7. 2020

Oliver Balun, v. r.

.....

podpis diplomanta

ABSTRAKT

Tato diplomová práce pojednává o tvorbě systému pro plánování kapacit pracovníků na definovanou provozní jednotku s využitím stávajícího systému tvorby harmonogramů směn v aplikaci Perm3. Zabývá se popisem kapacitního plánování a jeho využitelností pro tvorbu směn zaměstnanců. Popisuje návrh databázových struktur a datových objektů, využitelných k tvorbě modulu plánování, a jejich následnou realizací v rámci aplikace Perm3. V této aplikaci bude realizována pouze definice provozních jednotek. Vlastní plánování bude probíhat ve webové aplikaci PermWeb. Řešení se skládá jednak z výkonného modulu plánování kapacit, tedy knihovny pro manipulaci s daty vytvořené v prostředí Embarcadero Delphi, a také webového klienta vytvořeného pomocí TypeScriptového frameworku Angular.

Klíčová slova: Kapacitní plánování, systém pro plánování, plánování, plánování směn, plánování zaměstnanců, plánovací jednotka, Pascal, Delphi, DLL, Perm3, PermWeb, rozšíření aplikace, modul

ABSTRACT

This diploma thesis describes creation of a system for planning workers capacity for a defined work unit. This system uses current methods for creation shift schedules in the Perm3 application. Substantial theme is capacity planning and its usability for shift schedule purposes. It deals with the design of database structures and data objects that can be used to create the planning module and their subsequent implementation within the Perm3 application. Only the definition of work units will be implemented in this application. The actual scheduling will take place in the PermWeb web application. The solution consists of the executive planning module which is the library for data manipulation created in Embarcadero Delphi and web client in the Angular TypeScript framework.

Keywords: Capacity planning, systém for planning, planning, planning, shift planning, employee planning, planning unit, Pascal, Delphi, DLL, Perm3, PermWeb, application extension, module

Touto cestou bych chtěl poděkovat především svým rodičům, kteří mi dali možnost studovat a vždy mi pomáhali rozvíjet mé znalosti, schopnosti a dovednosti. Také bych chtěl poděkovat své rodině, přítelkyni a kamarádům za toleranci a psychickou podporu v čase vypracovávání diplomové práce. Dále děkuji společnosti Kvasar, spol. s r. o. za možnost tvorby této práce a poskytnutí aplikačních řešení, které byly v rámci práce využity, a za mnoho zkušeností, které jsem měl možnost získat. Děkuji svému vedoucímu diplomové práce doc. Ing. Jiřímu Vojtěškovi, Ph.D. a svému konzultantovi Ing. Jiřímu Uchytilovi za přímočaré vedení, rady a konzultace ohledně této práce. Děkuji také Mgr. Michalu Miklášovi a Ing. Jaromíru Světlíkovi, kteří mě na střední škole nasměrovali k oboru informačních technologií.

Prohlašuji, že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

Tato diplomová práci se zabývá tvorbou systému pro usnadnění činnosti plánování lidských zdrojů v rámci aplikace Perm3 a PermWeb. Jedná se především o vytvoření nového modulu aplikace a zapracování jeho funkcionality do stávajícího řešení. Aby bylo možné práci naplnit, je nutné seznámit se s aktuálními možnostmi aplikace, nastudovat teorii této problematiky a použít poznatky v praxi. Nejprve budou popsány technologie, které jsou při tvorbě systému použity, aby bylo srozumitelné jejich následné použití v praktické části. Základem pro zhotovení systému je návrh databázových struktur a vytvoření tabulek, které budou nést informace nutné k tvorbě plánů a definic pracovních jednotek. Znalosti o aplikaci Perm3 budou využity k následnému vytvoření aplikačních objektů pro práci s daty na základě vzniklých tabulek. Pro účely prezentace dat poskytnutých objekty bude vytvořeno několik aplikačních oken v souladu s aplikací Perm3, ve kterých bude možné s daty pracovat. Dále bude detailně popsáno vytváření výkonné dynamické knihovny, která obsluhuje manipulaci s daty, a za použití definic sestavuje plány pracovních jednotek, jejich pracovních míst a zaměstnanců. Tato knihovna je použita pro účely prezentace a úprav plánů ve webové aplikaci PermWeb, jejíž rozhraní slouží uživateli k práci s plány a je vhodné pro manažerskou činnost. Vytvoření rozhraní je posledním krokem ke kompletnímu zapracování systému do stávajícího řešení. Všechny tyto body jsou předmětem práce a jejich zpracování se věnují následující kapitoly. Při tvorbě bude dbáno na jednoduchost a intuitivnost grafického rozhraní a zajištění rychlého chodu všech částí systému. Práce bude vytvářena tak, aby byla srozumitelná, nebyla jednoúčelová, ale bylo možné na její části navázat a rozšiřovat funkcionalitu systému.

I. TEORETICKÁ ČÁST

1 PLÁNOVÁNÍ

Plánování je rozhodovací činnost, která je prováděna za účelem dosažení našich cílů. Umožňuje efektivní provádění plánovaných činností a pozitivně přispívá k jejich úspěšnému dokončení. Během této činnosti je definován náš cíl, kterým je rozuměn konečný stav, ke kterému směřuje naše úsilí. Dále jsou stanoveny dílčí úkoly k dosažení cíle a způsoby jejích dosažení. Také je důležité popsat vztahy mezi dílčími cíli, a v jakém časovém pořadí mají být řešeny. Je třeba stanovit dostupné zdroje a limity, a jakým způsobem bude reagováno na změny. Výstupem této činnosti pak vzniká plán, který formuluje vytyčené cíle. [1,2,3,4]

Je možné rozlišit několik typů plánování. Z hlediska časového rozvržení se může jednat o plánování krátkodobé, střednědobé a dlouhodobé. Za dlouhodobé plánování jsou považovány plány pro dobu delší než pět let. Na druhé straně krátkodobé plány jsou vytvářeny nejvýše na jeden rok. Plánování v časovém úseku mezi jedním až pěti lety se označuje jako střednědobé. [2,3,5]

Dále lze dělit plánování dle úrovně rozhodovacího procesu na strategické, taktické a operativní. Strategické plánování se orientuje především na dlouhodobý časový horizont, dosažení dlouhodobých cílů a zahrnuje důležité rozhodovací procesy v rámci celku. Jedná se o nejvyšší úroveň. Taktické plánování se zaměřuje na podporu strategie a určuje postupy a dílčí cíle jednotlivých oblastí. Rozhodovací procesy probíhají v rámci částí celku, tedy organizačních jednotek. Toto plánování je na střední úrovni a je více konkrétní. Na nejnižší úrovni plánování probíhá operativní plánování, které je prováděno v krátkých časových úsecích a vychází z taktiky. Operativní plánování je spojeno s vytyčováním cílů dílčích pracovišť a jednotlivců, a také s jejich řízením. Cílem je pak optimalizace zdrojů a maximalizace úspěchu. [2,3,4,5]

Pro plánování je také podstatné analyzovat možný budoucí vývoj. Pro odhady jsou využívány statistické metody nebo expertní odhady na základě minulých a současných dat. Vytvořené prognózy mají následně velký význam při tvorbě dalších strategií a reakcí na možná rizika. [2,6]

Plánování by mělo vést ke zlepšení rozhodovacích procesů. Při této činnosti jsou využívány analýzy možného budoucího vývoje, na základě kterých poté dochází k rozhodování mezi možnými variantami. Dále by spolu měly být jednotlivé plánovací činnosti propojené tak, aby s nimi bylo možné uvažovat jak při plánování v časovém rozmezí, tak pro jednotlivé části a organizační úrovně. Konkrétně se může jednat například o plánování lidských zdrojů.

Také by mělo docházet ke kontrolování splnění dílčích cílů a zpracování odchylek od plánu, abychom mohli plán zdokonalit. Velice důležitou vlastností plánování je pružná reakce. Plán by měl dokázat reagovat na jakékoliv změny, které mohou ovlivňovat chod naplánovaného procesu. Pro vytvoření reakcí je potřeba využít skutečné a plánované hodnoty, a také odhadované hodnoty. Srovnáním odchylek s těmito hodnotami jsou poté vytvářeny analýzy změn, které mohou následně vést ke zlepšení plánu. [2,3,4,6,7]

1.1 Plánování jako manažerská činnost

Plánování patří mezi základní manažerské činnosti. Dochází při něm k vytyčení cílů organizace a určení postupů pro jejich dosažení. Pokud není správně určena strategie a způsoby dosahování cílů, není možné kvalitně realizovat organizování procesů a rozmísťování pracovníků, ani jejich kontrola. Plány by nám měly určovat, jakou činnost, kdo, kdy a jak vykoná. [8]

V organizaci slouží plánování pro její rozvoj. Pomocí něj lze stanovit směr rozvoje, provádět změny v organizaci a pracovat se zdroji. Zajišťuje nám koordinaci a návaznost plánů, jak z časového hlediska, tak i z hlediska struktury organizace a plánů jejich jednotlivých částí. [8]

Aby mohlo docházet ke správnému rozhodování, musí být i manažerská činnost plánována, jelikož se jedná o významnou činnost, která ovlivňuje chod organizace a dosahování jejich cílů. Podle složitosti rozhodovacích procesů vzniká různě složitý plán, dle kterého se plní patřičné úkoly. Plán by měl být založen na reálných možnostech organizace a dostupných zdrojích. Dále by měly být způsoby k provádění plánovaných činností dostatečně popsány a postupy úplné. Součástí plánu by také měly být možné alternativy u prováděných činností, kdy je možno použít různé zdroje pro vykonání dané činnosti. Tvorba plánu je však velmi individuální a každá organizace si sestavuje plány podle svých potřeb tak, aby pro ně byl výstup nejvhodnější. [8]

1.2 Plánování kapacit a pracovníků

Lidské zdroje jsou nezbytné a je to to nejcennější, co má organizace k dispozici. Efektivní plánování má příznivý účinek na chod organizace a přispívá k jejímu vývoji. Při této činnosti dochází k plánování přidělení zaměstnanců na dané pracoviště, jejich počet a rozmístění v čase. Také je s ní spojeno budování kvalitních pracovních podmínek a plánování pracovní struktury a nutné způsobilosti zaměstnanců. Tato činnost také souvisí s plánováním směn

a provozu, a v neposlední řadě s financemi, které jsou nutné pro zajištění zdrojů. Během plánování je zapotřebí, aby docházelo ke kvalitní spolupráci mezi vedením, vedoucími dílčích pracovních jednotek, kontrolory a samotnými zaměstnanci z důvodu možnosti reakce a úpravy plánů. [2]

Plánování lidských zdrojů si klade za cíl vyvážení množství nutných pracovních sil a dostupných zdrojů. To znamená, že je požadováno, aby byl vždy k dispozici dostatek správně rozmístěných pracovníků, ale zároveň byli maximálně využiti, a nedocházelo tak zbytečně k čerpání zdrojů navíc. Účelem tedy je, aby byly zajištěny lidské zdroje v dostatečné kvantitě i kvalitě, které jsou nutné k dosažení cílů organizace. [2,9]

Aby bylo možné plnit požadavky organizace, je třeba plánovat s předstihem a předvídat změny, které mohou nastat. Předpovědi jsou prováděny odborníky na základě informací a analýz a nazývají se prognózy. Nejdůležitějšími prognózami pro plánování pracovníků jsou prognózy zdrojů pracovních sil a prognózy potřeby pracovních sil. Tedy jak už bylo zmiňováno, množství zdrojů, které organizace potřebuje, a kolik jich má k dispozici. [10]

Proces plánování pracovníků může být popsán v několika krocích. Nejprve je potřeba stanovit úkoly, které mají být splněny a vhodně je uspořádat. Následně dochází k odhadu počtu potřebných lidských zdrojů a kvalifikovaných pracovníků, které je možné v blízké době využít k plnění úkolů, tedy kolik jich je k dispozici, a kolik je zapotřebí ještě zajistit. Dále se uskuteční shrnutí celkových požadavků a vytvoří se plán pro pracovní útvar. Pracovní útvar bude na základě plánu získávat zdroje a následně řešit stanovené úkoly. Během plánování je složité odhadnout potřebné zdroje. Na tuto činnost má organizace své personální odborníky, kteří dokážou přizpůsobit plánování potřebám organizace. [10]

2 POUŽITÉ TECHNOLOGIE

V dnešní době existuje velké množství technologií, pomocí kterých je možné řešit nejrůznější problematiku. Vzhledem k tomu, kolik možností se nabízí, je na uvážení autora, pro jaké technologie se ve svém projektu rozhodne. Zpravidla volí ty technologie, se kterými má dobrou zkušenost, zná je nebo u kterých zjistil, že jsou vhodné právě pro jeho řešení. Jedná se především o výběr jazyka a vývojového prostředí (software) a technického vybavení (hardware).

Tato kapitola pojednává o technologiích, které jsou použity jak pro vývoj aplikace Perm3 a PermWeb, tak pro modul kapacitního plánování, který bude součástí těchto aplikací a bude zájmem této práce.

Vzhledem k těmto skutečnostem není nutné provádět průzkum a výběr vhodných dostupných možností. Naopak je nutné použít stanovené technologie a postupy, aby rozšíření do aplikace náležitě zapadlo. Používané technologie jsou nastíněny v následujících podkapitolách.

2.1 Pascal a Delphi

Pascal a Delphi spolu neodlučitelně souvisí. Během dlouholetého vývoje se totiž z původního Pascalu, který byl vytvořený jako výukový nástroj pro studenty programování, vyvinul komplexní nástroj Delphi. Společnost Kvasar využívá tuto technologii k vývoji desktopové aplikace Perm3, a proto je i základním kamenem této práce. [11]

Pascal vznikl jako nástupce vysokoúrovňového jazyka Algol, kdy poskytoval možnost použití dynamických datových struktur a více druhů datových typů. Jeho vývojem se následně zabývala společnost Borland, která vyvinula Turbo Pascal. Tento nástroj mimo původní Pascal poskytl kvalitní vývojové prostředí, díky kterému měla tato verze velký úspěch na trhu. [11]

2.1.1 Delphi

Firma Borland následně vytvořila novou verzi vývojového prostředí, kterou pojmenovala Delphi. Toto prostředí se vyznačuje přístupem Rapid Application Development (RAD), jehož podstatou je tvorba prototypů, které slouží jako náhled řešení a je možné pomocí nich řešit problémové části projektu, jelikož poskytuje okamžitou zpětnou vazbu. Prostředí tedy účinně slouží pro rychlou tvorbu aplikací pomocí grafického rozhraní a výběru komponent,

kdy jednotlivé komponenty usnadňují uživateli práci při definicích funkcí aplikace například reakce na stisk tlačítka či komunikace s databází. [11,12]

Tvůrci proměnili původní Turbo Pascal na objektově orientovaný jazyk (Object Pascal) a doplnili jej o plnohodnotné grafické rozhraní a funkce pro přístup k databázi. Nástroj poskytl velice rychlý kompilátor a formulářový přístup pro tvorbu front-endového obsahu. Je třeba zmínit, že se tvůrci zaměřili na úzké spojení s platformou Microsoft Windows, a proto bylo Delphi uzpůsobeno na vývoj aplikací pro tento systém. [11,13]

Následně došlo k úpravám a vydáním novějších verzí. Postupem času se však společnost Borland začala orientovat jiným směrem a vývoj Delphi převzala firma CodeGear, kterou následně odkoupila společnost Embaracadero Technologies, která se věnuje vývoji dodnes. [11,13]

2.1.2 Embarcadero Delphi XE2

Nyní však k verzi, která je používaná pro řešení této práce. Jedná se nástroj, ve kterém Embarcadero Technologies ukázalo, že se chce vývoji věnovat skutečně naplno. Tato verze poskytuje 64bitový kompilátor Delphi, a také knihovnu FireMonkey pro multiplatformní tvorbu a možnost kompilovat i pro zařízení s operačním systémem iOS. Umožňuje nadále pracovat v kvalitním vývojovém prostředí a využívat grafického rozhraní pro tvorbu formulářových oken za využití komponent. [13,14]

Využívá jazyka Object Pascal, který, jak bylo uvedeno, je rozšířením programovacího jazyka Pascal o objektový přístup. Tento jazyk poskytuje všechny moderní funkce, jako jsou tvorba tříd, rozhraní a jednotek. Vytvořené jednotky mají příponu **.pas**, stejnou příponu mají také formuláře, ty spolu se sebou nesou i soubor **.dfm**, pomocí kterého je možné zprostředkovávat náhled při upravování. Jedná se o vysokoúrovňový jazyk, který je možné jeho vlastnostmi přirovnat k C++, Java nebo C#. Vzhledem k tomu, že byl původní Pascal vyvinut pro studijní účely, je zápis kódu v tomto jazyce čitelný a přehledný. Tuto vlastnost podtrhuje samotné prostředí, které umožňuje standartní zvýraznění částí kódu a strukturování. [13,14,15]

Objektově orientovaný přístup

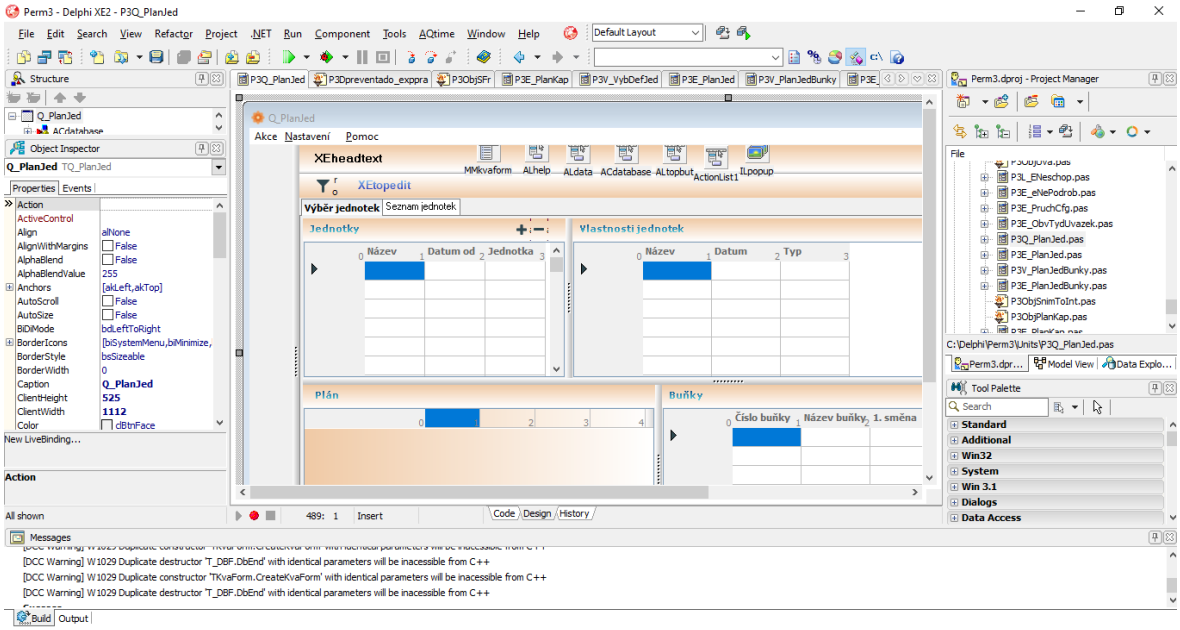
Objektově orientovaný přístup je programovací přístup, který dělí kód do znovupoužitelných částí, zapouzdřuje tyto části, poskytuje možnost dědit jejich vlastnosti a umožňuje odvozovat

různé části kódu na základě předchozích definic. To umožňuje rychlé úpravy kódu při nutnosti změn. Takto definovaná skupina vlastností se nazývá třída a její instancí je objekt, který realizuje konkrétní definice. Objekt obsahuje atributy, které určují data objektu, a metody, které určují, jak je možné s daty objektu pracovat. Data i metody jsou společně zapouzdřeny do jednoho celku, aby k nim nebylo možné přistupovat bez patřičných oprávnění. Práva k přístupu k atributům a metodám se využívají nejčastěji trojího typu. Nejpřísnější je oprávnění `private`, které neposkytuje přístup nikomu jinému než samotné třídě (případně v Object Pascalu třídám v rámci jednotky). Méně přísnou variantou je `protected`, která dovoluje přístup i ze tříd, které jsou od sebe odvozeny nebo v rámci jednotky. Nejvolnější je `public`, která dovoluje přístup odkudkoliv. S oprávněními souvisí dědičnost, což je přední vlastností objektového programování. Organizuje třídy do hierarchické struktury a umožňuje vytvářet nové třídy odvozením od jiné, čímž nová třída získává vlastnosti třídy, od které je odvozena, a může ji dále rozšiřovat a konkretizovat. Z toho vychází další vlastnost objektově orientovaného programování, a to je polymorfismus, který umožňuje používat jednotné rozhraní pro práci s metodami objektů. To znamená, že mají obecně definovány metodu u rodiče, ale každý potomek ji provádí s vlastními obměnami. Díky tomu je možné volat stejnou metodu s různými parametry. [16,17,18,19]

Je zde vidět jistá podobnost s reálným světem, kdy jednotlivé předměty (objekty) disponují svými vlastnostmi (atributy) a mohou vykonávat různé činnosti (metody). [18]

Vývojové prostředí

Vývojové prostředí Delphi XE2 poskytuje vizuální vývoj založený na komponentách a obousměrný RAD, a také vysoce výkonný přístup k databázím. Je možné snadno vytvářet aplikace, které jsou vysoce interaktivní a dokáží pracovat s velkým množstvím dat. Aplikace je možné od této verze překládat pro další typy zařízení (Mac a mobilní zařízení). Velká výhoda tohoto prostředí je bohatá dokumentace, která pokrývá spoustu let předchozího vývoje a je možné vyhledat mnoho informací, pokud uživatel narazí na nějaký problém. [20]



Obrázek 1 – Vývojové prostředí Embarcadero Delphi XE2

Na obrázku 1 je zobrazena podoba prostředí, které se skládá z hlavního panelu menu, který obsahuje mnoho funkcí pro tvorbu projektu a nastavení. Na panelu se také nachází rychlé menu pro ukládání, práci s jednotkami či debugging. Dále je zde výčet dostupných komponent pro tvorbu formulářů *Tool Palette* či okno zpráv, které slouží také pro zpětnou vazbu uživateli. Užitečným prvkem je *Object Inspector*, pomocí kterého je možné staticky upravovat vlastnosti prvků formuláře a rychle definovat jejich obslužné akce, které nás pomocí něj odkážou do předdefinovaného místa v kódu. Nejdůležitější část vývojového prostředí je samotný editor programového kódu, ve kterém dochází k tvorbě funkcionality aplikací, společně s editorem formulářů.

Programová okna se nazývají formuláře. Slouží pro grafický výstup aplikace, ovládání a jednoduché zpracování vstupů uživatele. Do formuláře je možné přidávat mnoho prvků (komponent) z *Tool palette* jako jsou tlačítka, textová pole či dialogy. Tím, že jsou komponenty předem definovány, stačí pouze doplnit jejich funkcionalitu. Tvorba aplikací je rychlá a snadná díky editoru programových oken, který se nazývá *Form Designer*, a samotnému náhledu na design formulářů. Spolupráce kódu formuláře s *Form Designerem* je zajištěna již zmiňovaným souborem s příponou **.dfm**. [15]

2.2 Dynamic-link library

Dynamicky připojitelné knihovny, zkráceně DLL, jsou moduly, které obsahují funkce a data, které mohou být použity dalšími programy či knihovnami, které s knihovnou pracují. Knihovny DLL umožňují modularizaci aplikací, díky čemuž můžou být jejich funkce snadněji aktualizovány a rozšiřovány. Pomáhají snižovat režii paměti pro více aplikací, jelikož sdílejí kód knihovny, data knihovny si nesou každá ve své vlastní kopii. Tato podkapitola se věnuje popisu funkcionality dynamicky připojitelných knihoven pro systémy Windows, jelikož je pro vytváření této práce využíváno právě tohoto systému. [21]

Knihovny DLL mohou definovat dva typy funkcí: exportované a interní. Exportované jsou určeny k volání jinými aplikacemi a moduly, a také v rámci vlastní knihovny. Interní funkce jsou zpravidla určeny pro volání uvnitř knihovny. Data knihovny jsou použita převážně pro její funkce, nicméně mohou být exportována, a také čtena nebo přepsána jiným modulem. [21]

Dynamické připojení slouží k tomu, že umožňuje modulu zahrnout jen ty informace, které jsou potřeba k nalezení exportované funkce v okamžiku načítání či spuštění. Rozdíl mezi statickým propojením je v tom, že u statického dochází ke zkopírování kódu funkce pomocí linkeru do každého modulu, který funkci volá. [22]

V systémech Windows existují dva typy volání funkcí DLL. Prvním z nich je dynamické připojení při načítání modulu, při kterém dochází k explicitnímu volání exportovaných funkcí DLL, obdobně jako místních funkcí. Je vyžadováno, aby byl modul propojen s knihovnou spravující import, která poskytuje potřebné informace pro načtení DLL a vyhledání exportovaných funkcí při načtení aplikace. Druhým způsobem je dynamické připojení modulu za běhu, pro který se využívají funkce *LoadLibrary* či *LoadLibraryEx*. Tyto funkce jsou volány procesem, který chce knihovnu použít a slouží k načtení knihovny do adresného prostoru procesu. Po načtení knihovny je možné využít funkci *GetProcAddress*, která slouží pro získání exportovaných funkcí nebo proměnných specifikovaných v DLL (jejich adres). Pomocí těchto adres dokáže modul funkce volat. Tím je odstraněna potřeba propojení s knihovnou spravující import. Pro následné uvolnění knihovny z paměti je možné použít funkci *FreeLibrary*. Volání těchto funkcí může být použito například u rozšíření funkcionality pomoci DLL u spustitelných aplikací. [22]

Dynamické propojení má výhody v tom, že načítá stejnou DLL na stejnou základní adresu a sdílí jednu kopii DLL ve fyzické paměti pro více procesů, čímž dochází k úspoře systémové paměti. Pokud dojde ke změně funkce v DLL, ale zůstanou argumenty, volání a výstupy stejné, není nutné překompilovat aplikace využívající knihovnu funkcí, konvence volání a návratové hodnoty. Pokud je dodržena konvence volání, je možné používat knihovnu pro programy psané v různých jazycích. Konvence volání určuje, v jakém pořadí vkládá funkce argumenty do zásobníku, která funkce je zodpovědná za vyčistění zásobníku nebo zda dochází k předání některých argumentů do registrů. [23]

2.3 Extensible Markup Language (XML)

XML je nástroj pro práci s daty, především jejich strukturaci, uchování a přenos. Zkratka vychází z anglického názvu eXtensible Markup Language, což v překladu znamená rozšiřitelný značkovací jazyk. Rozšiřitelný v tom slova smyslu, že je možné rozšiřovat strukturu dokumentu a přidávat data, aniž by došlo ke ztrátě funkcionality. XML soubory je možné využívat pro přenos struktur a dat. Této vlastnosti lze využít pro manipulaci se vstupními a výstupními daty DLL knihovny, pro kterou mohou řetězce ve formátu XML tímto způsobem vhodně sloužit. To je důvodem použití XML v této práci. [24]

Jedná se o značkovací jazyk podobný HTML (Hypertext Markup Language) s tím rozdílem, že HTML bylo navrženo pro zobrazování dat se zaměřením na jejich vzhled, kdežto XML se zaměřuje na jejich obsah. Na rozdíl od HTML nemá XML žádné předdefinované značky, autor definuje značky i strukturu dokumentu sám. Data jsou uložena v prostém textu, díky čemuž je XML nezávislý na softwaru a hardwaru. Formát XML je snadno čitelný, jak pro člověka, tak počítače či čtecí zařízení. Lze pomocí něj zajistit snadnou komunikaci mezi aplikacemi a předání dat. [24]

Nyní je uveden příklad, jak může vypadat obsah XML souboru.

```
<zprava>
  <komu>Taťka</komu>
  <odkoho>Oliver</odkoho>
  <predmet>Pozvánka</predmet>
  <textzpravy>
     Ahoj, večer v šest Vás zveme k nám domů na večeři.
  </textzpravy>
</zprava>
```

U příkladu je vidět, že je obsah zabalený mezi značkami (tagy) `zprava`, které zprávu zapouzdřují, a její jednotlivé položky poté mají také své vlastní značky. Díky tomu je možné

snadno rozeznat, jaký význam data mají, a jaké prvky zpráva obsahuje. Takovýchto zpráv či zapouzdřených úrovní značek může být v souboru mnoho a záleží na autorovi, jakou strukturu vytvoří.

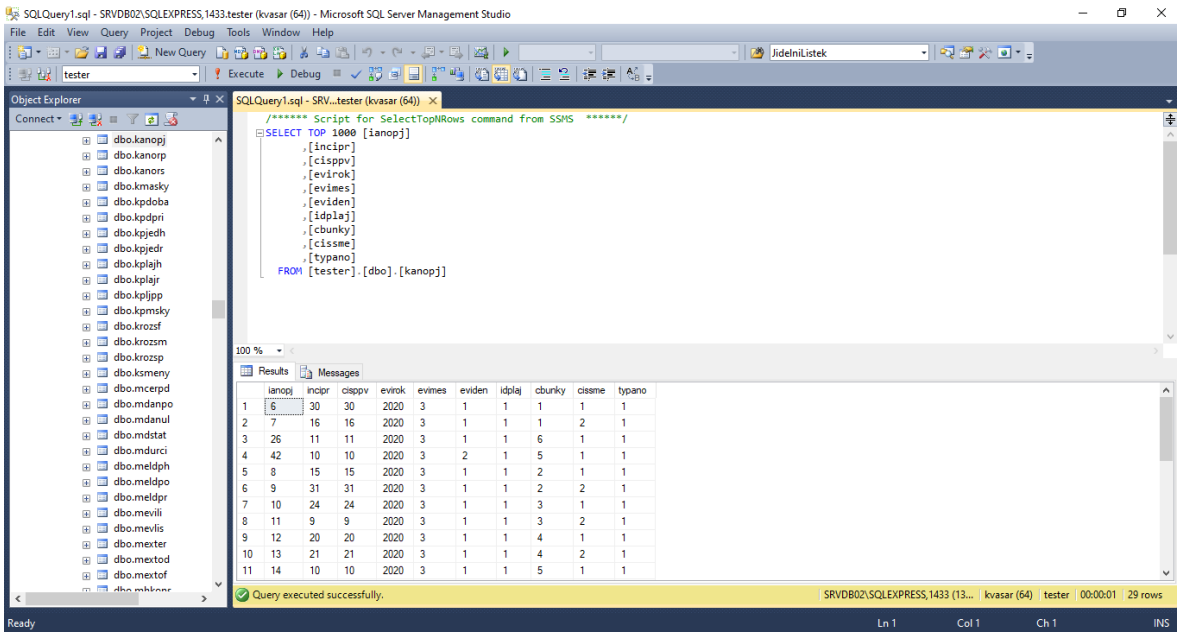
2.4 Microsoft SQL Server (MSSQL)

Microsoft SQL Server je relační systém řízení báze dat. Jedná se o jeden ze tří vedoucích produktů na trhu vedle Oracle Database a IBM DB2. Tento databázový systém byl v rámci této práce použit ve společnosti Kvasar pro práci s daty, které mají souvislost s aplikací Perm3 a je s nimi v aplikaci manipulováno. Základními funkcemi relační databáze jsou vytváření, čtení, aktualizace a mazání, které jsou označovány jako CRUD. Nejběžnějším způsobem pro přístup k datům relační databáze je jazyk SQL, který se v tomto systému k těmto účelům používá. Relační databáze ukládá data ve formě tabulek, kdy řádek vyjadřuje jeden záznam a sloupec je konkrétním prvkem záznamu. Struktura je založena na propojení souvisejících datových prvků mezi tabulkami bez nutnosti ukládání redundantních dat. [25]

Hlavní součástí Microsoft SQL Server je databázový engine SQL Server, který řídí ukládání, zpracování a zabezpečení dat. Engine sestává ze zpracování příkazů a dotazů, a paměťového modulu, který spravuje databázové soubory, tabulky, indexy, datové vyrovnávací paměti a transakce. Uložené procedury či spouštěče jsou také vytvářeny a prováděny pomocí databázového enginu. SQL Server je vázán na Transact-SQL (T-SQL), což je speciální implementace SQL vytvořená společností Microsoft, která do standardního jazyka přidává sadu patentovaných programovacích rozšíření. [25]

2.4.1 Microsoft SQL Server 2014 Management Studio

Systém pro správu databáze, který poskytuje kvalitní datové úložiště a prvky pro její tvorbu tak, jak společnost Microsoft navrhnula. Dotazy a příkazy databázi probíhají pomocí jazyka SQL, také je možné, vytvářet je pomocí designeru. Kromě prohlížení, umožňuje designer například modifikaci tabulek, vytvoření pohledů na databázi na základě klíčů či export a import dat.



Obrázek 2 – Grafické rozhraní MSSQL 2014 Management Studio

Na obrázku 2 lze vidět, že se prostředí skládá z hlavního panelu, který obsahuje navigační menu a panel pro rychlou práci s databází. Dále se zde nachází průzkumník databází a tabulek *Object Explorer*, který poskytuje přehled vytvořených a připojených databází. Největší část zabírá editor pro tvorbu příkazů a část pro zobrazení výsledků a zpráv. Práce s prostředím je jednoduchá a intuitivní, a vzhledem ke svému dlouholetému vývoji poskytuje také kvalitní dokumentaci, což napomáhá při řešení jakýchkoliv problémů.

2.4.2 Structured Query Language (SQL)

Structured Query Language je překládáno z anglického jazyka jako strukturovaný dotazovací jazyk. Jak název napovídá, jazyk pracuje za základě strukturovaných dotazů respektive příkazů, které používá při práci s relačními databázemi, které jsou popsány jako soubor dat, která jsou propojena pomocí relací a řízena některým ze systému řízení báze dat. Jedná se o interpretovaný jazyk, což znamená, že k vykonání jeho kódu postačí pouze interpret. Tím je speciální program, který je schopný zajistit provedení zdrojového kódu bez nutnosti překladu. [26,27]

Mezi základní příkazy jazyka patří příkazy, které slouží pro definici dat, manipulaci s daty a řízení dat. Definice dat slouží pro tvorbu struktury databáze a je možné pomocí nich vlastnosti struktury jak vytvářet, tak i měnit či odstranit. Mezi tyto příkazy patří například CREATE, ALTER či DROP. Se samotnými daty je poté možno pracovat pomocí příkazů SELECT, INSERT, UPDATE či DELETE. Tyto příkazy poskytují možnost data z databáze

získat nebo je upravit. Pro řízení dat jsou používány příkazy, jako jsou START TRANSACTION, COMMIT, ROLLBACK či REVOKE, které se používají pro řízení transakcí a nastavení přístupových práv. Dále je možné využít příkazy pro tvorbu podmínek, řazení dat, slučování tabulek či aritmetické výpočty. Pokud bychom chtěli specifikovat výběr dat, pro které bude daný příkaz platný nebo data seřadit, je možné použít klauzuli za příkazem WHERE a příkaz SORT pro seřazení. Další příkazy jazyka slouží například pro správu databáze, nastavení času či kódování. [27]

Jazyk má jednoduchou syntaxi, která se skládá z příkazu a jeho argumentu. Je možné vytvářet i vnořené a složitější dotazy, při kterých je třeba dodržovat pravidla, strukturu dotazu a pořadí příkazů. Výhodou jazyka je, že pokud se provedena chyba v zápisu dotazu, interpret nás upozorní a pomůže nás nasměrovat do místa, kde k chybě dochází. Při tvorbě dotazu se na jeho začátku nachází některý z výše uvedených typů příkazů společně s argumenty, za nimi mohou být následně umístěny podmínky, příkazy pro slučování či řazení. [27]

Nyní uveďme, jak může vypadat struktura SQL příkazu.

```
SELECT jmeno, prijmeni, goly, pocetzapasu
FROM hraci
WHERE sport='hazena'
ORDER BY goly DESC
```

Takový příkaz by například zajistil výpis seznamu hráčů házené (jejich jméno, příjmení, počet vstřelených gólů a počet odehraných zápasů) z tabulky `hraci` a seznam výsledných řádků by se seřadil od největšího počtu vstřelených gólů po nejmenší. Ke zpracování výsledků dotazu slouží kurzor. Pomocí něj je možné přistupovat k výsledným řádkům jednotlivě a provést jejich sekvenční zpracování. Je nutné deklarovat SQL dotaz, jehož výsledky mají být pomocí něj procházeny. Jeho práce je zahájena okamžikem jeho otevření (open). V této chvíli začíná procházet jednotlivé výsledné řádky SQL dotazu a posouvá se mezi nimi (fetch). V okamžiku nalezení konce seznamu, kdy už se žádná další položka nevyskytuje, dochází k ukončení jeho činnosti a je nutné jej zavřít (close). Jakmile je kurzor uzavřen je možné ho využít ke zpracování výsledků dalšího SQL dotazu. [28]

Jazyk SQL je velmi často používaný a práce s databází je pomocí něj jednoduchá a srozumitelná.

2.5 REST API

REST je zkratka pro Representational State Transfer a jedná se o architekturu pro distribuované prostředí, která poskytuje přístup k datům pomocí standartních HTTP metod, jako jsou například GET, POST či DELETE. Je založena na přístupu klient-server, uživatelské rozhraní je oddělené, a je pomocí ní možné ovládat i stavy aplikací. Webové aplikace využívají REST rozhraní, jelikož je možné je pomocí frameworků jednoduše vytvořit, a poskytnout tak jednoduchý přístup k funkcím či datům poskytovaných webovou aplikací. REST API (Application Programming Interface) je bezstavové, to znamená, že si neudržuje žádný stav, ani informace o spojení pro další vyhodnocování požadavků. Zkratka API slouží pro označení aplikačního programovacího rozhraní, které je souborem prvků a procedur aplikace, které je možné využívat skrze toto rozhraní, a určuje způsob, jak k nim správně přistupovat. Tato podkapitola se věnuje REST API z důvodu využití společností Kvasar pro implementaci jejího webového API. [29,30,31,32]

2.5.1 Hypertext Transfer Protocol (HTTP)

Hypertext Transfer Protocol je protokol pro přenos dokumentů v rámci světové sítě Internet. Byl navržen pro komunikaci mezi webovými prohlížeči a serverem, kdy dochází na jedné straně k zasílání požadavků (klient) a na druhé straně k jejich zpracování (server). Slouží tedy ke komunikaci typu klient-server dvou zařízení, mezi kterými dochází k přenosu požadavků, odpovědí a dokumentů. Není však používán pouze za účelem původně zamýšleného přenosu hypertextových dokumentů (HTML), přenáší se pomocí něj mnoho typů dokumentů, od webových stránek a souborů XML, po multimediální soubory. Server neuchovává žádné informace mezi dvěma požadavky, proto je tento protokol nazýván bezstavový. Nejčastěji používanými požadavky jsou GET, POST, PUT a DELETE, které slouží pro zisk, zápis nebo smazání dat umístěných na serveru a specifikovaných v požadavku. [33]

2.5.2 Representational State Transfer

Architektura REST je založena na principu klient-server. Klient zahajuje komunikaci vytvořením požadavku a server odpovídá na požadavky za účelem poskytnutí přístupu ke svým službám a datům. Komunikace mezi klientem a serverem je bezstavová, čímž je zajištěno, že mezi vyřizováním jednotlivých požadavků nejsou udržovány žádné informace ani stav komunikace, a každý požadavek od klienta musí obsahovat všechny potřebné informace k porozumění požadavku. Jednotlivé požadavky jsou tedy zpracovávány nezávisle na sobě.

Další vlastností této architektury je důraz kladený na jednotné rozhraní. To zajišťuje podporu nezávislého vývoje a manipulaci s daty především ve standardizované podobě než v podobě specifické pro danou aplikaci. Data a služby jsou považovány za zdroje a přistupuje se k nim pomocí identifikátorů URI (Uniform Resource Identifiers). Zdroje jsou využívány pomocí sady jednoduchých operací, kdy si klienti a servery vyměňují reprezentace zdrojů pomocí standardizovaného rozhraní a protokolu, zpravidla HTTP, a pracují se zdroji pomocí operací CRUD. Klient ovšem nepracuje přímo se zdroji, ale s některou z jejich reprezentací, takže k jejich obsahu lze přistupovat v různých formátech, jako je HTML, XML či JSON. Tímto způsobem je možné od serveru získávat data, využívat jeho služby pro zpracování dat či měnit stavy aplikace. Data jsou přenášena nejčastěji pomocí formátu JSON, který podobně jako XML poskytuje možnost vytvořit libovolnou strukturu. Zápis dat je objektového charakteru a položky prvku mohou být různých datových typů. Stavy aplikací mohou být určovány pomocí URL z odpovědi ze serveru či zvolením vhodné reprezentace pro potřebu aplikace a použitím API. [29,30,31]

Webové REST API je webová služba implementovaná pomocí protokolu HTTP a zásad REST. Definuje sadu funkcí, pomocí kterých je možné provádět požadavky a získávat od aplikace odpovědi. Jednoduchým příkladem je zadání hodnoty na patřičné místo v URL a odeslání, jako odpověď se nám vrací výsledek na základě zvolené hodnoty. Slouží jako rozhraní mezi částmi aplikace pro zjednodušení a poskytuje služby, procedury či volání svých dostupných vlastností pomocí jednotného rozhraní. Jedná se o soubor zdrojů, které mají definovány základní vlastnosti, pro funkcionalitu. První z nich je URI , který určuje zdroj, se kterým budou prováděny interakce, například http://sluzby.cz/zdroje/. Dále je třeba určit u procedur pro práci s daty, jaký typ reprezentace dat bude webová služba podporovat, nejčastěji se jedná o JSON a XML, ale může to být jakýkoli jiný platný formát. Důležité je popsat sadu operací podporovaných webovou službou pomocí metod HTTP (např. POST, GET, PUT nebo DELETE). Díky rozdělení aplikace na poskytování jednotlivých služeb je vytvořena modulární struktura, a tím se stává tvorba aplikace flexibilnější. [30,32]

Přístup k API a možnost jeho využívání je vhodné zabezpečit metodami autentizace a autorizace, například pomocí přihlašovacích údajů, aby nedocházelo k jejich zneužití či zahlcení nežádoucími a neoprávněnými požadavky. Při těchto metodách dochází k ověření identity žadatele o službu a k ověření jeho oprávnění službu využívat.

2.5.3 .NET Core

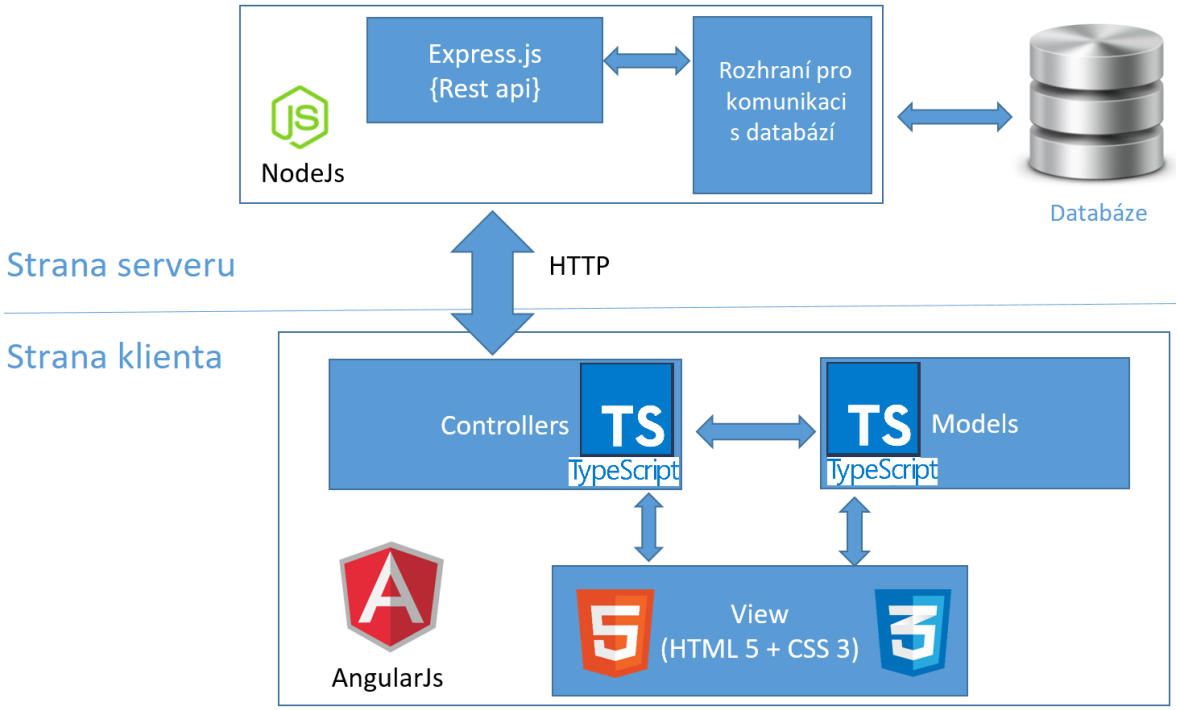
REST API je možné vytvořit různými způsoby a technologiemi. Jednou možných variant může být tvorba v prostředí Visual Studio a za použití technologie .NET. Tento způsob poskytuje automatické vytvoření základní struktury Model-View-Controller (MVC), která slouží pro oddělení aplikační logiky a grafického rozhraní aplikace. Vytvořenou strukturu je následně třeba doplnit o všechny nutné vazby a funkcionalitu. V sekci modelu dochází k definici tříd s vlastnostmi, na základě kterých je možné určit, ke kterým prvkům databáze přistupovat. View se stará o definici zobrazení v návaznosti na model, a tedy grafickou podobu aplikace. Controller slouží pro obsluhu požadavků a odpovědí a reaguje na uvedenou URL. Je možné definovat služby, rozhraní, cesty, způsoby ukládání a práce s daty. Tvorba kvalitního API není jednoduchá záležitost a může zabrat mnoho času. [34]

V návaznosti na výše popisované knihovny DLL (podkapitola 2.2) tato technologie umožňuje vytvoření API způsobem, který umožňuje standardní využívání knihoven. Aby bylo možné s knihovnou a jejími procedurami pracovat, je nezbytné knihovnu nejprve načíst. Pro tyto účely je třeba používat standardní funkce operačního systému pro práci s knihovnami, jako je například *LoadLibrary*. K volání těchto funkcí je však nutné použít takzvaný nespravovaný kód, při jehož používání je programátor zodpovědný za správu paměti sám oproti spravovanému kódu, kde je kontrola zajištěna. Technologie .NET Core respektive C# tuto správu paměti automaticky provádí a používá tedy spravovaný kód. Dovoluje ovšem i použití nespravovaného kódu, a tak je možné použít standardní funkce pro práci s knihovnami. Po ukončení práce s knihovnou je nutné veškerou alokovanou paměť uvolnit (*FreeLibrary*). [35]

2.6 Webová aplikace - Angular (TypeScript, ES6)

Webová aplikace je technologie, která je použita pro plánovací část tohoto systému, proto je nutné nastínit tento pojem a popsat vlastnosti při použití technologie Angular. Webová aplikace se skládá ze statických a dynamických webových stránek. Statické stránky nemění svůj obsah a jsou zobrazovány beze změny. Struktura dynamických stránek (HTML) je generována přímo na straně klienta pomocí zdrojového kódu. Děje se tak pomocí aplikačních knihoven a interpretace aplikační logiky webových stránek ve webovém prohlížeči. Data jim však poskytuje server, se kterým komunikují na základě HTTP požadavků webového prohlížeče. Využívají především definované webové API, jejíchž procedury a data jsou využívány ke zpracování a zobrazení obsahu webových stránek v prohlížeči. Díky této vlastnosti

je možné zobrazovat uživateli proměnlivý obsah a dynamicky reagovat na požadavky. Webová aplikace tedy může obsahovat stránky bez stanoveného obsahu a výsledný obsah stránky se může lišit vzhledem k předchozím provedeným akcím. O provádění požadavků a reakce na uživatelovi akce se na straně klienta stará webový prohlížeč a o zpracování požadavků aplikace na data či využití funkce API se stará aplikační server. Webové stránky ve webové aplikaci jsou tedy především dynamické a reagují na činnost uživatele. [36]



Obrázek 3 – Schéma zpracování požadavku webovou aplikací [37]

Spolupráce částí ve webové aplikaci probíhá principiálně dle obrázku 3. Na straně klienta je funkcionalita rozdělena dle modelu MVC na zobrazovací prvky View, které slouží pro interaktivní prezentaci dat a stavů aplikace reprezentovaných v části Models. Reakce na uživatelské akce a řízení změn aplikace jsou zpracovávány ovladači Controllers. Ty zajišťují také komunikaci se serverovou částí a provádí požadavky skrze API na data z databáze. [36]

Technologie **Angular** je open-source framework pro tvorbu klientských aplikací za použití HTML, TypeScriptu a šablon. Je možné pomocí něj vytvářet webové aplikace, které lze spustit za použití webovém prohlížeči na jakémkoliv zařízení. Implementuje základní a volitelné funkce jako sadu TypeScript knihoven, které je možné importovat do funkcionality vytvářených aplikací. [38]

Základními stavebními bloky Angular aplikace jsou *NgModules*, které poskytují kompilační kontext pro komponenty. Komponenty definují *views* (pohledy), které lze chápat jako sadu

obrazovkových prvků, které je možné vybírat a upravovat dle patřičné logiky aplikace a dat. Využívají *services* (služby) poskytující specifické funkce nepřímo související s *views*, mohou být vneseny do komponent jako závislosti (dependencies), čímž dělají kód více přehledný a efektivní. Komponenty i služby jsou třídy s dekorátory, které označují jejich typ a poskytují metadata, která určují, jak se třídami pracovat. Na základě metadat komponenty a šablony je definován a vytvořen *view*. Šablona kombinuje běžný HTML s Angularovými direktivami a značkami, pomocí kterých dokáže Angular upravit HTML před výsledným vykreslením. Metadata třídy služeb poskytují informace, které Angular potřebuje, aby bylo možné poskytnout služby komponentám prostřednictvím vnesením závislostí (dependency injection). Aplikace je definována sadou *NgModules*, které shromažďují související kód do funkčních sad a vychází z kořenového modulu, který umožňuje zavádění více funkčních modulů. Pomocí komponent aplikace je obvykle definováno mnoho *views*, které jsou následně uspořádány hierarchicky. Angular umožňuje jednoduše definovat mezi *views* navigační cesty, díky čemuž vznikají jednotlivé přechody a struktura dle požadavků a logiky aplikace. [38]

Moderním přístupem pro chod aplikace je **Lazy-Loading**. Dokáže urychlit inicializaci aplikace a její následující činnost. *NgModules* jsou totiž defaultně nastaveny tak, aby se při inicializaci aplikace načítaly všechny, i když nejsou okamžitě zapotřebí. Pro menší aplikace to nemusí být až takový problém, ale u rozsáhlých aplikací může docházet k načítání velkého množství počátečních dat, zbytečnému zatěžování a delší době načítání. Tento problém řeší Lazy-Loading. Dochází při něm k rozdělení hlavního modulu na menší části a jsou načítány pouze ty nejdůležitější. Těmto částem je třeba odstranit schopnost okamžitého načítání a importu při inicializaci. Následně je každému modulu vytvořen směrovací modul, který má definovánu cestu pro načtení modulu a komponentu, která se zde nachází. Aby byly tyto komponenty načítány způsobem Lazy-Loading, je třeba hlavnímu směrovacímu modulu aplikace uvést informace o cestě ke komponentě, a specifikovat její import v parametru *loadChildren*. Při sestavování aplikace je poté možné vidět rozdělení do těchto menších částí. Při inicializaci aplikace se však načtou jen hlavní požadované části, další části jsou importovány až v případě, že je vyžadována jejich funkcionalita. [39]

TypeScript je open-source objektově orientovaný kompilovaný programovací jazyk, který je možno použít pro jakýkoliv prohlížeč, server či operační systém. Může běžet v jakémkoli prostředí, na kterém běží JavaScript. Je nadmnožinou JavaScriptu a rozšiřuje ho o objektově orientovaný přístup a možnost používání tříd, modulů, hlavičkových souborů či dědičnosti.

Díky tomu je možné používat externí knihovny, jako jsou například jQuery nebo Node.js. Při kompilaci je převáděn do standartního JavaScriptu. Sestává ze tří částí: jazyka, kompilátoru a jazykové služby. Jazyk je tvořen syntaxí, klíčovými slovy a typovými anotacemi, kompilátor převádí instrukce do ekvivalentních instrukcí JavaScriptu. Jazykové služby poskytují vrstvu pracující s vývojovými prostředími, která podporuje typické operace používané v editoru jako je například doplňování příkazů, našeptávání, formátování a popis kódu nebo jeho zvýraznění na základě typu či kontextu. [40,41]

Funkce jazyka TypeScript, jsou ztotožněny se specifikací EcmaScript 6, což je standardizovaná specifikace skriptovacího jazyka, tedy oficiální specifikace pro JavaScript. Také ovšem zahrnuje funkce navíc, jako jsou typové anotace či generika, které už nejsou součástí ES6. [40,41]

Při vývoji je standardní využít **Angular Material Design**, což je knihovna komponent uživatelského rozhraní určená pro Angular. Je založena na principech Google Material Design a je sestavena a optimalizována týmem vývojářů Angular pro bezproblémovou integraci. Pomocí knihovny je možné vytvářet uživatelské rozhraní za použití komplexních moderních komponent, které jsou otestovány na moderních webových prohlížečích a platformách. Komponenty Angular Material Designu pomáhají při vytváření kvalitních, atraktivních, funkčních a rychlejších webových aplikací. Zároveň zajišťují zachování moderních principů webového designu, jako jsou přenositelnost napříč webovými prohlížeči a nezávislost zařízení. Webové stránky vytvořené Angular Material Designem jsou plně kompatibilní s mobilními zařízeními i PC a komponenty jsou navrženy tak, aby se zobrazovaný obsah vešel do jakékoliv obrazovky. Poskytuje běžné komponenty uživatelského rozhraní, jako jsou tlačítka a textová pole, ale také nové prvky pro prezentaci dat a interakci například karty a dialog. Všechny prvky jsou však upraveny tak, aby dodržovali zásady Material Designu. [42,43]

2.7 Reactive Extensions for JavaScript (RxJS)

RxJS je zkratkou pro Reactive Extensions for JavaScript, což v překladu znamená reaktivní rozšíření pro JavaScript, kdy slovo reaktivní je možné chápat ve smyslu reakce na změny. Tato JavaScriptová knihovna poskytuje koncept reaktivního programování softwarových aplikací. Jedná se o paradigma asynchronního programování, které se týká datových toků a reagování na změny, ke kterým dochází, například změna kontextu, kliknutí, načítání dat a další. Nejedná se tedy o typický způsob psaní softwaru, kdy je explicitně psán kód pro

obsluhu takových změn. Knihovna využívá *Observable* objekty, které usnadňují vytváření asynchronního kódu a slouží pro komunikaci mezi vydavateli (producenty) a odběrateli dat. Definují funkce pro poskytnutí dat, ale funkce nejsou vykonávány, dokud se o ně odběratel nepřihlásí, následně obdrží oznámení o dokončení funkce a dostává data k dispozici. Jedná se o sekvenci dat, které přichází jako odpověď na požadavek asynchronně v čase. *Observable* může doručovat jakýkoli typ dat v závislosti na kontextu. RxJS poskytuje implementaci typu *Observable* a pomocné funkce pro jejich vytváření a práci s nimi. Jako *Observables* je možné vytvořit například události, časovače či přísliby (promises), které slibují dodání požadovaných dat, jakmile budou k dispozici. Dále je možné využít obslužné funkce pro převod existujícího kódu pro asynchronní operace na *Observable* typ, mapování hodnot na různé typy, filtrování datových toků či jejich skládání. Sekvence *Observables* (požadavky) lze snadno řetězit, konfigurovat či rušit. V kombinaci s HTTP se využívá pro zpracování odpovědi. HTTP odpověď je v tomto případě jedna položka, jejíž obsah je doručen asynchronně. [44,45,46]

Pomocí těchto přístupů je pro webovou aplikaci možné asynchronně pracovat s daty. Díky tomu lze posílat požadavky API a zpracovávat asynchronní odpovědi a data bez zamrzání aplikace a synchronního čekání na odpověď. [44,45,46]

2.8 Perm3 a PermWeb

Firma Kvasar, spol s r. o. se specializuje na vývoj softwaru pro zpracování mezd a personalistiku. Především se jedná o systém Perm3 a jeho nadstavbu PermWeb. Jejich podstatnou funkcionalitou je výpočet mezd, zpracování importů, docházky a věcí s ní spojených (pracovní cesty, dovolená) a tvorba rozvrhů směn a exportů na základě zpracovaných dat. [47]

2.8.1 Personalistika a Mzdy – verze 3.0

Systém Perm3 je mzdový a personální systém, který poskytuje zpracování mezd dle platných legislativ České i Slovenské republiky, personální řízení, umožňuje spravovat docházku, organizační strukturu firmy, dovolenou či dovednosti zaměstnanců a spoustu dalších subsystémů. Grafické rozhraní je tvořeno přehlednou strukturou oken (formulářů), které mezi sebou nesou řadu automatizovaných vazeb, díky kterým je možné provázat data mezi formuláři a kontrolovat logickou správnost a úplnost dat. [47]

Systém je postaven na technologií *client-server*, umožňuje tak komunikaci s databázovými servery MSSQL, Oracle nebo Sybase. Jeho vývoj probíhá ve výše uvedeném Embarcadero

Delphi XE2 s využitím MSSQL. Také integruje MS Office, kdy umožňuje například tvorbu výstupních souborů patřičného formátu či import dat z takových souborů. Je možné ho provozovat na stolním PC či pomocí terminálového serveru a poskytuje nastavení přístupových práv pro různé uživatele. [47]

2.8.2 PermWeb

PermWeb je webový klient, který slouží jako nadstavba systému Perm3 pro zpřístupnění dat systému Perm3 širšímu spektru uživatelů jako jsou manažeři či samotní zaměstnanci. Pomocí internetového prohlížeče je možné se do aplikace přihlásit a následně mít přehledně dostupná data ze systému Perm3, respektive z příslušné databáze. Výhodou webové aplikace je, že je dostupná bez jakékoliv instalace ve webovém prohlížeči. Na základě role uživatele je možné poskytnout uživateli jen konkrétní funkcionality, jako je základní prezentace dat, podávání žádostí, editace údajů uživateli s vyšším oprávněním či uskutečňování schvalovacího procesu. [48]

Jedná se o modulární systém, který je vyvíjen jako webová aplikace pomocí technologie Angular. Společně s ní je vytvořeno rozhraní **PermWebAPI** za použití technologie .NET Core. Webová aplikace využívá PermWebAPI pro zajištění uživatelem požadované funkcionality. Jednotlivé moduly, na které je aplikace rozdělena, jsou používány zvlášť a načítány metodou Lazy-Loading pouze tehdy, kdy jsou zapotřebí. Tyto moduly využívají pro načtení či zpracování dat již zmíněné PermWebAPI. API dokáže pracovat s dynamickými knihovnami a ty dokáží zpracovávat data z příslušné databáze, proto jsou pro jednotlivé rozšíření aplikace vytvářeny právě tyto DLL knihovny. Tvorba jednotlivých DLL knihoven pro PermWebAPI probíhá v prostředí Embarcadero Delphi XE2. Modulárnost systému umožňuje vybírat, přidávat či poskytovat nové funkcionality dle požadavků uživatelů. DLL knihovny slouží ke zpracování nebo poskytnutí dat na základě předložených vstupních požadavků směrem k PermWebAPI definovaných ve vstupním XML řetězci. Na základě hlavičky popsané ve vstupním XML řetězci je použita odpovídající DLL knihovna pro obsluhu. Po dokončení zpracování je knihovnou vytvořen výstupní XML řetězec, který je následně zpracován a data výsledného výstupu jsou patřičně zobrazena. [48]

Přístup k funkcím PermWebAPI je zabezpečen autentizací a autorizací. K ověřování uživatelů dochází na základě tokenů. Jedná se o short-lived Access Tokeny (krátká doba platnosti) a long-lived Refresh Tokeny (dlouhá doba platnosti). Access Token se používá pro umožnění přístupu uživateli k API. K jeho vytváření dochází na vyžádání při autentizaci uživatele.

Po úspěšné autentizaci dochází k vygenerování příslušného tokenu a jeho předání směrem k uživateli. Access Token je využíván jako ověření při každém dalším požadavku na přistup k API. Token je soubor informací, které jsou API ověřovány, že nositel tokenu byl autentizován pro přístup k jednotlivým částím API a má oprávnění provádět specifické akce v rozsahu určeném tokenem. Také je třeba dodržet stanovené bezpečnostní opatření pro ověřování tokenů, aby bylo možné důvěřovat jejich obsahu. [49]

Dobu platnosti Access Tokenu lze libovolně zvolit v závislosti na bezpečnostních požadavcích aplikace. Když platnost vyprší a token se stane neplatným, ale uživatel stále potřebuje přístup k prostředkům API, je třeba poskytnout nový token, aniž by byl uživatel nucen k novému ověření přístupu. Pro tyto účely slouží Refresh Token, který zajišťuje vytvoření nového Access Tokenu po vypršení jeho platnosti. Aplikace může požádat o přidělení Refresh Tokenu během procesu přidělování Access Tokenu. Pro přidělení nového Access Tokenu je vytvořen požadavek POST na autorizační server s poskytnutím Refresh Tokenu. Server následně může přidělit uživateli nový Access Token. Refresh Tokeny mají dlouhou dobu platnosti, v porovnání s Access Tokeny, ovšem autorizační server je může zneplatnit například při změně autorizačních pravidel. Z důvodu dlouhé doby platnosti je nutné chránit Refresh Tokeny před zveřejněním a jejich uložiště musí být zabezpečeno. [49,50]

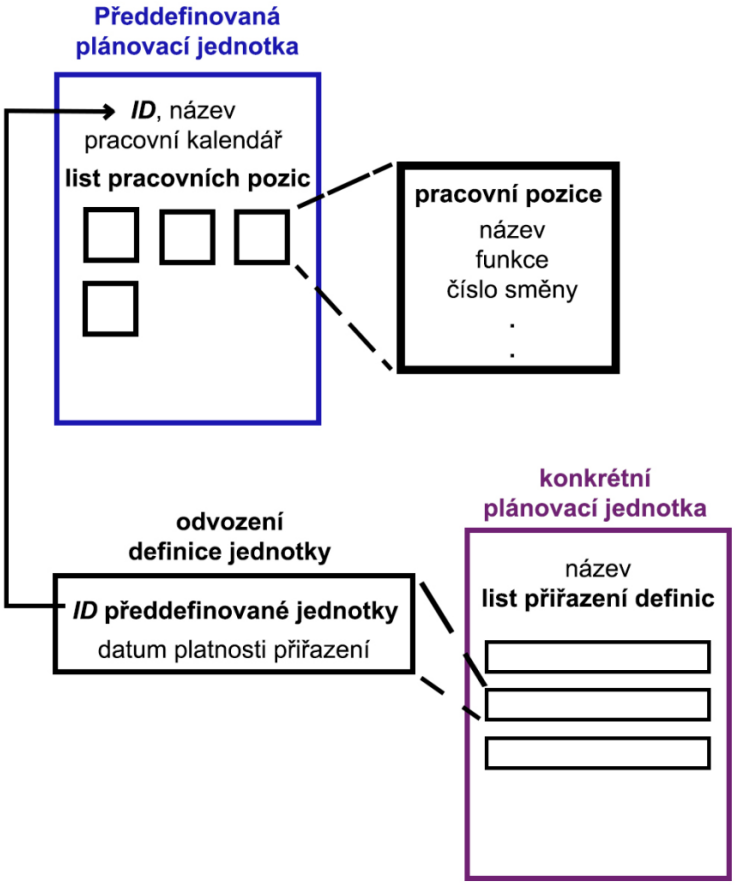
II. PRAKTICKÁ ČÁST

3 DATOVÉ STRUKTURY A APLIKAČNÍ OBJEKTY

Základním prvkem aplikace jsou datové struktury, od kterých se odvíjí následná tvorba databázových tabulek a aplikačních objektů, respektive tříd. Tato kapitola se zabývá jejich návrhem a vazbami pro další funkcionalitu aplikace. Jedná se o spodní vrstvu aplikační logiky plánovacího modulu, která zajišťuje funkcionalitu, komunikaci s databází a zpracování dat.

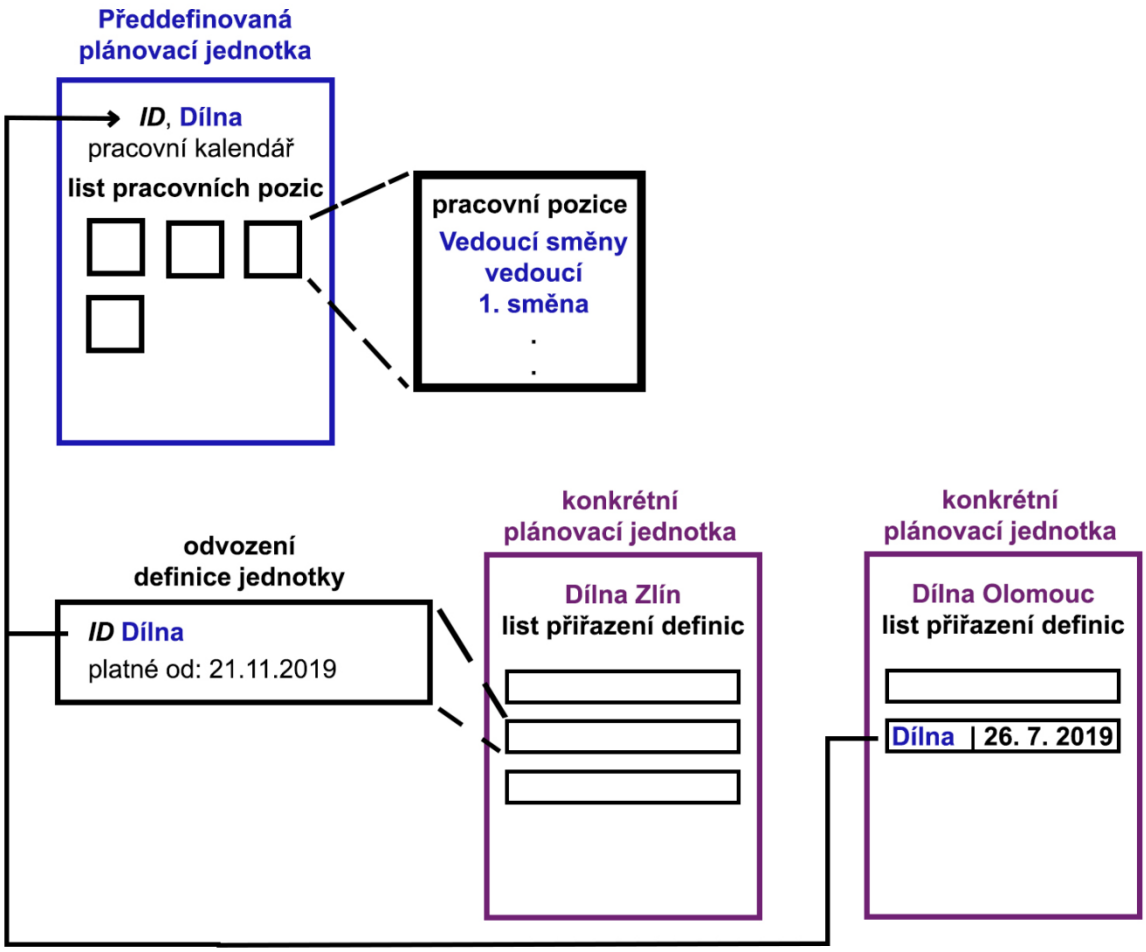
3.1 Plánovací jednotky

Během celé práce se bude používán pojem plánovací jednotka, proto je nutné popsat jeho význam. Plánovací jednotka je struktura, která definuje a sdružuje skupinu pracovních pozic se společným pracovním kalendářem a směnami. Pro její definici se využívá tabulek *kplajh* a *kplajr*, popsaných v následující podkapitole 3.2. Takto předdefinované struktury se dále využívají k definicím již konkrétních plánovacích jednotek, které jsou od nich odvozeny, a přiřazení je platné od určitého dne, jak je možné vidět na obrázku 4. Konkrétní plánovací jednotky využívají tabulek *kpjedh* a *kpjedr* (podkapitola 3.2).



Obrázek 4 – Schéma vazby předdefinované a konkrétní plánovací jednotky

Jako příklad uveďme vytvoření předdefinované plánovací jednotky Dílna, která udává směnovost a předpis pracovních pozic. Na obrázku 5 je znázorněna modrou barvou. Nyní chtějme vytvořit dvě konkrétní dílny Zlín a Olomouc, které jsou označeny fialovou barvou. Každá z nich převezme definici Dílny a jejich pracovních pozic a od zvoleného data se stává platnou. Této konkrétní dílně poté lze vytvářet plán pracovních pozic.

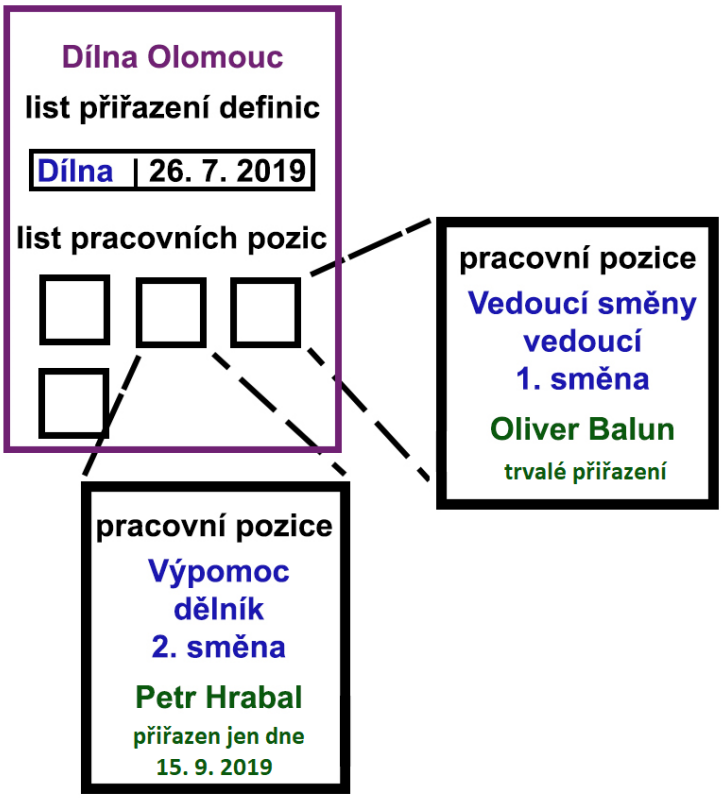


Obrázek 5 – Příklad odvození konkrétních plánovacích jednotek od předdefinované

Zařazení zaměstnanců do jednotek

Každá pracovní jednotka nese seznam pracovních pozic, které je třeba obsadit zaměstnanci. Přiřazením dochází k samotnému plánování a zapsání vazby do tabulky. Přiřazení může být dvojího druhu: defaultní a anomálie, proto jsou použity dvě tabulky *kpljpp* a *kanopj*, které jsou taktéž popsány v podkapitole 3.2. Defaultní přiřazení znamená, že je zaměstnanec přiřazen k pozici na stálo a toto přiřazení je bráno v potaz při plánování obsazení pozice pro další dny. K těmto účelům je použita tabulka *kpljpp* (podkapitola 3.2). Anomálie slouží k vytváření výjimek a přiřazením zaměstnance k pozici na konkrétní den a směnu, například jako

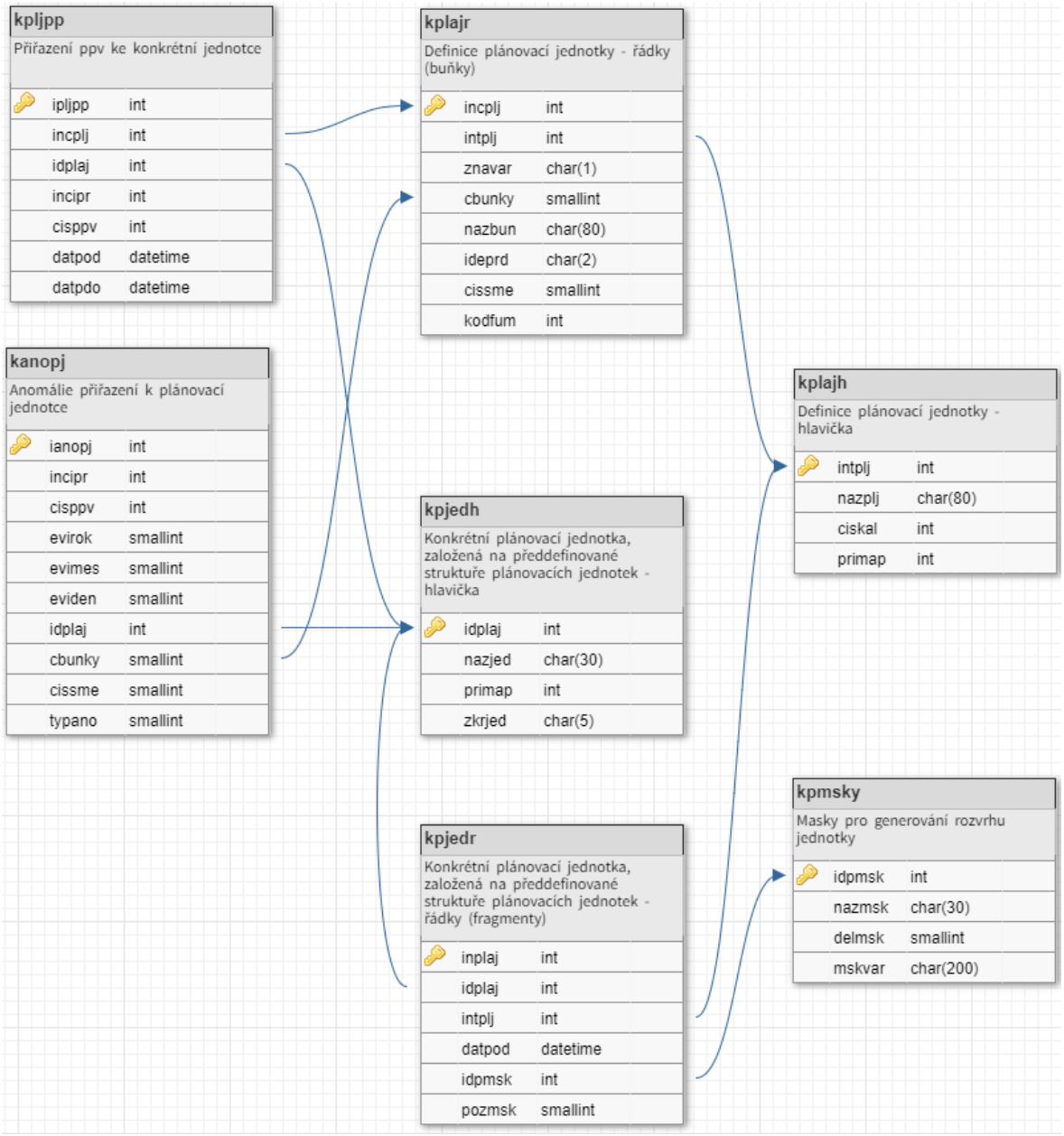
náhrada za defaultně přiřazeného. Anomálie jsou zapisovány do tabulky *kanopj* (podkapitola 3.2). Na obrázku 6 jsou znázorněny obě varianty přiřazení, také lze vidět, že jednotka nese definici pracovních pozic na základě předdefinované jednotky.



Obrázek 6 – Příklad přiřazení zaměstnance k pracovní pozici plánovací jednotky

3.2 Tabulky

Struktura je navržena jako sedm vzájemně provázaných tabulek. Sloupce tabulek korespondují s vlastnostmi objektů, které zajišťují práci s tabulkami a databází, a jsou popsány v následující podkapitole Třídy. Tabulky mají následující vazby a strukturu znázorněnou na obrázku 7.



Obrázek 7 – Návrh struktury databázových tabulek pomocí nástroje DbDesigner [51]

Názvy tabulek a sloupců jsou zkratkami slovních spojení pro vyjádření konkrétních vlastností. Zde je stručný popis jednotlivých tabulek:

- **kplajh** (kapacitní plánovací jednotka – hlavička) je tabulka, která slouží jako hlavička předdefinované plánovací jednotky. Obsahuje primární autoinkrementální klíč *intplj*, název jednotky *nazplj*, cizí klíč *ciskal* jako vazbu na konkrétní typ docházkového kalendáře a příznakovou mapu *primap*, která souží pro zápis případných detailů typu boolean.
- **kplajr** (kapacitní plánovací jednotka – řádky) je tabulka, která obsahuje jednotlivé pozice předdefinovaných plánovacích jednotek, které je třeba obsadit. Záznamy jsou

tvořeny primárním autoinkrementálním klíčem *incplj*, cizím klíčem *intplj* jako vazba na konkrétní předdefinovanou plánovací jednotku, jednopísmenným označením varianty *znavar*, číslem pozice (buňky) *cbunky*, která se bude v aplikaci obsazována, názvem pozice *nazbun*, identifikačním symbolem pracovní doby *ideprd*, číslem směny *cissme* a označením typu pracovní funkce *kodfum*, na základě které bude možné vybírat mezi konkrétními zaměstnanci vhodnými pro danou pracovní pozici.

- **kpjedh** (konkrétní plánovací jednotka – hlavička) je tabulka, která slouží jako hlavička pro vytvoření a přiřazení konkrétní plánovací jednotky na základě předešlých definovaných struktur plánovacích jednotek v tabulkách kplajh a kplajr. Obsahuje primární autoinkrementální klíč *idplaj*, název jednotky *nazjed*, zkratku názvu *zkrjed* a příznakovou mapu *primap*.
- **kpjedr** (konkrétní plánovací jednotka – hlavička) je tabulka, která obsahuje údaje o konkrétním přiřazení předdefinované plánovací jednotky, na které je založena, a od kdy je toto přiřazení platné. Znamená to, že jednotka bude mít přiřazenu právě takovou strukturu a na jejím základě bude plánováno rozložení pracovních míst. Skládá se z primárního autoinkrementálního klíče *inplaj*, cizího klíče *idplaj*, který slouží pro vazbu na konkrétní jednotku, z cizího klíče *intplj*, který je vazbou na předdefinovanou plánovací jednotku a data počátku platnosti *datpod*. Dále obsahují sloupce *idpmsk* a *pozmsk*, které nejsou doposud využity, ale v budoucnu by měly sloužit pro tvorbu masek, na základě kterých by mohl být generován rozvrh jednotky.
- **kpljpp** (kapacitní plánovací jednotka přiřazení pracovníka) je tabulka, která obsahuje přiřazení zaměstnanců na konkrétní pozici plánovací jednotky. Sestává z primárního autoinkrementálního klíče *ipljpp*, interního čísla pracovníka *incipr*, čísla pracovněprávního vztahu *cisppv*, data počátku a konce platnosti přiřazení *datpod a datpdo*, cizího klíče plánovací jednotky *idplaj* a čisla pozice *incplj*.
- **kanopj** (kapacitní anomálie plánovací jednotky) je tabulka, která slouží pro evidenci anomálií při plánování, které mohou vznikat jako výjimky přiřazení na pracovní pozice pro konkrétní datum a směnu. Obsahuje primární autoinkrementální klíč *ianopj*, číslo pracovníka *incipr* a číslo pracovněprávního vztahu *cisppv*, datum platnosti rozdělené na části rok, měsíc, den, respektive *evirok*, *evimes*, *eviden*. Dále nese vazbu na plánovací jednotku v podobě cizího klíče *idplaj*, číslo pracovní pozice v jednotce *cbunky*, číslo směny *cissme* a typ anomálie *typano*, který udává, zda je anomálie aktivní či zrušená.

- **kpmsky** (kapacitní plánování – masky) je přichystaná tabulka pro tvorbu masek, na základě kterých by mohl být generován rozvrh pracovní jednotky. Tato tabulka je momentálně nevyužitá, ale byla navrhnuta do struktury za účelem možnosti budoucího rozšíření funkcionality. Obsahuje autoinkrementální klíč *idpmsk*, název *nazmsk*, délku masky *delmsk*, po které by se mělo generování opakovat a symbol varianty masky, který může být zobrazen pro daný den rozvrhu.

3.3 Třídy

Třídy projektu slouží převážně pro práci s daty plánovacích jednotek poskytnuté databází, obsahují vlastnosti, funkce a procedury pro jejich obsluhu. Vzhledem k tomu, že vlastnosti třídy odráží strukturu databázových tabulek, je použito dvojic tříd, ve které jedna třída definuje vlastnosti jedné položky (row) a druhá seznam (list) položek. Třídy využívají i jiných tříd a jejich procedury s nimi dále pracují. Uveďme nyní stručný popis nejvýznamnějších vlastností a funkcí tříd, které se nachází v programovém kódu v příloze P I, souborech [1] a [2] respektive *P3ObjPlanJed* a *P3ObjPlanKap*.

Při tvorbě tříd je třeba dbát zaběhlých konvencí a způsobů, jakým se ve firmě Kvasar vytváří. Je důležité dodržet strukturu, vlastnosti a podobu obslužných funkcí tak, aby bylo možné třídy začlenit do stávající aplikace. Z tohoto důvodu je využíváno několik předdefinovaných prvků či vazeb.

TKvaDefBunka popisuje základní prvek, kterým je pracovní pozice v plánovací jednotce. Jednotlivé veřejné vlastnosti třídy (property) odpovídají sloupcům v tabulce *kplajr*. K práci s vlastnostmi jsou využívány gettery a settery, což jsou funkce, které slouží k získání, respektive nastavení hodnot. Tyto funkce přijímají vstupní parametr Index typu Integer, který určí, o kterou položku se jedná, a setter navíc hodnotu Value, na kterou má být hodnota vlastnosti nastavena. V případě getteru je návratová hodnota shodná s typem vlastnosti, setter návratovou hodnotu nemá. Definici třídy je možné vidět na obrázku 8.

Funkce `Status` slouží pro zjištění stavu řádku při jeho zpracování. Nabývá hodnot `rsNone`, `rsInsert`, `rsUpdate`, `rsDelete` a je využíván při obsloužení zápisu do databáze (Obrázek 15).

```
type TKvaDefBunka=class(TKvaColumnList)
     private
       function GetInteger(Index:Integer):Integer;
       procedure SetInteger(Index:Integer;Value:Integer);
       procedure SetSmallint(Index:Integer;Value:Smallint);
       function GetSmallint(Index:Integer):Smallint;
       procedure SetString(Index:Integer;Value:String);
       function GetString(Index:Integer):String;
     public
       constructor Create(AOwner:TComponent);override;
       property incplj:Integer index 0 read GetInteger;
       property intplj:Integer index 1 read GetInteger write SetInteger;
       property znavar:String index 2 read GetString write SetString;
       property cbunky:SmallInt index 3 read GetSmallint write SetSmallint;
       property nazbun:String index 4 read GetString write SetString;
       property ideprd:String index 5 read GetString write SetString;
       property cissme:SmallInt index 6 read GetSmallint write SetSmallint;
       property kodfum:Integer index 7 read GetInteger write SetInteger;
       function Status:TKvaRowStatus;override;
     end;
```

Obrázek 8 – Třída definující jednu buňku plánovací jednotky

Důležité vlastnosti třídy z hlediska plánování jsou vlastnosti:

- `cissme` – proměnná typu smallint, která určuje číslo směny pro danou buňku
- `cbunky` – jedná se o proměnnou typu smallint, která určuje pořadí buňky (pracovní pozice) v rámci seznamu pozic pro danou směnu. Ve spojitosti s číslem směny tedy jednoznačně určuje, o kterou pracovní pozici se jedná, a této pozici jsou poté přiřazovány pracovníka.
- `kodfum` – proměnná typu integer, která určuje číslo funkce. Toto číslo odpovídá číselníkové struktuře kódu funkcí v programu Perm3, a na jeho základě lze určit typ, jakého pracovníka je na pozici potřeba. Pokud nese buňka kód funkce pro konstruktéra, bude k buňce požadováno přiřazení takových zaměstnanců, kteří mají kód funkce shodný s kódem pro konstruktéra.
- `znavar` – jednopísmenná proměnná, která slouží k určení varianty pracovní jednotky. Nyní v základní verzi je použita jedna varianta A, která označuje stále stejný provoz. Pokud bychom chtěli docílit více variant například pro všední den a víkend, je možné zadat buňkám, které se v dané variantě mají v jednotce vyskytovat, shodné písmeno varianty a na základě něho se poté mohou uskupovat v dalším zpracování plánovací jednotky.

TKvaDefBunkaList je seznamem buněk plánovací jednotky a odpovídá záznamům v tabulce *kplajr*. Obsahuje položky typu *TKvaDefBunka* a funkce pro obsluhu seznamu:

- `RadekOfBunka` slouží pro nalezení indexu v seznamu buněk dle konkrétní buňky
- `DefBunkaDleKodu` slouží pro nalezení buňky dle primárního klíče *incplj*
- `DefBunkaDleSmenyAcisla` slouží pro nalezení konkrétní buňky dle čísla směny a cisla buňky
- `InsertRow` složí pro vložení řádku do seznamu.

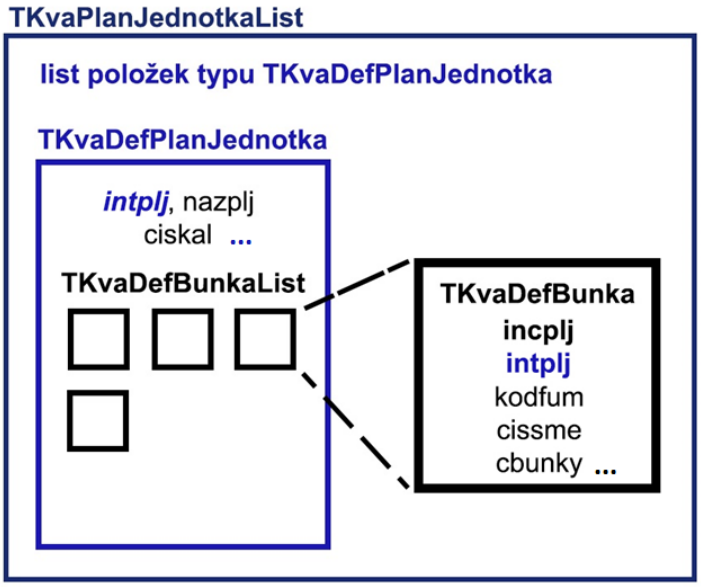
TKvaDefPlanJednotka popisuje hlavičku definice plánovací jednotky a její vlastnosti odpovídají sloupcům v tabulce *kplajh*. Kromě těchto vlastností nese také seznam buněk `DefBunkaList`, které jsou k jednotce přiřazeny. Tento seznam nemá getter a setter, ale je obsluhován privátní proměnnou `FDefBunkaList`. Dále se zde nachází funkce `Save` a `Restore` pro uložení a obnovu seznamu, a opět `Status`. Vlastnost `ciskal` je číselníková hodnota definovaná v programu Perm3, která odkazuje na typ kalendáře, podle kterého má jednotka směnovost, a i vlastnosti plynoucí z kalendáře.

TKvaDefPlanJednotkaList je seznamem definic plánovacích jednotek a odpovídá záznamům v tabulce *kplajh*. Obsahuje položky typu *TKvaDefPlanJednotka* a funkce pro obsluhu seznamu:

- `RadekOfJednotka` slouží pro nalezení indexu položky v seznamu jednotek dle konkrétní jednotky
- `DefPlanJednotkaDleKodu` slouží pro nalezení buňky dle primárního klíče *intplj*
- `InsertRow` složí pro vložení řádku do seznamu.

Dále se ve třídě nachází privátní proměnná `FSQL` typu *TKvaDefPlanJednotkaListSQL*. Tato třída je použita při definici třídy seznamu a obsahuje pouze dvě procedury `Read` a `Write`. Tyto procedury zajišťují práci s databází, tedy načtení seznamu jednotek, případně jeho zápisu. Tyto „SQL“ třídy a jejich procedury se jsou blíže popsány v další podkapitole.

Struktura tříd pro definici obecných plánovacích jednotek je znázorněna na obrázku 9. Je tvořena od základních pracovních pozic, až po kompletní pracovní jednotku, ke které je přiřazen seznam těchto pozic. Obdobným způsobem jsou definovány i následující třídy, které taktéž odpovídají strukturám patřičných tabulek v databázi.



Obrázek 9 – Schéma struktury tříd pro definici plánovacích jednotek

TKvaPJFragment popisuje základní prvek pro vytvoření konkrétní plánovací jednotky. Jednotlivé veřejné vlastnosti třídy odpovídají sloupcům v tabulce *kpjedr*. Jedná se o záznamy, které definují jednotku s platností od určitého data a odkazují na předdefinované struktury. Důležité sloupce pro vznik vazby jsou:

- `idplaj` – slouží jako cizí klíč pro vazbu na hlavičku konkrétní plánovací jednotky
- `intplj` – slouží jako cizí klíč pro vazbu na předdefinovanou plánovací jednotku, podle které má být vytvořena struktura
- `datpod` – proměnná typu TDateTime, která určuje platnost definice od určitého data

Na základě těchto vlastností jsme schopni vytvořit různé definice s různými platnostmi k hlavičce plánovací jednotky. Výhoda tohoto rozdělení a číselníkového přiřazování je v tom, že je možné reagovat například na rozšíření pracovní jednotky a přiřazení jiné předdefinované struktury plánovací jednotce.

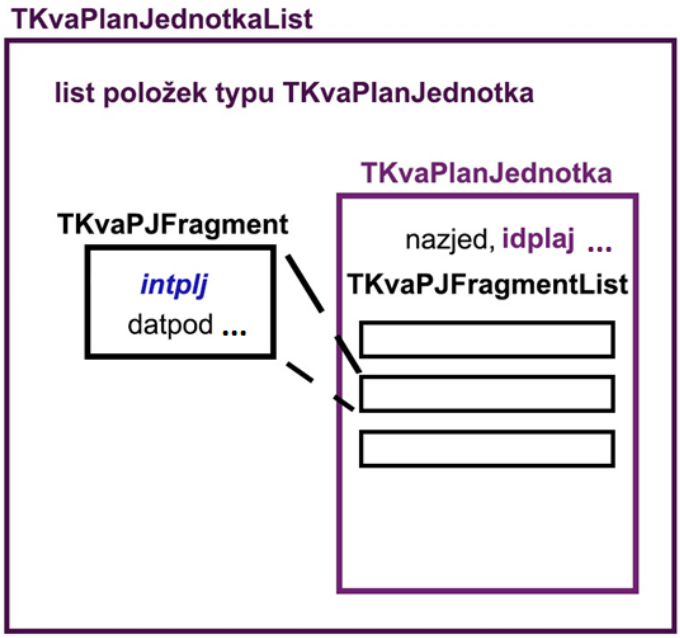
TKvaPJFragmentList je seznamem fragmentů plánovacích jednotek a odpovídá záznamům v tabulce *kpjedr*. Obsahuje položky typu *TKvaPJFragment*, a funkce pro obsluhu seznamu, shodně jako v předchozích případech, které slouží pro nalezení indexu prvku v seznamu, vložení řádku a získání konkrétního záznamu.

TKvaPlanJednotka popisuje hlavičku konkrétní plánovací jednotky a vlastnosti odpovídají sloupcům v tabulce *kpjedh*. Kromě těchto vlastností nese také seznam přiřazení definice ke konkrétní jednotce `Fragmenty`. Ten je obsluhován privátní proměnnou `FFragmentList`.

Užitečnými vlastnostmi z hlediska plánování mohou být do budoucna `Rok`, `Mesic`, `Den`, na základě kterých se pomocí kalendáře přiřadí aktuální hodnoty pro dané dny. Tyto vlastnosti mohou být využity ve funkci `PoziceVeSmene`, které mají vstupní parametry `cbunky` a `cissme` a slouží pro zisk konkrétní buňky plánovací jednotky. Důležitou funkcí je `FragmentDleData`, která má vstupní parametr typu TDateTime a na základě vlastnosti fragmentu `datpod` vrací odpovídající fragment ze seznamu `Fragmenty`, aktuálně platný k zadanému datu.

TKvaPlanJednotkaList je seznamem hlaviček konkrétních plánovacích jednotek a odpovídá záznamům v tabulce *kpjedh*. Obsahuje položky typu *TKvaPlanJednotka* a dále objekt státního kalendáře, pomocí kterého je možné pracovat s konkrétními dny. Struktura vazeb tříd pro konkrétní plánovací jednotky je nastíněna na obrázku 10. Dále se ve třídě nachází vazba na předpis definic plánovacích jednotek jako vlastnost `DefPlanJednotkaList`, aby bylo možné vytvořit jednotlivé struktury na základě vazby v záznamech konkrétních jednotek. Dále se zde nachází typické funkce pro nalezení indexu položky v seznamu `PlanJednotkaRadekOf` a pro nalezení jednotky na základě primárního klíče *idplaj* `PlanJednotkaDleKodu`.

Ve třídě se opět vyskytuje privátní proměnná `FSQL` pro práci s databází, tentokrát typu *TKvaPlanJednotkaListSQL* a podobně jako v předchozím případě obsahuje dvě procedury `Read` a `Write`.



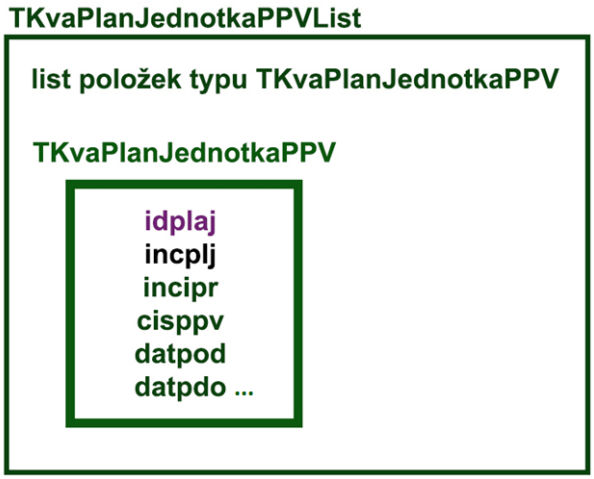
Obrázek 10 – Schéma struktury tříd pro tvorbu konkrétních plánovacích jednotek odvozených od předdefinovaných

TKvaPlanJednotkaPPV popisuje přiřazení zaměstnance na pozici v pracovní jednotce. Jedná se o jednoduchou třídu, kdy jednotlivé veřejné vlastnosti odpovídají sloupcům v tabulce *kpljpp*.

TKvaPlanJednotkaPPVList obsahuje položky typu *TKvaPlanJednotkaPPV*, a funkce pro obsluhu seznamu, shodně jako v předchozích případech. Odpovídá záznamům v tabulce *kpjedr*. Důležitou roli pro vkládání nových záznamů plní funkce `BudouciPlanJednotkaPPV`, která má za úkol nalézt nejbližší budoucí přiřazení pracovníka na danou pozici dle parametrů. Parametry, které funkce přijímá, jsou datum počátku hledání, id plánovací jednotky, id buňky, číslo pracovníka a číslo pracovněprávního vztahu. Dochází k procházení seznamu záznamů a v případě shody s parametry a nižšího data počátku platnosti je tento záznam brán jako výsledek funkce.

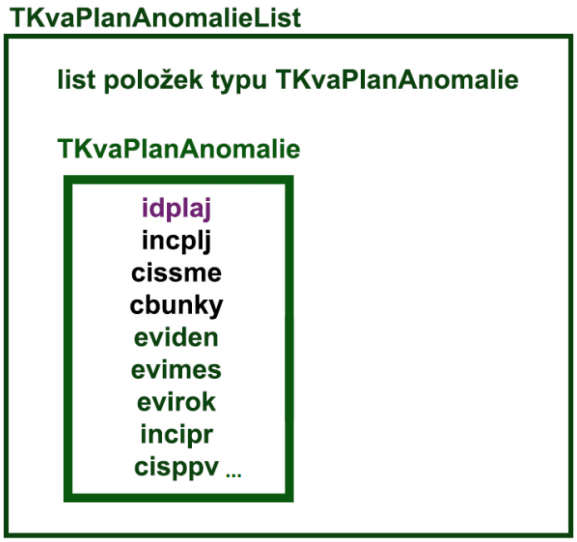
Užitečnou funkcí je také `AktualPlanJednotkaPPV`, která přijímá čtyři parametry: datum, číslo konkrétní plánovací jednotky, číslo definované pracovní pozice a id předchozího přiřazení, kdy na základě těchto parametrů vrátí aktuálně platné přiřazení pracovníka na danou pozici v plánovací jednotce. Vzhledem k tomu, že může být na danou pozici přiřazeno více zaměstnanců, je pro určení všech použit parametr přechozího id, aby bylo možné při nalezení přiřazení pokračovat na základě jeho id k dalšímu záznamu. Funkci je třeba volat v cyklu

nad celým seznamem s použitím pomocné proměnné pro udržování informace o předchozím id. I zde se vyskytuje proměnná `FSQL` pro práci s databází typu *TKvaPlanJednotkaPPVList-SQL* obsahující dvě procedury `Read` a `Write`. Struktura tříd pro trvalé přiřazení pracovníka na pracovní pozici je znázorněna na následujícím obrázku 11.



Obrázek 11 – Schéma třídy pro trvalé přiřazení pracovníků k pracovnímu místu plánovací jednotky

TKvaPlanAnomalie slouží pro tvorbu výjimek přiřazení na pozici v plánovací jednotce. Vznikají ručními úpravami plánu a dosazením do pozice v případě neobsazení pozice či náhrady. Veřejné vlastnosti odpovídají sloupcům v tabulce *kanopj*.



Obrázek 12 – Schéma třídy pro jednorázové přiřazení pracovníků k pracovnímu místu plánovací jednotky

TKvaPlanAnomalieList je seznam anomálií přiřazení a obsahuje položky typu *TKvaPlanAnomalie*, jak je znázorněno na obrázku 12. Kromě funkcí pro obsluhu seznamu se zde vyskytuje funkce pro nalezení konkrétní anomálie dle zadaných parametrů `AnomalieDleParam`, pokud není při vyhledávání k dispozici primární klíč, v opačném případě je standardně využíváno funkce `AnomalieDleKodu`. Pro práci s databází je využito proměnné `FSQL` typu *TKvaPlanAnomalieListSQL* obsahující procedury `Read` a `Write`.

3.4 Práce tříd s databází

Třídy pro práci s databází obsahují pouze dvě procedury `Read` a `Write`, které nepřijímají žádné parametry a slouží k načtení seznamů případně jejich zápisu do patřičných tabulek. Příklad definice v programovém kódu je uveden na obrázku 13. Pro komunikaci s databází je použito několik tříd a procedur vytvořených a standardně používaných společností Kvasar jako jsou TKvaSelecter, TKvaInserter, TKvaUpdater, TKvaDeleter či `Transfer`, `OpenCursor` a `CloseCursor`.

Konkrétní třídy, které zajišťují čtení a zápis jsou definovány společně s třídou, které manipulaci s databází poskytují, v souladu s postupy společnosti Kvasar. Nesou shodné jméno s danou třídou, ke kterému je připojen řetězec „SQL“, například *TKvaPlanJednotkaList* a *TKvaPlanJednotkaListSQL*. Pomocí nich je na seznamu definována privátní proměnná `FSQL`, která obsluhuje veřejnou vlastnost `SQL`, skrz kterou je možné následně při použití třídy volat příslušné funkce pro čtení či zápis. Vytvořené třídy jsou:

- TKvaDefPlanJednotkaListSQL – slouží pro práci s tabulkami *kplajh* a *kplajr*
- TKvaPlanJednotkaListSQL – slouží pro práci s tabulkami *kpjedh* a *kpjedr*
- TKvaPlanAnomalieListSQL – slouží pro práci s tabulkou *kanopj*
- TKvaPlanJednotkaPPVListSQL – slouží pro práci s tabulkou *kpljpp*

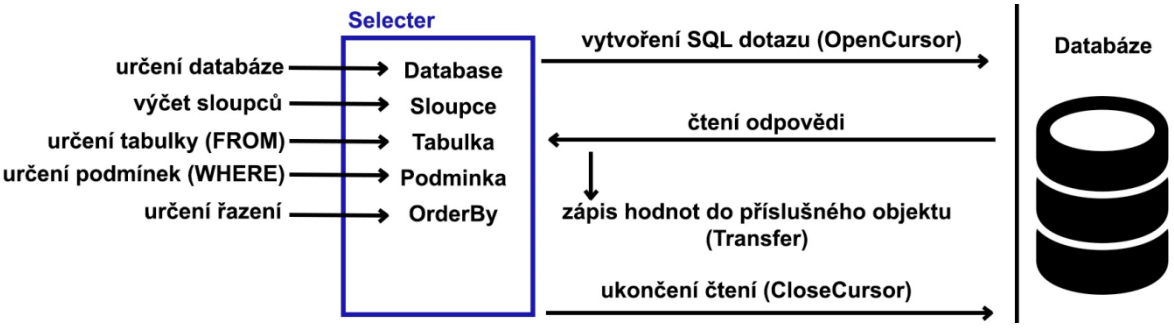
```
TKvaPlanJednotkaPPVListSQL=class(TKvaSQL)
public
  procedure Read;override;
  procedure Write;override;
end;
```

Obrázek 13 – Příklad programového kódu definice třídy pro práci s databází

`Read` slouží pro čtení záznamů z databáze za použití objektu `Selecter`, třídy TKvaSelecter, který zpracovává SQL dotazy na základě zadaných hodnot. Na obrázku 14 je

znázorněn průběh procedury. Objektu je určena databáze, název tabulky, názvy sloupců, podmínky či způsob řazení hodnot. Následně dochází k volání procedury `OpenCursor`, která otevírá databázový kurzor. Dále dochází k procházení všech nalezených záznamů v tabulce, kdy jsou záznamy postupně vkládány do příslušných řádků objektu a procedurou Transfer převáděny položky záznamu do příslušných vlastností objektu dle názvu. Poté dochází k výběru dalšího záznamu procedurou `Fetch`. Jakmile dojde k vyčerpání řádků, je čtení ukončeno a kurzor zavřen procedurou `CloseCursor`.

Jelikož se u tříd *TKvaDefPlanJednotkaListSQL* a *TKvaPlanJednotkaListSQL* pracuje s více než jednou tabulkou, je zde `Read` rozšířen o další úroveň. Tabulky jsou formulovány jako hlavička a řádky, to znamená, že ke každé hlavičce je nutno číst příslušné řádky. Čtení hlavičky je zajištěno standardně objektem `Selecter`, a čtení přidružené tabulky je prováděno dalším objektem `SubSel` také typu TKvaSelecter, který v rámci cyklu čtení hlavičky provede další cyklus čtení řádků přidružených k hlavičce. Následně je pomocí procedury `Transfer` zapíše do patřičných vlastností objektů a tím dojde k jeho kompletnímu načtení. Jedná se zde tedy o čtení seznamu seznamů neboli vazbu N:N.



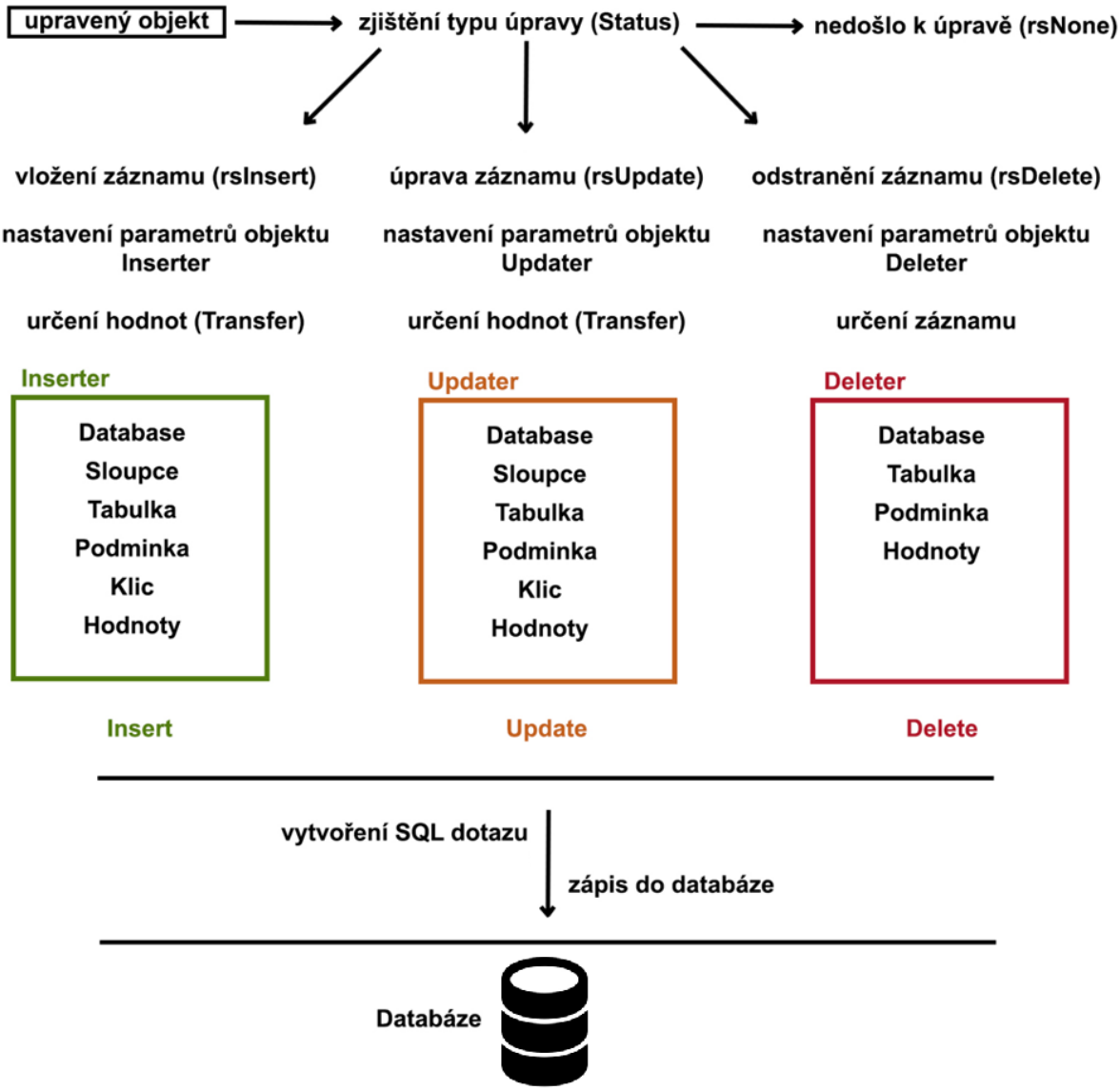
Obrázek 14 – Schéma načítání dat z databáze použitím třídní procedury Read [52]

`Write` zajišťuje zápis hodnot objektů do databáze. Využívá k tomu objekty `Inserter, Updater, Deleter` typu TKvaInserter, TKvaUpdater, TKvaDeleter, které slouží pro vykonávání příslušného typu zápisu. Na obrázku 15 je znázorněn průběh procedury zápisu nad úpravami v datech objektu. Dochází zde k cyklickému procházení objektu seznamu položek, při kterém je zjišťován status, již dříve zmiňovanou funkcí `Status`, na základě kterého dochází k provádění dalších akcí.

- Status **rsInsert** zapříčiňuje vložení nového záznamu. Dochází k vytvoření objektu `Inserter` a podobně jako při čtení je objektu přiřazena databáze, tabulka, výčet sloupců a dále primární klíč. Na základě těchto údajů a hodnot v řádku zpracovávaného objektu jsou hodnoty namapovány procedurou Transfer na sloupce tabulky.

Výjimkou jsou autoinkrementální primární klíče, které jsou určeny samy zvlášť databází. Následně je vytvářen nový záznam v tabulce procedurou `Inserter.Insert`, která po bezchybném průběhu spojení s databází zapíše nový prvek do tabulky.

- Status **rsUpdate** slouží pro úpravu záznamu v případě, že došlo ke změně daného objektu. Využívá objektu `Updater` typu TKvaUpdater, kterému obdobně jako u `Inserteru` je přisouzena databáze, tabulka a podmínka, na základě kterých je určen řádek, který je třeba aktualizovat. Následně je volána procedura `Updater.Update`, která provede samotný požadavek na zápis hodnot.
- Status **rsDelete** udává požadavek na smazání záznamu. Slouží k němu objekt `Deleter` typu TKvaDeleter, kterému je opět přisouzena databázi, tabulku a podmínku pro rovnost klíče dle kterého se určí, který záznam bude smazán. Poté dochází k volání procedury `Deleter.Delete`, která zajistí smazání záznamu.
- V případě statusu **rsNone** nedochází k žádné akci, neboť nedošlo k žádné změně od původně načtených dat.



Obrázek 15 – Schéma zápisu dat do databáze za použití třídní procedury Write [52]

Obdobně jako u procedury `Read` se u tříd *TKvaDefPlanJednotkaListSQL* a *TKvaPlanJednotkaListSQL* pracuje s více než jednou tabulkou, a proto je třeba rozšířit `Write` o další úrovně. Tabulky jsou formulovány jako hlavička a řádky, to znamená, že kromě změn hlaviček je potřeba zapisovat i změny příslušných řádků. Zápis změn hlaviček je zajištěn standardně objekty `Inserter, Updater a Deleter`. Zápis řádků do přidružené tabulky je prováděn v dalších subcyklech objekty `SubIns, SubUpd, SubDel`, které v rámci cyklu zápisu hlavičky provedou zápis úprav řádků přidružených k hlavičce. Následně jsou vlastnosti opět namapovány pomocí procedury `Transfer` do patřičných sloupců tabulek. Jedná se o zápis seznamu seznamů opět ve vazbě N:N. Tímto způsobem je zajištěn kompletní zápis všech možných změn v rámci obou tabulek.

4 ZAPRACOVÁNÍ STRUKTUR DO APLIKACE PERM3

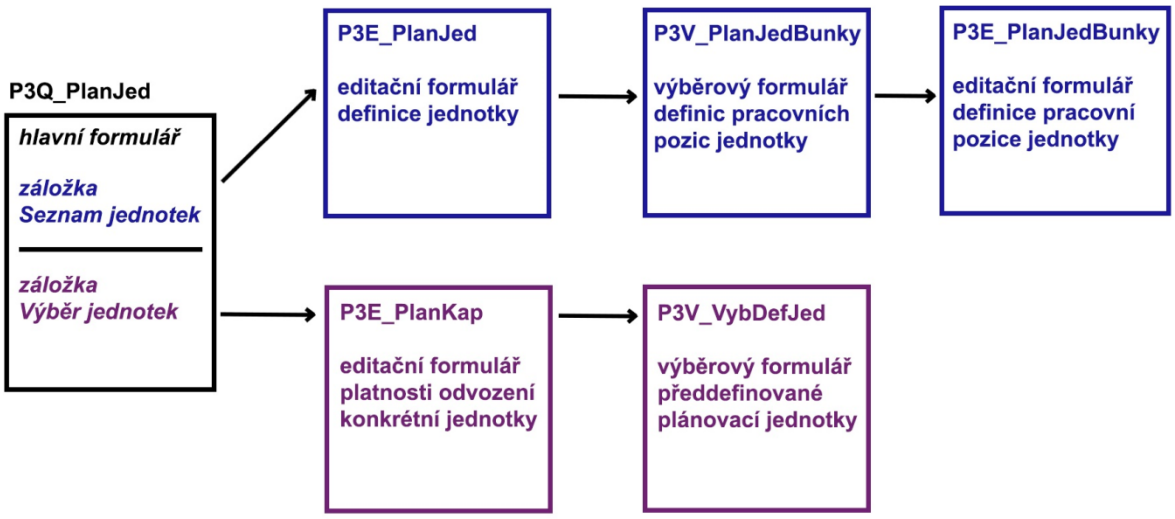
Dalším krokem tvorby tohoto modulu je zapracování navržených struktur do samotné aplikace. Jsou k tomu využity vytvořené třídy, které jsou přidány do projektu, a tabulky vloženy, respektive vytvořeny do databáze. Dále je třeba tyto prvky vhodně prezentovat a v rámci aplikace s nimi pracovat. K těmto účelům slouží programová okna, která zajišťují funkcionalitu, vazbu na obsah objektů a ostatní aplikační vlastnosti. V následující podkapitoly věnují se těmto vytvořeným formulářům.

4.1 Formuláře

Vzhled aplikace Perm3 a jeho prezentační vrstva je založena na formulářích (oknech), které dědí své vlastnosti od předdefinovaného předka *TForm*. Formuláře jsou ve firmě Kvasar tvořeny zaběhlým postupem, který je třeba dodržovat. Z tohoto důvodu jsou popsány, jaký mají jednotlivé formuláře účel, a k čemu slouží jejich funkce.

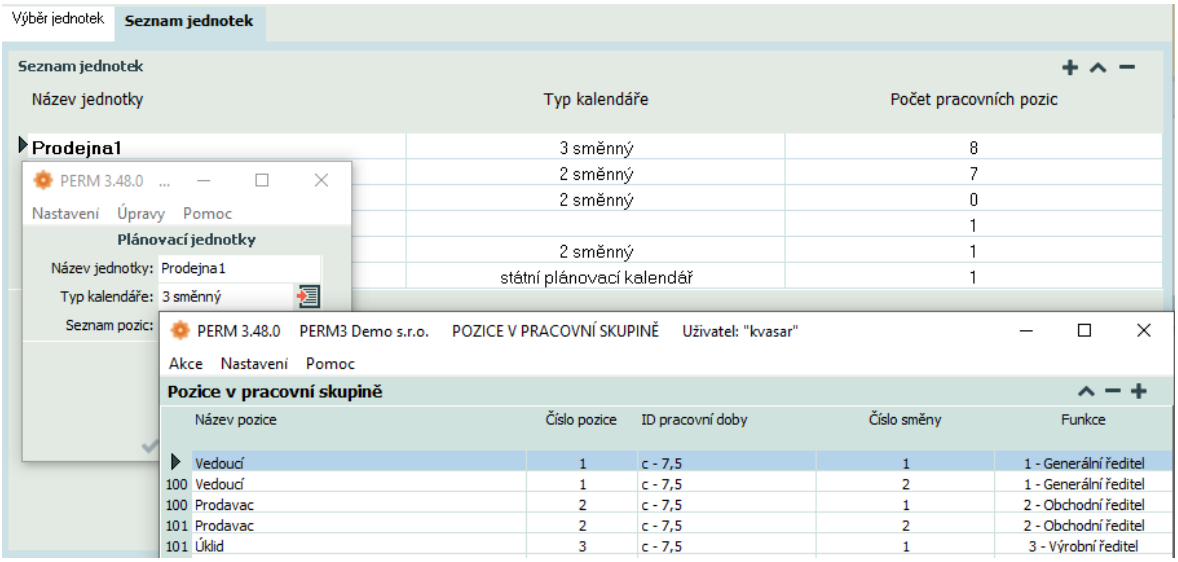
4.1.1 P3Q_PlanJed

Formulář P3Q_PlanJed je základem kapacitního plánování v aplikaci Perm3 a odvíjí se od něj veškerá další interakce s uživatelem. Zastává hlavní funkci zobrazení obsahu a práce se seznamy plánovacích jednotek, a to jak předdefinovaných, tak konkrétních. Z toho důvodu obsahuje dvě záložky: Výběr jednotek a Seznam jednotek. Jedná se o programový kód v příloze P I, souboru [3]. Na obrázku 16 je možné vidět propojení spouštění jednotlivých formulářů. Všechny vychází z hlavního formuláře P3Q_PlanJed.



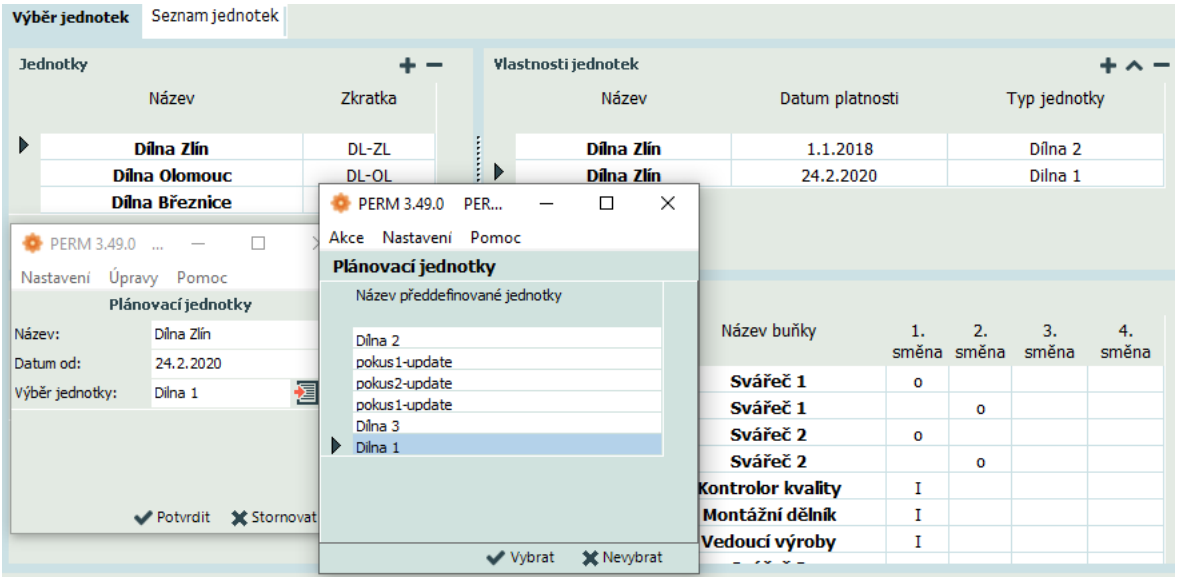
Obrázek 16 – Znázornění vazeb a vzájemného spouštění vytvořených formulářů

Záložka Seznam jednotek prezentuje seznam předdefinovaných plánovacích jednotek a poskytuje tlačítka pro jejich editaci + ^ –, kterými dochází k vyvolání příslušných formulářů určených k úpravám seznamu DefPlanJednotkaList: *P3E_PlanJed*, *P3V_PlanJedBunky*, *P3E_PlanJedBunky*., jak je možné vidět na obrázku 17.



Obrázek 17 – Formuláře pro tvorbu definice plánovací jednotky

Výběr jednotek je záložka, která slouží pro prezentaci konkrétních jednotek vytvořených na základě těch předdefinovaných ze záložky Seznam jednotek. Je rozdělena do čtyř oblastí, gridů. První je seznam hlaviček konkrétních plánovacích jednotek, který je možné upravovat přímo bez nutnosti vyvolávání podřízeného formuláře. Druhou částí jsou Vlastnosti jednotek, které po výběru některé hlavičky, zobrazují k ní přiřazené řádky, a tedy konkrétní definice plánovacích jednotek. Seznam je možné upravovat pomocí tlačítek pro editaci, a tím vyvolání příslušného formuláře pro editaci *P3E_PlanKap*. Třetí část Plán je mapa dnů v měsíci nesoucí typ varianty pracovní jednotky pro konkrétní dny. Tato část je zde nachystána pro možnosti budoucích rozšíření na základě masek pracovních jednotek. Bylo by možné mít například jinou variantu jednotky ve všední den a o víkendu, a tím i plánovat jiný počet zaměstnanců. Aktuálně však je použita pouze se základní varianta A, která na každý den definuje stejnou strukturu pracovních pozic jednotky. Čtvrtá část jsou Buňky, což je prezentační seznam pracovních pozic definovaných přiřazenou pracovní jednotkou. Jeho obsah závisí na výběru daného řádku (vlastnosti) jednotky a nelze upravovat, slouží pouze pro výčet definovaných pozic. Na obrázku 18 je možné vidět část podoby této záložky společně s navazujícími formuláři při úpravě konkrétní plánovací jednotky.



Obrázek 18 – Formuláře pro tvorbu konkrétní plánovací jednotky

Formulář ke svému chodu potřebuje být správně inicializován. To zajišťují procedury, které jsou postupně volány ještě před samotným zobrazením formuláře. Zaobaluje je procedura **InitForm**. Ta nejprve volá proceduru **InitFirstGo**, která zajišťuje prvotní nastavení formuláře a jeho jednotlivých částí (gridů), počtu řádků či sloupců, včetně zpracování počtu dnů v měsíci pro mapu dnů. Následuje **InitGrid**, který zajišťuje nastavení přepočtů velikostí řádků při změně velikosti fontu. Nakonec je volána přeplňovací procedura **RefillGrid**, pomocí které dochází k načtení zobrazovaných dat pro jednotlivé gridy (mřížky), určení počtu řádků, povolení či zamezení úprav a aktuálních objektů, se kterými se dále v rámci formuláře pracuje. Procedura `RefillGrid` je také volána při jakékoliv změně na formuláři, jako je úprava záznamů či výběr jiné jednotky. Provedené změny je pak možné ukládat pomocí tlačítka `BQsave`, které z předka dědí proceduru `ACsaveExecute`, pomocí které je nad příslušnými objekty volána procedura `Write` a dochází k zápisu dat do databáze. V opačném případě je možné provést zahození změn stiskem tlačítka „Storno“. Při zavírání formuláře je taktéž vyvolán dialog umožňující uložení provedených změn.

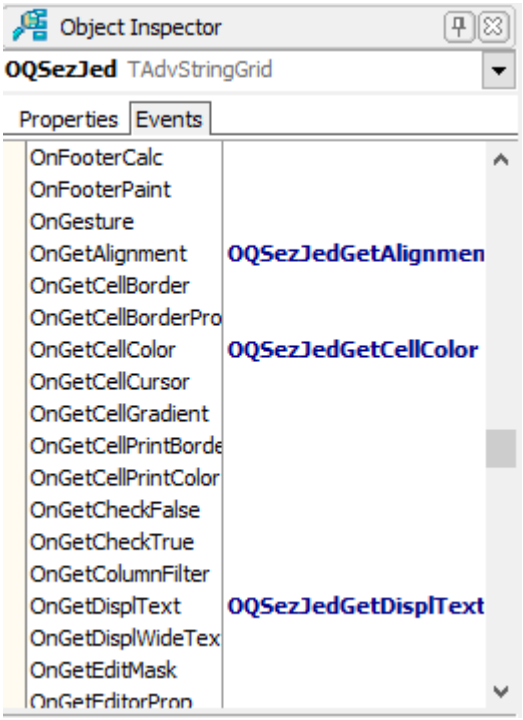
V rámci formuláře je nutné upravovat i seznamy, které nelze editovat přímo v gridu, ale pomocí editačních tlačítek. Jejich stiskem dochází k vyvolávání akcí, které vedou k inicializaci editace a zobrazení příslušných formulářů. Výjimkou je mazání záznamů, při kterém dochází pouze k zobrazení dialogu, zda chce uživatel daný záznam opravdu smazat.

Pomocí prvku *ActionList* jsou vytvořeny akce pro editaci seznamů. Tyto akce pomáhají při obsluze událostí kliku na editační tlačítka nebo dvojkliku na položku seznamu. Definované akce a jejich účel:

- **ACInsertExecute** – během této procedury dochází k vytvoření nového řádku v seznamu předdefinovaných plánovacích jednotek `DefPlanJednotkaList` a volání editačního formuláře *P3E_PlanJed* pro plnění jeho vlastností. Pokud je zadávání hodnot potvrzeno, jsou data překreslena a změny je možné uložit. V opačném případě dojde ke smazání řádku a data jsou zahozena.
- **ACEditExecute** – podobně jako při vkládání dochází k volání formuláře *P3E_PlanJed*. Vybraný záznam z objektu `DefPlanJednotkaList` je pomocí něj podroben editaci a při potvrzení jsou data překreslena.
- **ACDeleteExecute** – slouží pro jednoduché mazání vybraného záznamu z `DefPlanJednotkaList` pomocí dialogu.
- **ACJednotkyInsertExecute** – tato akce vloží nový řádek do objektu konkrétních plánovacích jednotek `PlanJednotkaList` a provede překreslení seznamu.
- **ACJednotkyDeleteExecute** – zajišťuje mazání vybraného záznamu z `PlanJednotkaList` pomocí dialogu.
- **ACVlastJedInsertExecute** – tato procedura zajišťuje vytvoření nového řádku v listu `Fragmenty`. Tento seznam je součástí konkrétní plánovací jednotky a obsahuje vlastnosti (vazby) plánovacích jednotek na předdefinovanou plánovací jednotku. Následně dochází k volání editačního formuláře *P3E_PlanKap* pro plnění vlastností nového záznamu. V případě, že je zadávání hodnot potvrzeno, jsou data překreslena a změny je možné uložit. V opačném případě dojde ke smazání řádku a data jsou odstraněna.
- **ACVlastJedEditExecute** – obdobně jako při vkládání dochází k volání formuláře *P3E_PlanKap*. Vybraný záznam z objektu `Fragmenty` je pomocí něj editován a při potvrzení jsou data překreslena.
- **ACVlastJedDeleteExecute** – poskytuje jednoduché mazání vybraného záznamu ze seznamu `Fragmenty` pomocí dialogu a následné překreslení.

Dále je vhodné zmínit obslužné akce definované pomocí *Object inspectoru* (Obrázek 19), které zajišťují nastavení zobrazení a úprav na jednotlivých gridech formuláře. Jedná se o děděné procedury obsluhy událostí, rozšířené o konkrétní požadavky na funkcionalitu gridu. Základními vstupními parametry jsou konkrétní řádek a sloupec gridu, se kterými procedura pracuje. Mezi nejdůležitější patří:

- **GetDisplText** – opakovaně volaná procedura pro vykreslení obsahu na základě načtených objektů `PlanJednotkaList` a `DefPlanJednotkaList`. Je doplněna o proměnnou `Value`, jejíž hodnota se prezentuje do gridu.
- **GetEditText** – nese shodný kód s procedurou `GetDisplText`, ale slouží pro poskytnutí hodnoty buňky při jejím upravování.
- **CellValidate** – navazuje na proceduru `GetEditText` v tom smyslu, že upravené hodnoty kontroluje, zda nejsou zadány chybně. Je doplněna o proměnné `Value` a `Valid`, kdy `Value` nese přiřazenou hodnotu a `Valid` určuje její platnost. Pokud jsou v pořádku, zapíše je do patřičného objektu. Tato procedura je na formuláři využita při editaci hlaviček jednotek v gridu `OQJednotky`.
- **RowChanging** – je volána v případě změny řádku a vstupními parametry jsou staré a nové číslo řádku a proměnná umožňující řádek změnit. Dochází při ní k přepsání pomocné proměnné aktuálního řádku seznamu, na základě které následně dochází k přeplnění obsahu gridu procedurou `RefillGrid`. Využívá se u gridů `OQJednotky` a **OQVlastJed** z důvodu vykreslení na sebe navazujících hodnot.
- **CanEditCell** – umožňuje určit, které sloupce a řádky gridu je možné upravovat

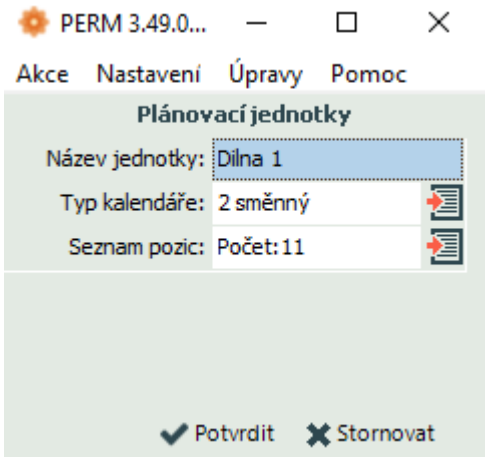


Obrázek 19 – Prvek vývojového prostředí Object Inspector

4.1.2 P3E_PlanJed

Editační formulář *P3E_PlanJed* (Příloha P I, soubor [4]) slouží k přidání nového záznamu do seznamu předdefinovaných plánovacích jednotek nebo k jeho úpravě. Je inicializován standardně procedurou `InitForm`, k přeplňování používá opět `RefillGrid` a k obsluze událostí jeho gridu `OQedity` slouží stejné procedury jako u hlavního formuláře *P3Q_PlanJed*, proto zde i u dalších formulářů nebudou znovu specifikovány. Podoba formuláře je zobrazena na obrázku 20.

K jeho volání dochází na záložce Seznam jednotek formuláře *P3Q_PlanJed*. Prvním způsobem je stisk tlačítka + pro vložení nového záznamu, při kterém dochází k vyvolání akce `ACInsertExecute`. Druhým je dvojklik na položku seznamu nebo její označení a stisk tlačítka ^ pro úpravu záznamu, čímž je vyvolána akce `ACEditExecute`. Formulář obsahuje veřejnou proměnnou `EDefPlanJednotka`, která je při inicializaci nastavena na konkrétní záznam, se kterým pracuje, a jehož vlastnosti jsou upravovány.



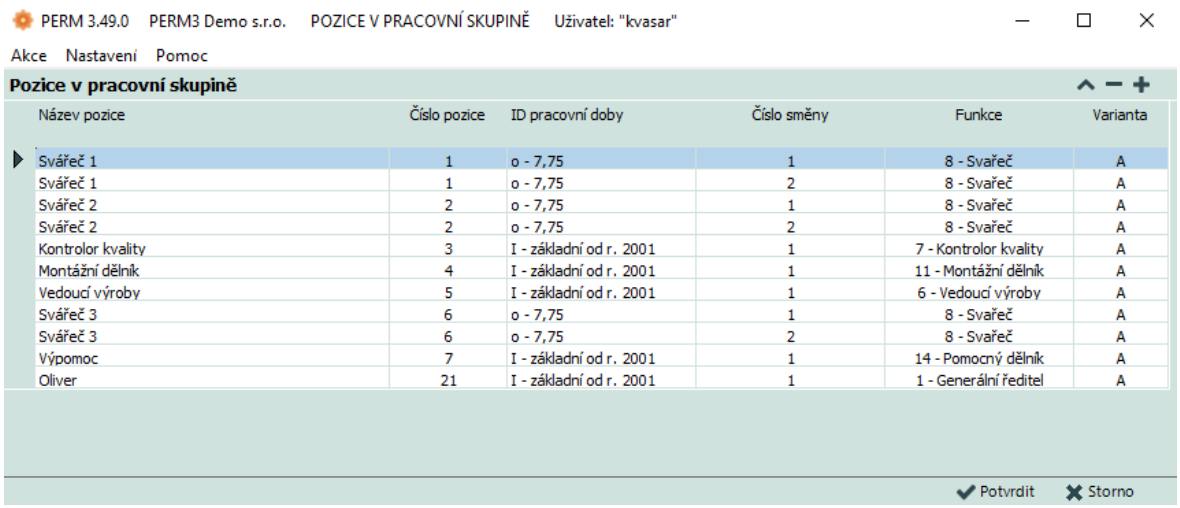
Obrázek 20 – Vzhled editačního formuláře *P3E_PlanJed*

Kromě pole pro zadání názvu jednotky umožňuje výběr kalendáře pomocí tlačítka `BQvyberKalendare`, jehož stisk vyvolá akci `ACvyberKalendareExecute`. Tato akce provede volání předdefinovaného formuláře pro výběr kalendáře firmy Kvasar *P3V_RozvrhSmen*. Jelikož je potřeba údaj o čísle kalendáře (`ciskal`) a zároveň je nutné dát uživateli možnost si vybrat, je zvolen tento přístup. Druhé tlačítko `BQseznamPozic` vyvolá akci `ACvyberPozicExecute`, která má obdobnou podobu jako akce pro editaci záznamu. Dochází v ní k inicializaci a následné prezentaci formuláře *P3V_PlanJedBunky*, po jehož uzavření je prováděna detekce změn a překreslení voláním procedury `RefillGrid`.

4.1.3 P3V_PlanJedBunky

Formulář *P3V_PlanJedBunky* (Příloha P I, soubor [5]) je definován jako výběrový. Důvodem není výběr konkrétních buněk, ale to, že nevyvolává svým potvrzovacím tlačítkem ukládání, ale pouze ukončení náhledu na seznam buněk. Slouží tedy pro prezentaci a úpravu podřízeného seznamu buněk `DefBunkaList`, přiřazeného k editované předdefinované plánovací jednotce `EDefPlanJednotka`. Jeho podoba je zobrazena na obrázku 21.

Seznam buněk je možné opět upravovat editačními tlačítky. Tlačítko + vyvolá akci `ACInsertExecute` pro vložení nového záznamu do seznamu buněk. Následně je inicializován editační formulář *P3E_PlanJedBunky* a plněny vlastnosti záznamu. Stisk tlačítka pro úpravu ^ nebo dvojklik na položku seznamu vyvolá akci `ACEditExecute`, která obdobně inicializuje a zobrazí editační formulář s hodnotami upravovaného záznamu. Při potvrzení dojde k ponechání nových údajů, v opačném případě k jejich zahození. Mazání záznamu je zajištěno vyvoláním dialogu, a při potvrzení je záznam ze seznamu odstraněn.

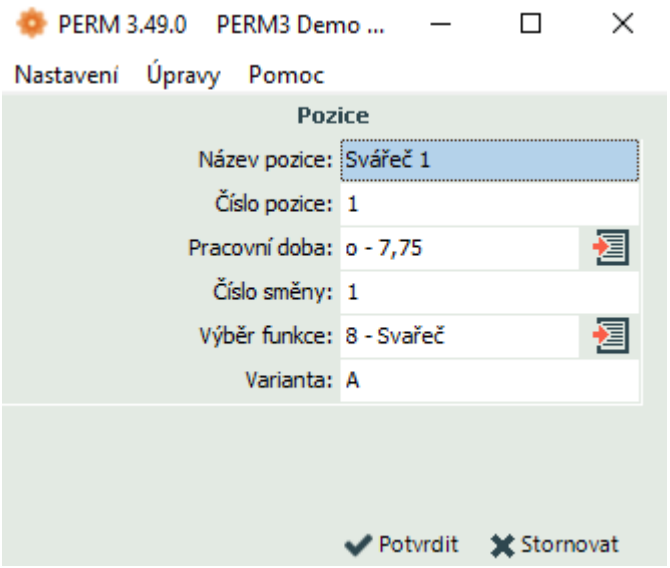


Název pozice	Číslo pozice	ID pracovní doby	Číslo směny	Funkce	Varianta
Svářeč 1	1	o - 7,75	1	8 - Svařeč	A
Svářeč 1	1	o - 7,75	2	8 - Svařeč	A
Svářeč 2	2	o - 7,75	1	8 - Svařeč	A
Svářeč 2	2	o - 7,75	2	8 - Svařeč	A
Kontrolor kvality	3	I - základní od r. 2001	1	7 - Kontrolor kvality	A
Montážní dělník	4	I - základní od r. 2001	1	11 - Montážní dělník	A
Vedoucí výroby	5	I - základní od r. 2001	1	6 - Vedoucí výroby	A
Svářeč 3	6	o - 7,75	1	8 - Svařeč	A
Svářeč 3	6	o - 7,75	2	8 - Svařeč	A
Výpomoc	7	I - základní od r. 2001	1	14 - Pomocný dělník	A
Oliver	21	I - základní od r. 2001	1	1 - Generální ředitel	A

Obrázek 21 – Vzhled formuláře pro seznam definovaných pracovních pozic plánovací jednotky *P3V_PlanJedBunky*

4.1.4 P3E_PlanJedBunky

Editační formulář *P3E_PlanJedBunky* (Příloha P I, soubor [6]) slouží pro editaci jednotlivých pracovních pozic (buněk) v předdefinované pracovní jednotce. Upravovaná buňka je reprezentována veřejnou proměnnou `EDefBunka`, která je při inicializaci nastavena na konkrétní záznam, jehož vlastnosti jsou upravovány. Formulář má podobu uvedenou na následujícím obrázku 22.



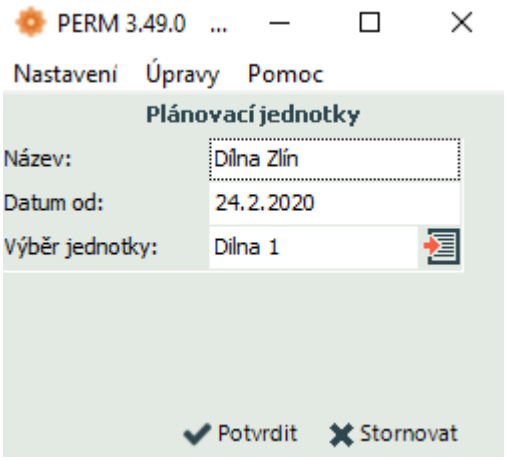
Obrázek 22 – Vzhled editačního formuláře *P3E_PlanJedBunky*

Formulář umožňuje ruční zápis hodnot a u položek „Pracovní doba“ a „Výběr funkce“ jejich zprostředkovaný výběr pomocí výběrových formulářů. Pro výběr jsou použita dvě tlačítka. Prvním je `BQPracDoby`, jehož stisknutím dojde k inicializaci a zobrazení předdefinovaného formuláře pro výběr typu pracovní doby *P3V_TyPrDo*, díky kterému je možné zvolit buňce identifikátor pracovní doby (`ideprd`). Druhým tlačítkem je `BQFunkce`, které po stisknutí inicializuje a zobrazí výběrový formulář P3V_Funkce. Tento výběrový formulář nabízí výčet dostupných funkcí ve firmě, a na základě vybrané položky je poté určen kód funkčního místa (`kodfum`) pro danou buňku. Obsluhu událostí při editaci, zobrazení a validaci zajišťují opět procedury vytvořené pomocí Object inspectoru.

4.1.5 P3E_PlanKap

Editační formulář *P3E_PlanKap* je určen k přidání nového záznamu do seznamu fragmentů nebo k jeho úpravě. Má podobu uvedenou na obrázku 23 a jeho zdrojový kód je přiložen v příloze P I, soubor [7]. Formulář může být volán na záložce Výběr jednotek formuláře *P3Q_PlanJed* vyvoláním dvou různých akcí. Prvním způsobem je stisk tlačítka + pro vložení nového záznamu, při němž dochází k vyvolání příslušné akce `ACVlastJedInsertExecute`. Druhým je dvojklik na položku seznamu nebo její označení a stisk tlačítka ^ pro úpravu záznamu, a vyvolání tak akce `ACVlastJedEditExecute`. Formulář obsahuje veřejné proměnné `EPlanJednotka`, která je při inicializaci nastavena na příslušnou hlavičku plánovací jednotky, `EFragment`, která je při inicializaci nastavena

na upravovaný fragment a `DefPlanJednotkaList`, která slouží pro přehled dostupných předdefinovaných plánovacích jednotek pro přiřazení do vazby.

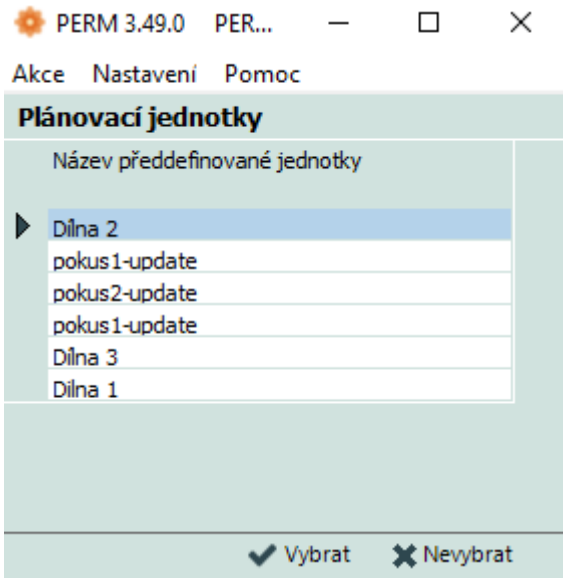


Obrázek 23 – Vzhled editačního formuláře *P3E_PlanKap*

Výběr definice jednotky je zajištěn stisknutím tlačítka `BQVyber`, které vyvolá inicializaci a zobrazení formuláře pro přiřazení typu předdefinované jednotky *P3V_VybDefJed*. Na základě vybrané položky je fragmentu zapsána hodnota `intplj` a konkrétní jednotka získává vlastnosti vybrané předdefinované jednotky.

4.1.6 P3V_VybDefJed

Již výše uvedený výběrový formulář *P3V_VybDefJed* je přiložen v příloze P I, soubor [8] a slouží pro výběr předdefinované plánovací jednotky a přiřazení vazby na ni fragmentu (Obrázek 24). Je inicializován a obsluhován stejným způsobem jako ostatní formuláře a zobrazuje výčet předdefinovaných plánovacích jednotek na základě proměnné `DefPlanJednotkaList`, jehož hodnoty předány při inicializaci. Obsahuje dvě další proměnné `AktDefPlan`, která slouží pro aktuálně planou vybranou položku a `VybDefPlan`, která nese hodnoty nově vybrané položky. Po ukončení výběru slouží tyto proměnné pro zjištění, zda byla provedena změna ve výběru.



Obrázek 24 – Vzhled výběrového formuláře předdefinovaných plánovacích jednotek *P3V_VybDefJed*

4.1.7 Použité předdefinované formuláře pro výběr položek

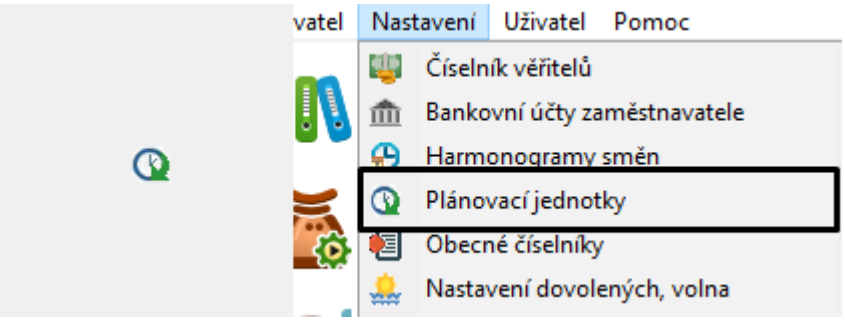
Společnost Kvasar disponuje již hotovými výběrovými formuláři pro stávající řešení, proto bylo možné jejich upravené znovupoužití. Jednalo se o formuláře:

- *P3V_RozvrhSmen* – slouží pro výběr kalendáře určujícího rozvrh směn. Byl využit proto, že vybírané položky nesou hodnotu `ciskal`, díky čemuž je možné usnadnit její vyplňování na základě výběru.
- *P3V_TyPrDo* – zajišťuje výběr typu pracovní doby včetně počtu směn. Je použit z důvodu, že vybírané položky nesou hodnotu `ideprd`, a díky tomu je usnadněno její vyplňování na základě výběru.
- *P3V_Funkce* – poskytuje možnost výběru pracovních funkcí definovaných seznamem funkcí ve firmě. Použitím tohoto výběrového formuláře je usnadněno vyplňování vlastnosti `kodfum` na základě vybrané funkce, která tuto hodnotu obsahuje.

4.2 Po zapracování

Vytvořením tabulek v databázi a přidáním definic tříd a formulářů do projektu aplikace vznikla možnost definovat v aplikaci Perm3 plánovací jednotky a odvozovat od nich další konkrétní definice. Vznikla tak požadovaná funkcionalita a struktura oken aplikace.

Hlavní formulář je spustitelný pomocí položky v hlavním menu Nastavení → Plánovací jednotky.



Obrázek 25 – Ikona a položka v hlavním menu pro vytváření plánovacích jednotek v aplikaci Perm3

Položce hlavního menu `MIPlanJed` typu TKmenuItem byla také vytvořena ikona (Obrázek 25) a přiřazena její vlastnosti `Bitmap`. Další formuláře pak na něj navazují a jsou zobrazovány vyvoláváním patřičných, výše uvedených událostí. Výsledkem jsou vytvořené skelety pracovních jednotek s výčtem buněk, které je třeba následně obsadit zaměstnanci. Toto obsazování bude objasněno v následující kapitole 5.

5 VÝKONNÝ MODUL PRO APLIKAČNÍ SERVER PERMWEBAPI

Tato kapitola je zaměřena na zpracování výkonné knihovny a popisuje její funkcionalitu. Jak již bylo zmíněno, komunikace webové aplikace s aplikačním serverem probíhá na základě HTTP protokolu. Požadavky na API jsou obsluhovány pomocí příslušných knihoven pro každý modul. V tomto případě se jedná o vytvořenou knihovnou *P3k_plankap* (Příloha P I, soubor [9]), která zajišťuje požadovanou funkcionalitu, zpracování dat a jejich poskytnutí. Aplikační server přijímá vstupní požadavky, na základě kterých poté vytváří odpovědi, které jsou prezentovány uživateli. Klade si za cíl práci přístup k datům aplikace Perm3, tedy definicím plánovacích jednotek, pomocí webového rozhraní a zároveň přiřazení zaměstnanců na definované pozice v plánovací jednotce, na základě kterého je možno tvořit plán pro celou jednotku.

5.1 Komunikace použitím XML

Pro práci s knihovnami na aplikačním serveru slouží řetězce s XML strukturou. Ty jsou používány pro vstup i výstup knihovny. Požadavky, které přicházejí z webového klienta, nesou data ve formátu JSON a jsou API zpracovány do podoby vstupního XML řetězce. Výstup knihovny je v podobě výstupního XML řetězce zpracován API a slouží pro vytvoření odpovědi na požadavek, aby mohly být výstupní hodnoty knihovny následně zpětně prezentovány a poté upravovány. Tyto XML řetězce fungují jako dotazy a odpovědi, se kterými je knihovna schopna pracovat. Jsou tvořeny dvěma hlavními částmi `header` a `body` (hlavička a tělo). `Header` obsahuje název knihovny (`name`), která má řetězec zpracovat, název databáze `compid`, se kterou má knihovna pracovat, id uživatele `userid` a `filter`, který úzce specifikuje, která konkrétní data má knihovna zpracovat. Například pro konkrétní plánovací jednotku a období. Další položkou řetězce kromě `header` a `body` je značka `error`, která v sobě nese zprávu o případné chybě při zpracování knihovny, a `protocol` který informuje o zpracování. Struktura je znázorněna na následujícím obrázku 26. [53]

```
<?xml version="1.0" encoding="utf-8"?>
<PermMessage>
  <header>
    <name>k_plankap</name>
    <compid>tester</compid>
    <userid jmepri="">permweb</userid>
    <password />
    <action />
    <actionid />
    <filter>obdobi=202003;</filter>
  </header>
  <error />
  <protocol />
  <body>
  </body>
</PermMessage>
```

Obrázek 26 – Struktura XML řetězce pro práci s knihovnou

5.1.1 Vstupní

Vstupní XML řetězec (Příloha P I, soubor [10]) nese v hlavičce veškeré z výše zmiňovaných parametrů. Na obrázku 27 je možné vidět, jak je struktura doplněna o další uzly. V uzlu body se nachází zaobalující značka (tag) s názvem knihovny a uvnitř ní položky, které mají být editovány, to znamená, že nesou některý z příznaků pro status *i, u, d* uvnitř značky zaobalující konkrétní prvek. Prvním prvkem je dceřiná značka `plan`, která nese informaci o identifikaci konkrétní plánovací jednotky ve značce `idplaj` a dnu, pro který je prováděna úprava, ve značce `planden`. Plán dne obsahuje položky pro určení data, počtu směn, příznak chybějícího přiřazení `prazdnabunka` a samotné pracovní pozice (buňky), kterým jsou přiřazováni pracovníci. Značka `bunka` v sobě nese informace o vybrané pozici, příznak chybějícího přiřazení a značku `pracovnik`, s informacemi o pracovníku, se kterým jsou prováděny akce ve spojitosti s buňkou na základě statusu (přiřazení, upravení, odebrání). Vzhledem k tomu, že k buňce může být přiřazeno více zaměstnanců, může dojít k výskytu seznamu více značek `pracovnik`. Značka `pracovnik` nese informaci o čísle pracovníka, čísle pracovněprávního vztahu, defaultním přiřazení k pozici, aktivitě, oprávnění viditelnosti na webu `cisweb`, a dále čísle anomálie a čísle přiřazení pracovníka k pozici jednotky pro snadnější zpracování úprav. Na základě příznaku `status` jsou poté položky `plan`, `planden`, `bunka` a `pracovnik` patřičně obslouženy knihovnou.

```
<body>
  <k_plankap>
    <plan status="u">
      <idplaj>1</idplaj>
      <planden status="u">
        <evirok>2020</evirok>
        <evimes>3</evimes>
        <eviden>9</eviden>
        <prazdnabunka>true</prazdnabunka>
        <pocetsmen>2</pocetsmen>
        <bunka status="u">
          <cbunky>6</cbunky>
          <nazbun>Obráběč kovů</nazbun>
          <smena1 />
          <smena2>X</smena2>
          <smena3 />
          <smena4 />
          <varianta>A</varianta>
          <cissme>2</cissme>
          <kodfum>9</kodfum>
          <prazdnabunka>false</prazdnabunka>
          <incplj>1025</incplj>
          <pracovnik status="i">
            <incipr>41</incipr>
            <cisppv>41</cisppv>
            <cisweb>0</cisweb>
            <defpri>true</defpri>
            <stav>0</stav>
            <ianopj>-1</ianopj>
            <ipljpp>-1</ipljpp>
          </pracovnik>
        </bunka>
      </planden>
    </plan>
  </k_plankap>
</body>
```

Obrázek 27 – Podoba struktury vstupního XML řetězce

5.1.2 Výstupní

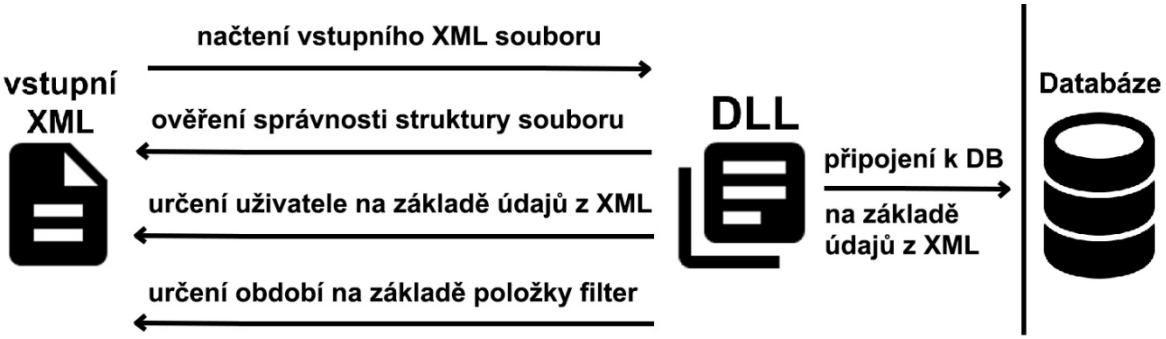
Výstupní XML řetězec, na rozdíl od vstupního, neobsahuje v hlavičce uživatelské jméno, heslo ani filter (Příloha P I, soubor [11]). Tyto položky už nejsou pro podobu odpovědi důležité. Uvnitř značky `body` se nachází výčet příslušných plánů jednotek po zpracování knihovnou, tentokrát však bez příznaků pro status. Samotná struktura záznamů je obdobná s tou ve vstupním řetězci, navíc obsahují plány značku s názvem jednotky a jsou strukturovány do seznamů kompletních dnů (značky `planden`) s příslušnými buňkami a jim přiřazenými zaměstnanci. Dochází tedy ke kompletnímu měsíčnímu výpisu pro prezentaci na webu. Struktura je nastíněna na následujícím obrázku 28.

```
<body>
  <k_plankap>
    <plan>
      <idplaj>1</idplaj>
      <nazjed>Dílna Zlín</nazjed>
      <prazdnabunka>true</prazdnabunka>
      <planden>
        <evirok>2020</evirok>
        <evimes>3</evimes>
        <eviden>1</eviden>
        <prazdnabunka>false</prazdnabunka>
        <pocetsmen>2</pocetsmen>
        <bunka>
          <cbunky>1</cbunky>
          <nazbun>Svářeč 1</nazbun>
          <smena1>X</smena1>
          <smena2></smena2>
          <smena3></smena3>
          <smena4></smena4>
          <varianta>A</varianta>
          <cissme>1</cissme>
          <kodfum>8</kodfum>
          <prazdnabunka>false</prazdnabunka>
          <incplj>1015</incplj>
          <pracovnik>
            <incipr>30</incipr>
            <cisppv>30</cisppv>
            <defpri>false</defpri>
            <stav>1</stav>
            <ianopj>6</ianopj>
            <ipljpp>-1</ipljpp>
          </pracovnik>
        </bunka>
          .
          .
          .
        <bunka>
        </bunka>
      </planden>
          .
          .
          .
      <planden>
        <evirok>2020</evirok>
        <evimes>3</evimes>
        <eviden>31</eviden>
          .
          .
          .
      </planden>
    </plan>
  </k_plankap>
</body>
```

Obrázek 28 – Náhled struktury výstupního XML řetězce

5.2 Knihovna – DLL

Vytvořená výkonná knihovna *P3k_plankap* je v podstatě tvořena pouze jednou hlavní funkcí `ExecuteTask` (Příloha P I, soubor [12]), která přijímá pět argumentů typu PChar. Jsou to argumenty `Input`, `Output`, `AUserName`, `APassword` a `ALog`, které slouží pro vstup, výstup, uživatelské jméno, heslo a log pro logování aktivit knihovny. Důležitou definicí je u této funkce direktiva **`stdcall`**. Ta zajistí, že funkce může být standardně volána i aplikacemi napsanými v jiném jazyce za použití Windows konvencí. Při volání této funkce dochází nejprve k načtení vstupního XML řetězce, vytvoření proměnné výstupního XML řetězce a poté funkce `PMCheck` zjistí, zda má vstupní řetězec správnou strukturu (Obrázek 29). Následně dojde k pokusu o navázání spojení s databází. Po úspěšném spojení se provádí identifikace uživatele na základě přihlašovacích údajů pomocí funkce `GetPermUserFromUserID`. Pokračuje se na čtení položky `filter`, která v sobě nese údaj `obdobi`, na základě kterého může docházet k použití knihovny pro konkrétní vybrané období (měsíc). Jakmile dojde k úspěšnému načtení a ověřeni těchto vstupních parametrů, pokusí se knihovna načíst potřebná data z tabulek a na základě definice lokálních objektů je správně naplnit. [53,54]



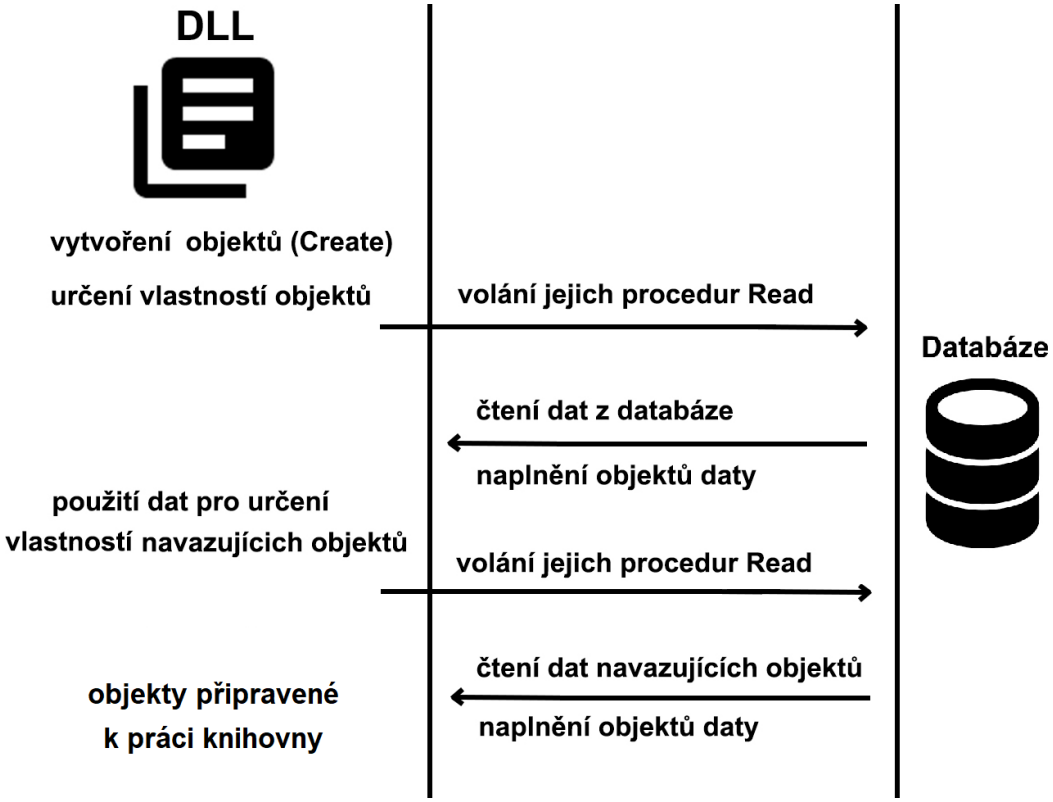
Obrázek 29 – Schéma počátku hlavní funkce knihovny *ExecuteTask* [52,55]

Data je možno získat dvěma způsoby. Prvním způsobem je vytvoření patřičného objektu a zavoláním jeho procedury `Read`, kterou dojde k načtení dat za pomoci objektu `Selecter`, jak bylo popsáno v předchozích kapitolách. Druhým způsobem je lokální vytvoření objektu `Selecter`, určení jeho vlastností a načtení dat v cyklu do lokální proměnné obdobně jako v proceduře `Read`. V první fázi probíhá čtení prvním způsobem do lokálních objektů. Druhý způsob je využit později v části čtení tabulek docházky.

5.2.1 Objekty pro vytváření plánovacích možností

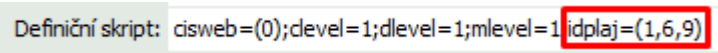
Pro vykonání správného čtení je nutno určit objektům jejich vlastnosti, aby mohla být procedura `Read` vykonána. Vzhledem k tomu, že na sebe objekty odkazují a čtení může vycházet z načtení dat jiného objektu, jsou všechny objekty stejně důležité a je potřeba zajistit jejich načtení a následné přiřazení. Princip načtení dat knihovnou je zobrazen na obrázku 30.

Podstatnými objekty z hlediska plánování a další práce knihovny jsou ve fázi prvotního čtení `LAktUzivatel`, `LDefPlanJednotkaList`, `LPlanJednotkaList`, `LPlanAnomalieList`, `LPlanJednotkaPPVList` a `LRozvrhy`. Dále jsou to objekty `LPPVList`, `LRozvrhSmenPPV`, které jsou nachystány, ale nejsou ještě načteny z důvodu pozdější konkretizace jejich parametrů.



Obrázek 30 – Schéma načtení dat objektů, se kterými knihovna pracuje [52,55]

LAktUzivatel je objekt, který nese informace o uživateli přihlášeném do systému. Na základě tohoto objektu lze určit jeho oprávnění k viditelnosti obsahu a možnosti úprav a interakce se systémem. Tyto informace je možné určit v nastavení aplikace Perm3 (Obrázek 31), kdy je uživatelům určen výčet jednotek zápisem do takzvaného skriptu.



Obrázek 31 – Definice id jednotek ve skriptu uživatele

Tuto definici je možné nalézt jako vlastnost `LAktUzivatel.skript` a následně jsou v knihovně zpracovávány funkcí `GetJednotky`. Výsledkem funkce je pole textových řetězců s id čísly jednotek, ke kterým má uživatel přístup. Na základě tohoto pole dochází následně k tvorbě odpovídajícího výpisu plánů jednotek uživateli. Jedná se o standardní postup firmy Kvasar pro tvorbu a čtení oprávnění uživatelů, a proto je v této práci s drobnými obměnami použit.

LDefPlanJednotkaList v sobě nese definice struktur plánovacích jednotek včetně listu buněk. Tento objekt je následně využíván při načítání dat objektu **LPlanJednotkaList**, který obsahuje jednotlivé konkrétní plánovací jednotky se seznamem jejich vazeb na definici a datem platnosti. Společně tyto objekty definují kostru konkrétní jednotky pro dané období, daný den. Na základě těchto definic je tedy poté možno vyskládat informace o jednotce a obsazovaných pozicích do řetězce ve formátu XML.

Informace o plánování obsazování pozic zaměstnanci nesou objekty `LPlanJednotkaPPVList` a `LPlanAnomalieList`. Objekt **LPlanJednotkaPPVList** sdružuje informace o trvalém přiřazení na danou pozici. To znamená, že je požadováno tohoto zaměstnance plánovat k této pozici i do budoucna. V kombinaci s rozvrhy zaměstnance `LRozvrhSmenPPV` je možné vytvářet implicitní přiřazení na některou z buněk jednotky. Na druhou stranu objekt **LPlanAnomalieList** nese informace o ručních záznamech přiřazení pro konkrétní dny směny a pozice. Záznamy jsou vytvářeny explicitně uživatelem a slouží jako výjimky v případě nespokojenosti s implicitním přiřazením nebo pro ruční vyplnění plánu.

Objekt **LRozvrhy** je zde určen především pro účely načtení již zmiňovaného objektu rozvrhu směn zaměstnanců `LRozvrhSmenPPV` a také pro stanovení počtu směn dané jednotky. Samotný objekt **LRozvrhSmenPPV** slouží pro načtení rozvržení směn zaměstnance v daném období. Za pomoci `LRozvrhů` určuje pro každý den, jakou směnu má plánovanou. Na základě tohoto rozvrhu je poté možno implicitně naplánovat zaměstnance na buňku s odpovídajícím číslem směny.

Načtení rozvrhu směn předchází načtení seznamu pracovněprávních vztahů `LPPVList`. Specifikováním konkrétních vlastností tohoto seznamu je poté možné načíst objekt `LPPV`, který

stanovuje pracovněprávní vztah zaměstnance, se kterým se pracuje, a je podstatný pro načtení jeho rozvrhu směn.

Společným provázáním dat z těchto objektů jsme následně schopni vytvářet plány dané jednotky, s plány pracovat a upravovat je.

5.2.2 Decode XML

Další částí zpracování vstupního požadavku XML řetězce je `body`, které může obsahovat značku `plan` a uvnitř ní záznamy. Jedná se o záznamy se značkou `planden`, na základě které je vytvořena kolekce uzlů `LDnyColl` typu IXMLNodeCollection. Ty jsou následně zpracovány procedurou `DecodeXmlDen`, pomocí které jsou hodnoty záznamů mezi jednotlivými značkami zapsány na příslušné místo do proměnné `LPlanDenXML` typu *TDJednotkaDen* (Obrázek 32). Čtení podřazených značek `bunka` a `pracovnik` je v proceduře `DecodeXmlDen` zajištěno také pomocí kolekcí.

Obrázek 32 – Dekodování struktury XML řetězce a zápis hodnot do příslušných proměnných [55]

Na obrázku 33 jsou zobrazeny vytvořené datové typy, které jsou použity pro obsluhu dekódování XML řetězce, zápis i načtení a zakódování dat pracovní jednotky pomocí XML řetězce. Typ *TDJednotkaDen* v sobě nese pole `BunkyDen` typu *TDBunkaDen* a každá z těchto buněk dále nese pole pracovníků, kteří k ní mohou být přiřazeni, `Pracovnici` typu *TDBunkaPracovnik*. Pro zjednodušení, proměnná slouží k popisu jednoho konkrétního dne, kdy den obsahuje seznam pracovních pozic a pozice nese seznam k ní přiřazeným pracovníků.

```
TDBunkaPracovnik=record
          incipr:integer;
          cisppv:integer;
          defpri:boolean;
          stav:smallint;
          status:Char;
          ianopj:integer;
          ipljpp:integer;
          end;
TDBunkaDen=record
          cbunky:Smallint;
          nazbun:String;
          smena1:String;
          smena2:String;
          smena3:String;
          smena4:String;
          varianta:String;
          cissme:smallint;
          kodfum:Integer;
          prazdnabunka:Boolean;
          status:Char;
          incplj:integer;
          Pracovnici:array of TDBunkaPracovnik;
          end;
TDJednotkaDen=record
        idplaj:integer;
        nazjed:String;
        evirok:Smallint;
        evimes:Smallint;
        eviden:Smallint;
        prazdnabunka:Boolean;
        status:Char;
        pocetsmen:smallint;
        BunkyDen:array of TDBunkaDen;
end;
```

Obrázek 33 – Definice datového typu pro práci s plánovací jednotkou

Záznamy mohou mít ve značce příznak `status`, který rozhoduje o jejich případném vložení, upravení či smazání. Pro status *„i“* a *„d“* nedochází u značky `planden` k žádné činnosti,

protože počet dnů v měsíci je neměnný. Status *„u“* způsobí posun k čtení na úroveň seznamu buněk daného dne. Zde se opět pro status *„i“* a *„d“* nic neděje, jelikož počet buněk je neměnný a struktura je dána definicí plánovací jednotky, status *„u“* informuje o změně obsahu na buňce a posune úroveň čtení na seznam pracovníků dané buňky. Na obrázku 34 lze vidět počátek postupného zanořování při rozhodování.

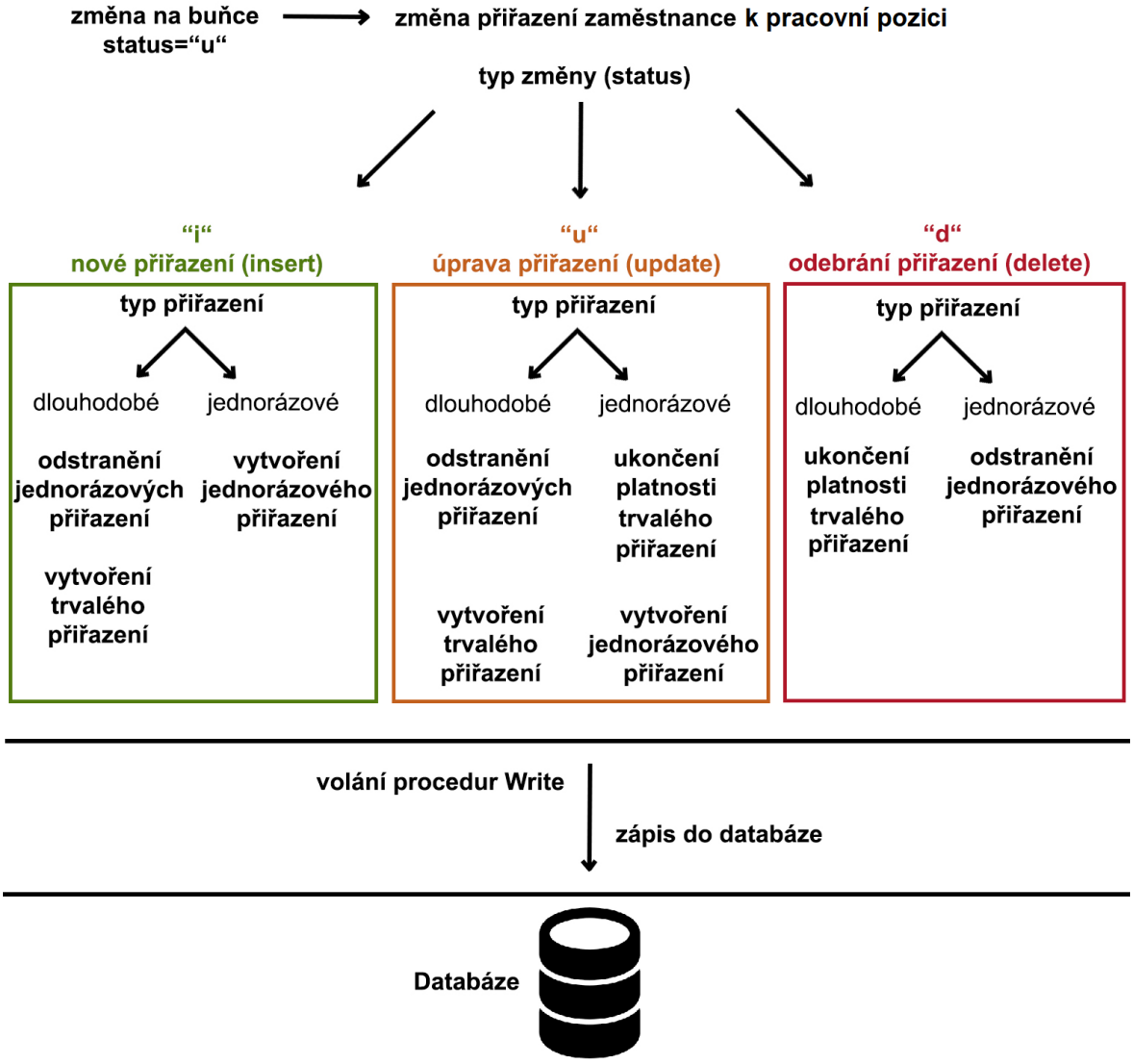
```
GetRowOK:=DecodeXmlDen(LDnyColl.Nodes[i],LPlanDenXml);
if GetRowOK then
begin
LPlanDenXml.idplaj:=Lidplaj;
case LPlanDenXml.status of
  'i':
    begin
      //nevkládám dny, mají daný neměnný počet
    end;
  'u':
    begin
      //vždy když je nějaká změna = anomálie na dnu -> na nějaké buňce/buňkách
      PomDate:=EnCodeDate(LPlanDenXml.evirok,LPlanDenXml.evimes,LPlanDenXml.eviden);
      //projdu na dnu bunky ktere maji status
      for j:=0 to High(LPlanDenXml.BunkyDen) do
      begin
        LBunkaDen:=LPlanDenXml.BunkyDen[j];
        //zjisteni jestli je nekdo prirazen na bunku defaultne
        //LPlanJednotkaPPV:=LPlanJednotkaPPVList.AktualPlanJednotkaPPV(PomDate,LPlanDenX
        case LBunkaDen.status of
          'i':
            begin
              //nevkládám buňky, mají neměnný počet
            end;
          'u':
            begin
              //změna na buňce
```

Obrázek 34 – Programový kód počátku rozhodování o provedení akce na základě statusu uzlů vstupního XML řetězce

`Pracovnici` je pole, se kterým už lze pracovat v rámci buňky a přiřazovat do něj záznamy nebo je editovat. Status *„i“* ve značce `pracovnik` určuje data, zaobalená touto značkou, ke vložení nového záznamu. Podle příznaku `defpri` dojde k vyhodnocení, zda se jedná o vložení do seznamu trvalého přiřazení nebo seznamu anomálií. V případě vložení trvalého přiřazení dochází navíc k nalezení všech budoucích anomálií pro danou buňku a jejich odebrání. Princip rozhodování a navazující provedení akcí jsou vyobrazeny na obrázku 35.

Následně je prováděno hledání nejbližšího budoucího trvalého přiřazení na vybranou pozici pro vybraného pracovníka. Při tomto přiřazení není použito číslo směny, proto je třeba hledat záznamy přes všechny směny a číslo buňky. Jestliže je nalezena buňka, ke které může být přiřazen pracovník, dochází na základě ní k hledání nejbližšího přiřazení vybraného pracov-

níka na pozici pomocí funkce `BudouciPlanJednotkaPPV`. Pokud dojde k nalezení, je datum ukončení nového záznamu přiřazení nastaveno na o jedno nižší, než je datum počátku platnosti nejbližšího nalezeného budoucího trvalého přiřazení. To zajistí, že nebude docházet k překrývání intervalů přiřazení.



Obrázek 35 – Provedení příslušných akcí při dekódování vstupního XML řetězce na základě statusu uzlu `pracovnik` [52]

Pro status *„u“* dochází také k vyhodnocení příznaku `defpri`. Pokrývá změnu z dlouhodobého přiřazení na jednorázové a obráceně. V prvním případě dochází k nastavení data ukončení na datum o jedno nižší a pro zvolený den je vytvořeno jednorázové přiřazení, díky čemuž není předchozí dlouhodobé přiřazení ztraceno. V opačném případě dochází ke smazání všech budoucích anomálií a vytvoření záznamu do seznamu trvalých přiřazení

`LPlanJednotkaPPVList`. Obdobně jako u vkládání záznamu, dochází ke hledání nejbližšího budoucího trvalého přiřazení. Pokud je nalezeno, je datum ukončení nastaveno na o jedno nižší než datum počátku platnosti budoucího přiřazení.

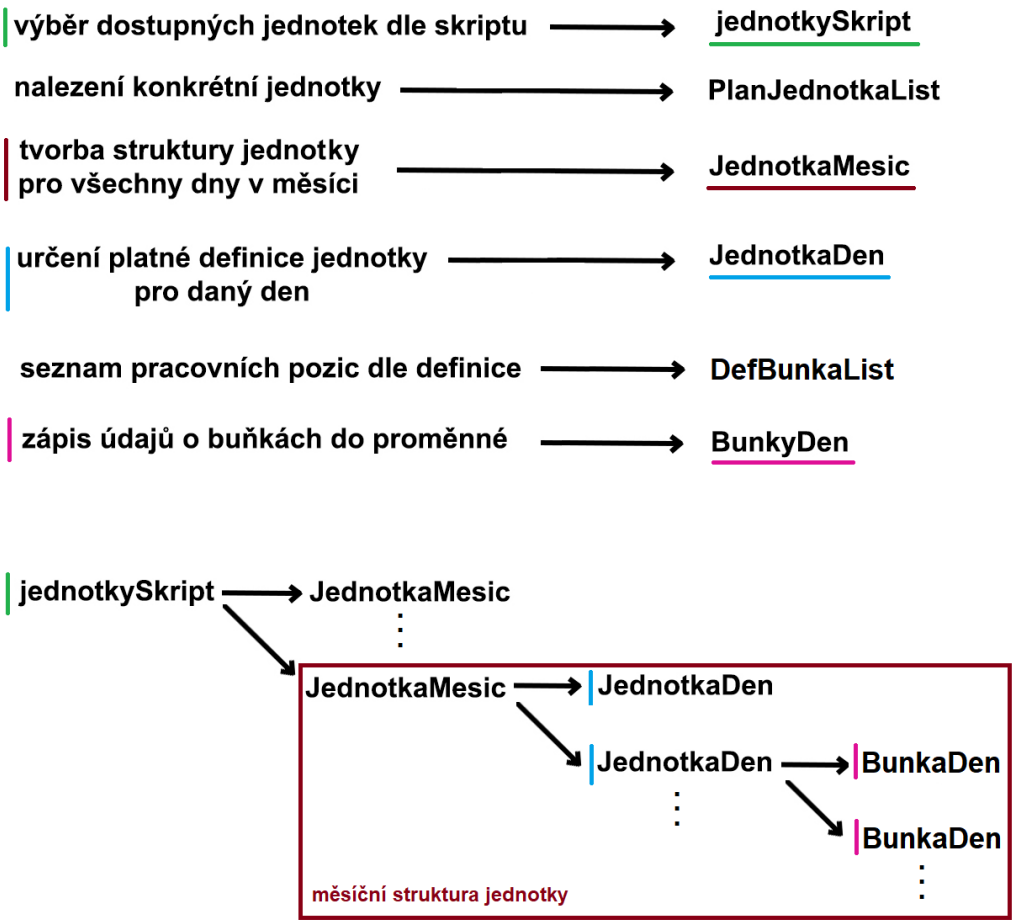
V případě statusu *„d“* dochází k mazání přiřazení pracovníka k buňce. Při požadavku na odstranění trvalého přiřazení dochází k jeho reálnému smazání, pouze pokud se jedná o stejný den, jako je den vzniku přiřazení, v opačném případě dojde k nastavení data konce platnosti a veškeré předchozí implicitní plánování je zachováno. Jednorázová přiřazení jsou naopak ze seznamu anomálií mazána okamžitě na základě příchozího id ve značce `ianopj`.

Nad upravenými objekty `LPlanJednotkaPPVList` a `LPlanAnomalieList` je následně zavolána procedura `Write`, která se pokusí o zápis nových dat do databáze. Při neúspěšném zápisu jsou obnovena původní data procedurami `Restore` a chyba zapsána do logu.

5.2.3 Tvorba obsahu pole LJednotkaMesic

Po dekódování a zápisu příchozích dat dochází ke čtení a prezentaci aktuálního stavu plánů jednotek. K tomu slouží již zmiňované pole identifikátorů plánovacích jednotek `jednotkySkript`. Ten je cyklicky procházen a na základě aktuálně vybraného `idplaj` je zvolena jednotka, se kterou se bude pracovat. Definice této jednotky je tvořena do proměnné `LJednotkaMesic`, ve které je nejprve vytvořena struktura jednotky a následně jsou do ni přiřazovány vazby na pracovníky.

Struktura je vytvářena postupně po jednotlivých dnech, pro které je hledána aktuálně platná definice jednotky (funkcí `FragmentDleData`), pomocí které je určen seznam pozic pro daný den. Seznam definovaných pozic je následně cyklicky procházen a jsou vytvářeny záznamy do subpole `BunkyDen`. Zde jsou postupně každé buňce přiřazovány definované vlastnosti a jsou naplněny patřičné proměnné (Obrázek 36).



Obrázek 36 – Postup vytváření struktury plánovací jednotky pro jeden měsíc na základě definic

Bezprostředně poté je pomocí cyklu procházejícím seznam definovaných buněk inicializováno přiřazení pracovníků. Nejprve jsou cyklicky hledány záznamy trvalého přiřazení pracovníků k buňce funkcí `AktualPlanJednotkaPPV`, ovšem jen ty, které mají v daný den počátek platnosti. Tyto nalezené záznamy jsou se svými vlastnostmi zapsány do subpole `Pracovnici`. Tím dojde k jejich inicializaci v poli, aby s nimi bylo možné dále pracovat, plánovat pro ně další dny. Následuje cyklické čtení seznamu anomálií přiřazení pro daný den a danou buňku. Nalezené záznamy jsou obdobně vloženy do subpole `Pracovnici`, a tím je čtení výjimek přiřazení pracovníků na buňku pro účely prezentace dokončeno.

5.2.4 Čtení dat pro plánování

Čtení dat pro tvorbu plánů je v této fázi teprve na začátku a data v poli obsahují pouze informace o zaměstnancích a struktuře pro jednotlivé dny plánů. Nyní je třeba jim doplnit údaje o rozvrhu směn, absencích v docházce a výskyty v mimoevidenčním stavu. Pro tyto účely byl definován typ *TDDovPrehl* a pole tohoto typu `LArrDovPrehl`, které slouží pro přehled

rozvrhu směn a absencí v docházce. Dále byla definována proměnná `LArrMES` jako pole mimoevidenčních stavů pracovníků (Obrázek 37).

Proměnná `LArrMES` je plněna čtením informací z databáze druhým uvedeným způsobem. Pro čtení je použit lokální objekt `LSel` typu TKvaSelecter, kterému jsou určeny parametry pro provedení SQL dotazu. Jedná se o standardní čtení mimoevidenčních stavů z tabulky *pprmes* a zjišťování výskytu stavů v daném období, v našem případě měsíci. Pokud se v tomto období vyskytuje některý zaměstnanec v mimoevidenčním stavu, je do pole `LArrMES` vložen nový záznam s příslušnými daty o zaměstnanci a délce tohoto stavu. Tyto informace je možné později využít při plánování.

```
type
TDDovPrehl=record
           incipr:Integer;
           cisppv:Integer;
           oscipr:String;
           rozvrh:String;{rozvrh směn}
           mapado:String;{docházka}
           {denní docházka}
           den:array [1..31] of record
                    eviden:smallint;
                    evimes:smallint;
                    evirok:smallint;
                    cissme:smallint;
                    kodpri:smallint;
                    znarep:string;
                   end;
           end;
var
LArrDovPrehl:array of TDDovPrehl;
LArrMES:array of record
                 incipr:Integer;
                 cisppv:Integer;
                 MESdne:array[1..31]of boolean;
                 allmes:boolean;
                 end;
```

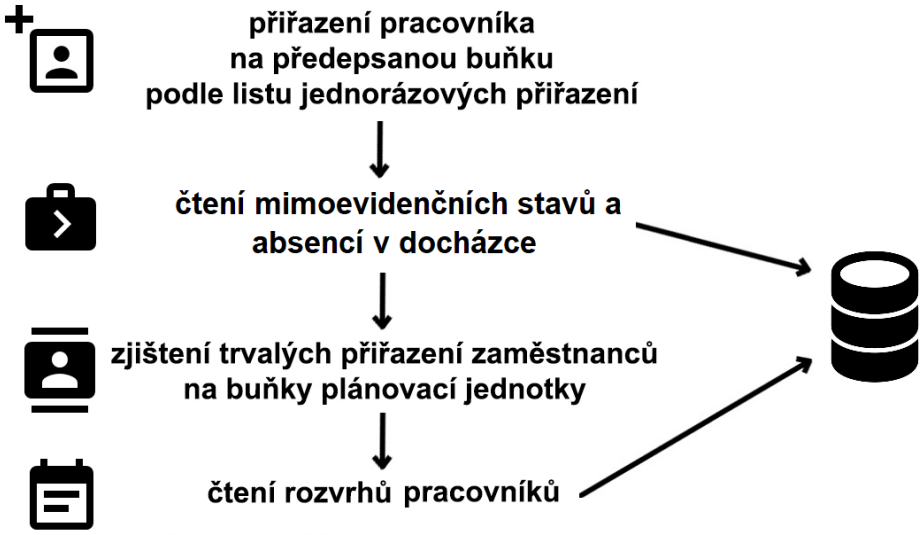
Obrázek 37 – Definice proměnných pro zápis rozvrhu směn, absencí v docházce a výskytu v mimoevidenčních stavech

Následuje čtení dat docházky. Ke čtení je opět využito třídy TKvaSelecter a zadání jeho parametrů pro tvorbu SQL dotazu do databáze. Odpovědi na dotazy jsou cyklicky zpracovávány a zapisovány do pole `LArrDovPrehl`. Po naplnění pole jsou k dispozici řetězce docházkových kódů `mapado` a subpole dnů s docházkovými kódy i číslem směny pro daný

den. Všechna tato data jsou vhodná pro následující plánování z hlediska kontroly dostupnosti pracovníka v daný den.

5.2.5 Plánování

Plánování pracovníků na pracovní pozice je podstatnou částí této práce. V knihovně je řešeno na základě záznamů v tabulce trvalých přiřazení *kpljpp* (objekt `LPlanJednotkaListPPV`) a jejich osobního rozvrhu směn `LRozvrhSmenPPV`.

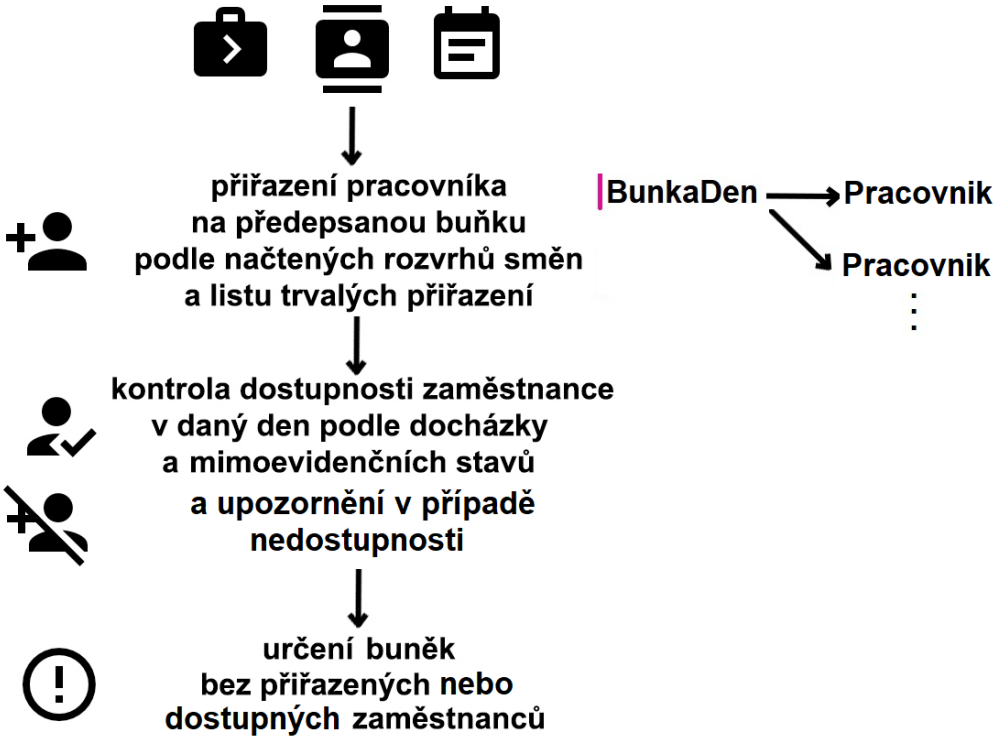


Obrázek 38 – Schéma inicializace plánování přiřazení zaměstnanců na pracovní pozice plánovací jednotky [52,55]

Plánovaní předchází zápis jednorázových přiřazení a čtení dat pro zjištění dostupnosti zaměstnance pro vybrané období (Obrázek 38). V cyklu je procházen seznam trvalých přiřazení, ve kterém jsou nejprve vyhodnocovány podmínky platnosti záznamu pro vybranou jednotku. Ke čtení osobního rozvrhu dochází až tehdy, pokud jsou podmínky splněny. V tom případě probíhá specifikace konkrétních údajů o pracovníku a dochází ke standardnímu čtení rozvrhu na základě seznamu pracovněprávních vztahů a procedury `LRozvrhSmen.SQL.Read`. V této fázi je tedy určeno, že pro tohoto pracovníka je nutné ve vybraném měsíci vytvořit plán.

V případě úspěšného načtení rozvrhu pracovníka dochází k cyklickému průchodu všech dnů v daném měsíci. Na základě konkrétního dne jsou objektu `LRozvrhSmenPPV` přisouzeny parametry `Rok`, `Mesic`, `Den`, pomocí kterých je možné vyhodnocovat funkce objektu. Jedná se především o funkci `CisloPlatneSmeny`, která vrací číslo rozvrhované směny pracovníka pro daný den. Tato hodnota je zapisována do řetězce `rozvrh` příslušného záznamu

v poli `LArrDovPrehl`, čímž je vytvářena mapa směn rozvrhu. Také je třeba určit pro daný den platnou definici jednotky a pracovní pozici, na kterou je záznam vázán. Důvodem je, že definice struktury jednotky může být změněna ze dne na den, když například dojde k rozšíření počtu pracovních pozic nebo úplné změně definice pracovní jednotky. Je nutné brát tyto možné změny v potaz a plánovat pracovníka pouze na tu pozici, na kterou byl přiřazen, a pouze do té doby, kdy je definice jednotky platná. K určení aktuální definice k zadanému datu slouží funkce `LPlanJednotka.FragmentDleData`. Následně je pomocí funkce `LDefPlanBunkaList.DefBunkaDleKodu` zjišťováno, zda se buňka, na kterou má záznam `LPlanJednotkaPPV` vazbu, skutečně nachází v definici jednotky.



Obrázek 39 – Schéma plánování přiřazení pracovníků na pracovní pozice plánovací jednotky [55]

Na obrázku 39 jsou schematicky znázorněny kroky plánování. Ty také zahrnují kontrolu dostupnosti pracovníka na základě jeho rozvrhu směn. Dochází ke zjištění, zda je daný den státní svátek a zda má být i v den svátku odpracována směna pracovníkem. Tato informace je určena načteným rozvrhem směn pracovníka, konkrétně funkcemi `LRozvrhSmenPPV.JeSvatek` a `LRozvrhSmenPPV.SvatekBudeOdpracovan`. Pokud výsledkem obou funkcí je shodně `true`, je možné tohoto pracovníka naplánovat na směnu, v opačném případě nebude k dispozici pro plánování.

Na základě těchto informací lze určit, zda má u tohoto zaměstnance a u této buňky dojít k plánování pro daný den a danou směnu. Podmínky pro plánování jsou:

- Číslo plánované směny musí nabývat kladné hodnoty, aby bylo splněno, že má pracovník pro daný den plánovánu některou směnu z rozvrhu směn. Na tuto směnu je poté plánován v rámci buňky.
- Přiřazení pracovníka na buňku musí být pro daný den platné. Počátek platnosti musí být menší nebo roven danému dnu a Ukončení platnosti přiřazení musí být větší nebo rovno danému dnu.
- Definice buňky, na kterou je zaměstnanec přiřazen, musí být pro daný den známá (rozdílná od hodnoty `nil`). Pokud se daný den buňka, na kterou je pracovník přiřazen, nevyskytuje v definici jednotky, k plánování nedochází.

V případě splnění podmínek pro plánování jsou údaje zapsány na příslušné místo do pole měsíčního plánu jednotky. Jak již bylo zmíněno, k plánování dochází cyklicky pro celý měsíc, a proto jsou podmínky vyhodnocovány pro každé platné přiřazení zaměstnance k jednotce a každý den v měsíci.

Nyní jsou k dispozici data měsíčního naplánovaní zaměstnanců a jednotky. Nad těmito daty je však nutné provést kontrolu, zda jsou přiřazení zaměstnanci v daný den a danou směnu skutečně k dispozici. Důvodem je to, že data jsou naplánována na základě rozvrhů směn. Rozvrhy nám však jen definují, jak by měl zaměstnanec chodit do práce, nikoli jaký je reálný stav. K tomu je potřeba docházka, absence zaměstnance a výskyty v mimoevidenčním stavu. Tyto data už jsou načtena a k dispozici v poli `LArrDovPrehl`. Nyní je lze použít ke kontrole.

Kontrola probíhá cyklickým průchodem celého měsíčního plánu jednotky po dnech, všech buněk a pracovníků k nim přiřazených. Vždy je kontrolován jeden pracovník přiřazený k buňce. Pokud je zjištěno, že v daný den a danou směnu není k dispozici, je jeho `stav` nastaven na hodnotu -1. Nejprve dochází ke zjištění, zda se nachází v mimoevidenčním stavu funkcí `inMes`. Pokud ano, není k dispozici, pokud ne, kontrola pokračuje ke zjištění čísla směny. V případě, že směna není specifikována a je nulová, není k dispozici celý den. Pokud je směna známa, není k dispozici pro tuto směnu. Dále dochází k samotné kontrole s absencemi v docházce. Při tom je porovnávána mapa docházky příslušného záznamu z pole `LArrDovPrehl` s výčtem docházkových kódů, které symbolizují, že je zaměstnanec k dis-

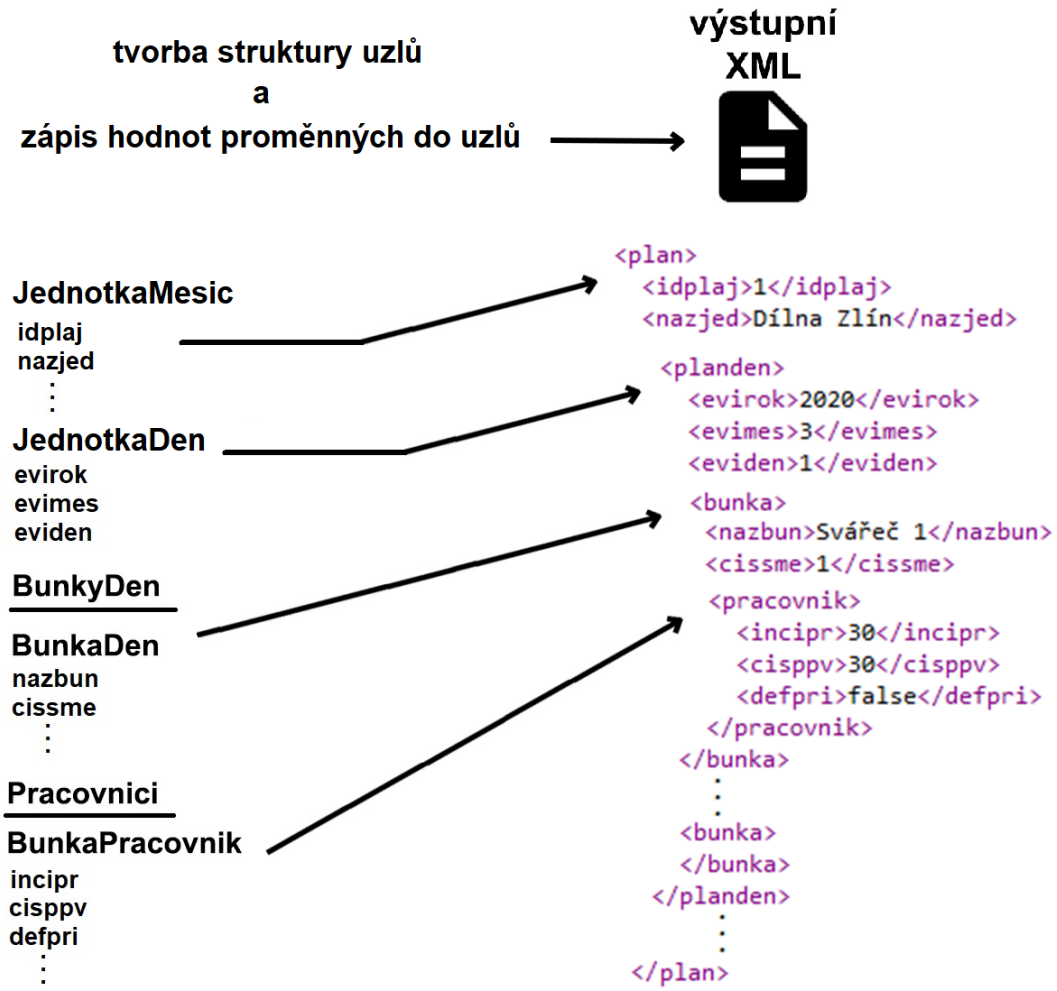
pozici. Pokud se na pozici `mapado` shodné s číslem daného dne nachází jiný znak, zaměstnanec není pro tento den k dispozici. Následně dochází k obdobné kontrole s mapou rozvrhu směn, zda je zaměstnanec naplánován na správnou směnu. V případě, že se na pozici `rozvrh` shodné s číslem daného dne nenachází číslo směny shodné s číslem směny buňky, na kterou je zaměstnanec přiřazen, není pro toto přiřazení k dispozici. Poslední krok, ke kterému zde dochází, je zjištění, zda se na buňce vyskytuje alespoň jeden přiřazený zaměstnanec, který je k dispozici (`stav>-1`). Pokud ne, je tato buňka označena jako prázdná, tedy její příznak `prazdnabunka` je nastaven na hodnotu `true`. Při její prezentaci je díky tomu možné dát najevo, že je potřeba obsadit.

Je třeba zdůraznit, že implicitní plánování je vytvářeno pro trvale přiřazené pracovníky, kdežto jednorázové (anomálie) plánovány nejsou. Jedná se totiž o jednorázové změny a výluku od definovaného stavu, a právě proto je u nich umožněno je vytvářet nezávisle.

Tímto způsobem je vyhotovený měsíční plán jednotky na základě definic, přiřazení a kontrol. Nyní je třeba zpracovaná data poskytnout jako výstup knihovny.

5.2.6 Tvorba výstupního XML

Abychom mohli prezentovat zpracovaná data a vytvořit odpověď na vstupní XML řetězec, je zapotřebí správně vytvořit výstupní XML řetězec. Ke tvorbě struktury slouží uzly definované proměnnými typu IXMLNode. Tyto proměnné mohou obsahovat *ChildNodes*, což jsou jejich podřazené uzly. Tímto způsobem je možné strukturu libovolně rozšiřovat a větvit.



Obrázek 40 – Enkódování dat do struktury XML řetězce a zápis hodnot proměnných do příslušných uzlů [55]

Struktura výstupního XML je částečně vyobrazena na obrázku 28. Při tvorbě výstupu je nejprve vytvářena proměnná `oXML` typu IXMLDocument, které je přisouzen název úlohy a id databáze. Poté je vyhledáván uzel podřazený uzlu body. K této inicializaci dochází na začátku hlavní funkce `ExecuteTask`. Následně až po načtení dat a plánování probíhá práce s tímto nalezeným uzlem `body`, kterému je vytvářen obsah. Následně se jedná o vytvoření jeho struktury. Nejprve je vytvořen podřazený uzel s názvem úlohy `k_plankap`, do kterého budou přidávány uzly `plan`, které zaobalují plán jedné konkrétní jednotky. Plánů se může ve výpisu vyskytovat více. Struktura XML plánu odráží strukturu proměnné `LPlanJednotkaMesic` (Obrázek 40). Je vytvářena cyklicky pro každý den v měsíci, buňky a přiřazené pracovníky pomocí funkcí `NewRow`, `NewDen`, `NewBunka` a `NewPracovnik`. Funkce přijímají jako parametr uzel, kterému bude podřazená struktura vytvořena. Následně jsou

vždy do vytvořené struktury uzlů zapsány příslušná data z proměnné `LPlanJednotkaMesic`, případně údaje o definici jednotky z `LPlanJednotka`. Funkce `NewRow` vytváří nový uzel `plan` a jeho podřazené uzly pro základní informace jednotky. `NewDen` slouží pro tvorbu uzlu `planden` a podřazených uzlů s informacemi o daném dnu. Uzel `planden` je podřazený uzlu `plan` a vyskytuje se u něj tolikrát, kolik je počet dnů v měsíci. Navazují na něj uzly `bunka`, které jsou mu podřazeny a vytvářeny funkcí `NewBunka` společně s jejími podřazenými uzly pro informace o definici pracovní buňky. Počet buněk se odvíjí od definice jednotky. Poslední úroveň struktury je tvořena funkcí `NewPracovnik`, která vytváří buňce podřazený uzel `pracovnik` a jemu podřazené uzly pro popis informací o přiřazeném pracovníku.

Nyní je takto vytvořený výstupní XML řetězec nachystaný k uložení a k dispozici pro výstup knihovny.

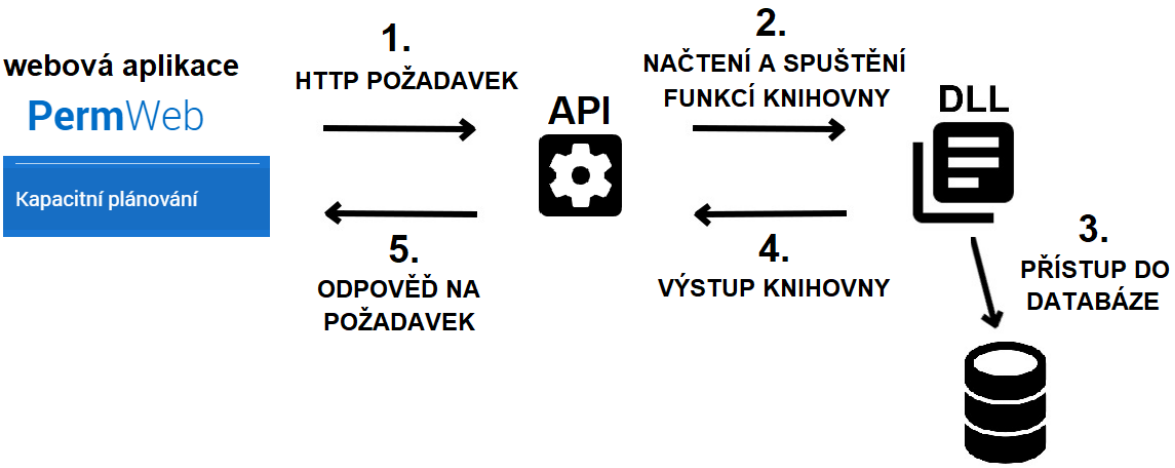
5.2.7 Ukončení práce knihovny

Jakmile dojde k ukončení výpočtů plánování a je vytvořen výstupní XML řetězec, je nutné provést rutinní kroky pro vyčištění obsazené paměti. To je provedeno zavoláním procedury `Destroy` nad všemi vytvořenými objekty a vymazáním obsahu proměnných polí. Následně dochází k uložení výstupního XML řetězce do souboru funkcí `oXML.SaveToXML`. Nakonec je reference na vzniklý výsledný XML řetězec uložena jako výstupní parametr knihovny **Output**.

6 ROZHRANÍ APLIKACE PERMWEB

Webová aplikace PermWeb slouží pro práci s daty systému Perm3 pomocí internetového prohlížeče odkudkoliv bez jakékoliv instalace, jako nadstavba aplikace Perm3 pro jednoduchý přístup zaměstnanců či manažerů. Jedná se modulární systém, který je vyvíjen pomocí technologie Angular, s využitím REST API, které zajišťuje zpracování požadavků na poskytnutí dat, a tvorby jednotlivých DLL knihoven pro aplikační server.

Pro účely zpracování dat daným způsobem je použito DLL knihoven, ty však musejí být správně načteny a obslouženy. Tyto operace vykonává API společnosti Kvasar, které obsluhuje webové požadavky a poskytuje přístup k datům. Prezentace poskytnutých dat ve webovém klientu, manipulace s nimi a tvorba nových požadavků jsou vytvářeny za použití již zmiňovaného frameworku Angular. Schéma komunikace mezi jednotlivými částmi je zobrazeno na následujícím obrázku 41.



Obrázek 41 – Schéma komunikace webové aplikace, API a DLL knihovny [55]

Knihovně *P3k_plankap* bylo třeba vytvořit rozhraní v rámci webové aplikace PermWeb a zapracovat ho jako modul do stávajícího řešení za použití standardních postupů společnosti Kvasar. Ty byly v rámci této práce poskytnuty firmou a šablonové prvky použity k sestavení řešení standardními způsoby a doplněním o konkrétní údaje. API bylo firmou doplněno o zpracování požadavků vytvářených tímto modulem. Řešení využívá také již vzniklých modulů aplikace PermWeb a je vytvořeno doporučenými postupy společnosti.

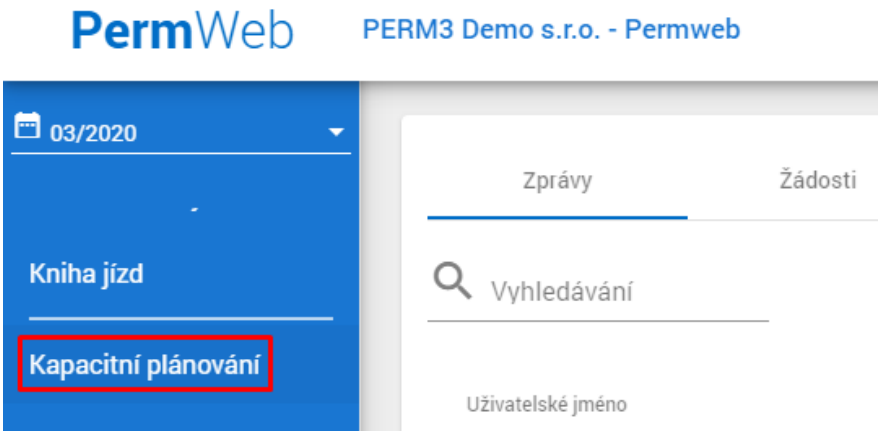
Rozhraní slouží pro plánování přiřazení zaměstnanců na pracovní pozice plánovací jednotky, jejíž struktura byla vytvořena v aplikaci Perm3. V rozhraní webové aplikace je možné tuto strukturu prezentovat, plnit a s využitím knihovny plánovat její obsah. Vytvořené rozhraní

sestává z komunikace s API pomocí protokolu HTTP, zpracování dat pomocí výkonné knihovny a prezentace dat za použití Angular Material Designu. S prezentovanými daty je možné manipulovat a upravovat je. Společně tyto části vytváří intuitivní a přehledné prostředí pro uživatele pracujícího s plány jednotek.

6.1 Čtení a prezentace dat

Prvním krokem pro práci v aplikaci PermWeb je přihlášení uživatele. Zadáním a odesláním přihlašovacích údajů dochází k vytvoření požadavku na autentizaci a tento požadavek je odeslán ke zpracování API. Při úspěšném ověření přihlašovacích údajů je uživateli udělen přístup do aplikace, a na základě jeho údajů a oprávnění vygenerován přístupový token (Access Token), společně s obnovovacím tokenem (Refresh Token). Přístupový token je velmi důležitý, jelikož zajišťuje uživateli přístup k datům a funkcím aplikace, a je odesílán jako součást každého dalšího HTTP požadavku API.

Po úspěšném přihlášení dojde k zobrazení hlavní stránky společně s nabídkou modulů, se kterými je možné pracovat. Aplikace využívá techniky Lazy-Loading, díky čemuž nedochází k inicializaci všech modulů bezprostředně po spuštění aplikace. Jednotlivé moduly jsou dynamicky připojovány, jakmile je vyžadována práce s nimi. V našem případě se jedná o položku Kapacitní plánování, jak je znázorněno na obrázku 42. Výběrem této položky dochází v kódu webové aplikace k připojení modulu kapacitního plánování a volání jeho servisní služby. Ta zajišťuje vytváření HTTP požadavků API a přihlašování se k odpovědím. Je zde využito *Observable* objektů, na základě kterých je možné provádět asynchronní zpracování požadavků na data.



Obrázek 42 – Výběr položky Kapacitní plánování v hlavním menu

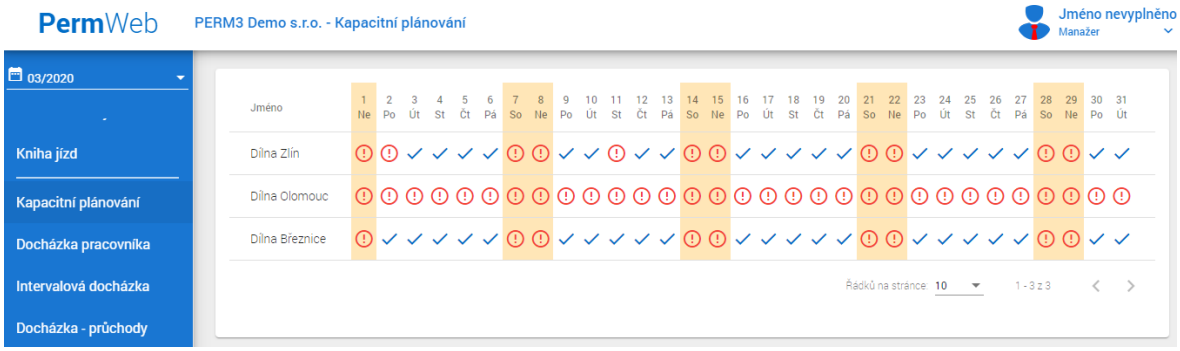
Tento modul primárně slouží pro tvorbu plánů jednotek a přiřazování zaměstnanců na pracovní pozice, ke své práci ovšem také využívá již existující modul určený k prohlížení seznamu pracovníků pro účely prezentace zaměstnanců. V obou případech dochází k vytvoření požadavků směrem k API a získání dat. API zpracovává požadavek na základě přiloženého přístupového tokenu. V případě platnosti tokenu a oprávněnosti k provedení požadovaného úkonu dochází k jeho vykonání.

Načítání dat probíhá za použití vytvořené knihovny *P3k_plankap*, respektive její hlavní funkce `ExecuteTask`. Díky definici funkce s direktivou **`stdcall`** je možné ji standardně volat i z API. Knihovnu však musí API nejprve načíst. K tomu jsou zapotřebí standardní funkce operačního systému (*LoadLibrary*), a proto je nutné použít v této části takzvaný nespravovaný kód, při jehož používání je programátor zodpovědný za správu paměti sám oproti spravovanému kódu, kde je kontrola zajištěna. Technologie .NET Core respektive C# tuto správu paměti automaticky provádí a používá tedy spravovaný kód. Dovoluje ovšem i použití nespravovaného kódu, a tak je možné použít standardní funkce pro práci s knihovnami. Po ukončení práce s knihovnou je nutné veškerou alokovanou paměť uvolnit (*FreeLibrary*). Funkcí *LoadLibrary* provede API načtení knihovny a následně dokáže provést volání funkce `ExecuteTask`, které však musí zadat vstupní parametry. Vstupní XML řetězec je vygenerován na základě informací v požadavku z webu, ze kterého je získán název úlohy, název databáze, zvolené období a z přiloženého tokenu jsou získány informace o uživateli, a má strukturu uvedenou na obrázku 26. Společně s ním je alokována paměť pro výstupní XML. Na základě vstupního XML řetězce dochází k volání funkce `ExecuteTask` a provedení funkcionality knihovny *P3k_plankap*, popsané v kapitole 5. Výstupem knihovny je XML řetězec se strukturou uvedenou na obrázku 28. Ten je zkopírován do alokované paměti API pro výstup. Knihovna je po zpracování uvolněna z paměti společně se všemi daty, které ke svému chodu potřebovala, aby byla zajištěna správná manipulace s pamětí. [35]

Následně z načtených dat vytváří API odpověď, která je nachystána k odeslání směrem k uživatelskému rozhraní ve formátu JSON. Jelikož je v servisní třídě webové aplikace manipulace s daty definována pomocí *RxJS Observables*, je třída přihlášena k odpovědi na HTTP požadavek (subscribe), aby získala data, jakmile jsou k dispozici.

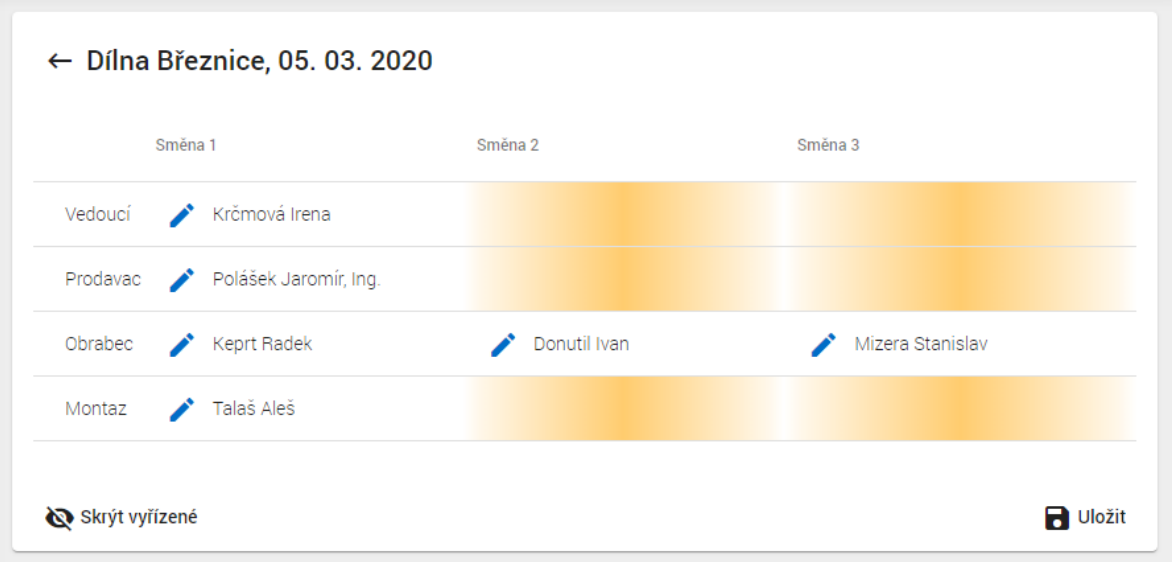
Tímto způsobem je docíleno načtení požadovaných dat. Ty jsou následně rozděleny do příslušných proměnných a na základě čísla zaměstnance a čísla pracovněprávního vztahu doplněny o paralelně načtená data modulem pro prohlížení seznamu pracovníků. Takto jsou data připravena k prezentaci.

Obsah je na HTML stránce prezentován za použití komponenty Angular Material Designu `mat-table`, což je tabulka uzpůsobená k zobrazení seznamu dat dle zásad Material Designu. V tomto případě se jedná o tabulku obsahující seznam jednotek a jejich plánů pro celý měsíc. Výsledkem je prezentace seznamu jednotek, ke kterým má uživatel přístup, a stav jejich plánu pro dny v měsíci. Pokud se vyskytuje některý den s pravdivou hodnotou příznaku `prazdnabunka`, je v tabulce zobrazen červený varovný symbol. Znamená to, že tento den není správně naplánován a nejsou obsazeny všechny pracovní pozice. V opačném případě je zobrazena značka odškrtnutí, která symbolizuje správné naplánování a zaplnění všech pozic zaměstnanci. Na obrázku 43 je možné vidět takto vytvořený přehled plánů jednotek pro vybrané období.



Obrázek 43 – Přehled plánů jednotek pro vybrané období v aplikaci PermWeb

Kliknutím na buňky tabulky, které obsahují symboly, je možné zobrazit naplánování přiřazení zaměstnanců pro daný den dané jednotky. Tímto gestem dojde k překreslení obsahu stránky a zobrazení nové tabulky `mat-table` s výčtem pracovních pozic, směn a přiřazenými zaměstnanci (Obrázek 44). Pokud má uživatel oprávnění, zobrazují se v buňkách také jména přiřazených zaměstnanců na danou pozici a směnu. Tato tabulka slouží zároveň i pro úpravy přiřazování zaměstnanců. Nachází se zde tlačítko pro uložení změn a tlačítko pro zobrazení pouze těch pozic, které nejsou správně naplánovány. Po kliknutí na tlačítko Skrýt vyřízené dochází k zobrazení pouze nesprávně naplánovaných pozic. Pro zjištění, o které pozice se jedná, je využito příznaku `prazdnabunka`, který udává informaci, zda je na pozici správně naplánováno přiřazení zaměstnance. V případě pravdivosti je tato pozice zobrazena.

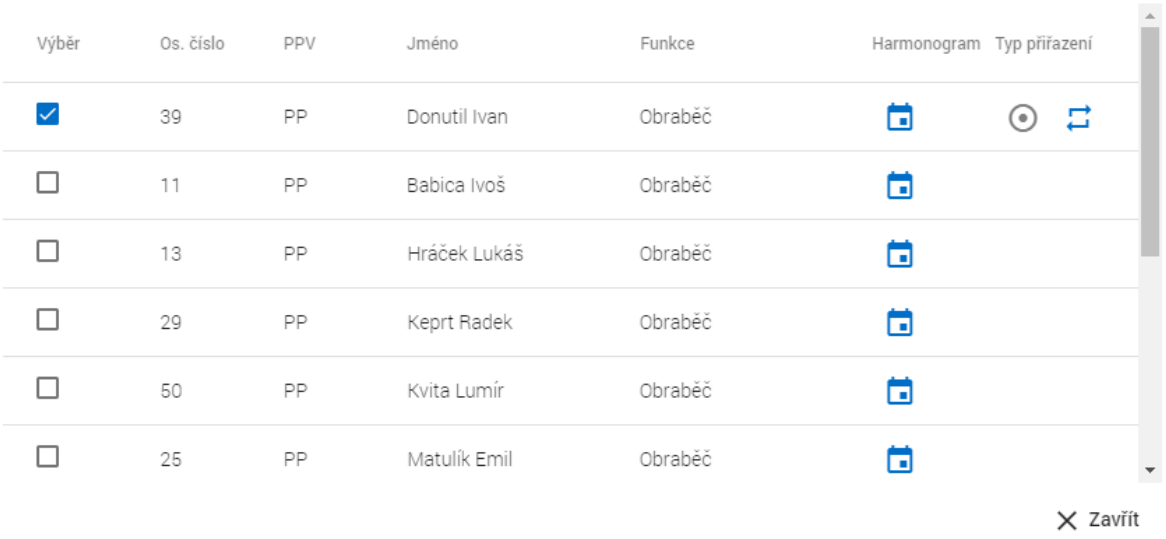


Obrázek 44 – Plán vybrané jednotky pro konkrétní den

6.2 Přiřazování zaměstnanců a zápis dat

Důležitou částí pro tvorbu plánu je přiřazování zaměstnanců na pracovní pozice, případně úpravy těchto přiřazení. Na jejich základě může docházet k tvorbě implicitních plánů a rozmístění zaměstnanců na směny dle jejich rozvrhu, které zajišťuje funkce knihovny. Také je nutné umožnit explicitní přiřazení na konkrétní den a směnu, aby bylo možné vytvářet výjimky v plánu.

Další částí při popisu obrázku 44 je možnost editace přiřazení, která je zajištěna tlačítkem se symbolem tužky. Toto tlačítko je součástí buněk tabulky, které definují vazbu pracovní pozice a směny. V buňkách jsou zároveň prezentovány jména, případně čísla přiřazených zaměstnanců na danou pozici a směnu. Kliknutím na tlačítko je vyvolán dialog. Ten je zajištěn komponentou `MatDialog`, která je uzpůsobená požadavkům Material Designu. V dialogu je pomocí komponenty `mat-table` zobrazen výčet zaměstnanců, které je možné na pozici dosadit. Jedná se o zaměstnance, kteří mají kód funkce shodný s kódem funkce buňky. Na obrázku 45 lze vidět detail dialogu.



Obrázek 45 – Dialog výběru zaměstnanců pro přiřazení na vybranou pozici

V prvním sloupci se nachází tlačítko pro výběr či odebrání přiřazení daného zaměstnance. Přiřazení je zajištěno aktivací zatržítka, jeho odebrání deaktivací zatržítka. V posledním sloupci se naopak nachází dvě tlačítka, které jsou přepínači typu přiřazení, a určují, jestli se jedná o jednorázové nebo trvalé přiřazení. Prvním typem je *Ruční přiřazení*, které zajišťuje jednorázové přiřazení zaměstnance pouze na danou směnu daný den. Tento typ přiřazení je vhodný pro plánování náhrady pro daný den. Druhým typem je *Generováno harmonogramem pracovníka*, který slouží pro trvalé přiřazení na pracovní pozici, kdy každé další přiřazení bude plánováno na základě rozvrhu směn zaměstnance (Harmonogram). Toto přiřazení je určeno pro zajištění dlouhodobého plánu. Vzniklé výjimky v dlouhodobém plánu lze poté vyřešit jednorázovým přiřazením. Tlačítko v předposledním sloupci slouží pro vytvoření požadavku na zobrazení harmonogramu směn daného zaměstnance.

Provedené změny ve výběru zaměstnanců a typu přiřazení je možné po zavření dialogu uložit. Klinutím na tlačítko Uložit dochází k volání funkce servisní třídy pro vytvoření HTTP požadavku na API pro zpracování nových informací. Při tomto procesu dochází k obdobným postupům jako při načítání s tím rozdílem, že API nyní zpracovává požadavek typu PUT (na rozdíl od GET při čtení). Na základě přiloženého tokenu je provedena autentizace a vytvoření hlavičky vstupního XML řetězce. Tělo je vytvořeno na základě dat požadavku a má strukturu uvedenou na obrázku 27. Dochází k načtení knihovny *P3k_plankap* a volání její hlavní funkce s parametrem vytvořeného vstupního XML řetězce. Data jsou zpracována knihovnou, uložena do databáze a zpětně načtena pro zobrazení aktuálního stavu. Knihovna je uvolněna z paměti a data jsou k dispozici k odběru. Servisní třídy webové aplikace jsou vždy

přihlášeny o odběr dat a čekají na odpověď svého HTTP požadavku. Po získání odpovědi je možné data vykreslit jako aktuální obsah do zmiňovaných tabulek `mat-table`.

Chod aplikace lze popsat jako neustálý koloběh, kdy dochází k tvorbě požadavků a načítání aktuálně požadovaných modulů, komponent a knihoven. Následně je prováděna funkcionalita knihoven nad zvolenou databází a zisk dat pro prezentaci. Data je možné prezentovat, případně je v rámci prezentace i upravovat. Na základě úprav poté dochází k tvorbě nových požadavků a opakování koloběhu zpracování.

V průběhu tvorby řešení se však zadání ukázalo jako daleko rozsáhlejší a časové náročnější. Zpracování tohoto úkolu se z časových důvodů projevilo jako nesplnitelné vznikem situace spojené s bezpečnostními opatřeními, kdy bylo daleko obtížnější spolupracovat s členy firmy, a větší komplexností řešeného problému. Bylo tedy nutné tuto situaci řešit, aby byl bod zadání včas zpracován. Z toho důvodu nebyla část pro webové API a grafické webové rozhraní vytvořena individuálně, ale zpracována firmou, kdy jsem se podílel a spolupracoval na jejím vytvoření. Webová část je díky tomu zhotovena ve funkční podobě v návaznosti na vytvořenou DLL knihovnu, a řešení je tak kompletně zapracováno do webové aplikace PermWeb.

7 MOŽNOSTI BUDOUCÍHO VÝVOJE

V oblasti informačních technologií dochází k neustálému vývoji. Proto je nutné na vývoj reagovat a snažit se neustále vylepšovat vytvořené aplikace. Pokaždé se nabízí možnost rozšířit funkcionalitu aplikace nebo vytvořit kvalitnější výsledek a reagovat na požadavky uživatelů. Tím dochází ke zdokonalování řešení a poskytování lepších služeb. Moje řešení zadané práce má několik možností vývoje, které mohou být v budoucnu zpracovány. Se svým konzultantem a vedoucím práce jsme si je částečně nastínili. Z toho důvodu jsem se pokusil nachystat v rámci mého řešení prostředky pro možnost jejich budoucí realizace.

7.1 Více variant provozu

Varianta je jednopísmenná vlastnost uvedená u definice pracovní pozice. V současném řešení je pro definici pracovních pozic a měsíčního plánu pracovní jednotky použita základní varianta „A“. Při tvorbě plánu jednotky jsou použity jen ty pracovní pozice, které nesou toto písmeno varianty. Této jednopísmenné vlastnosti by se však dalo využít k definici dalších variant, jako jsou například víkendové či sváteční provozy („V“, „S“). Na základě kalendáře a varianty by mohlo docházet k tvorbě různých definic jednotky pro daný den s různými pracovními místy a směnami.

7.2 Tvorba pravidelného rozvrhu s využitím masek

Maska slouží pro tvorbu rozvrhu směn. Podle ní dochází k rozvržení podoby směn celého týdne. V současném řešení nejsou masky jednotky využity. Jsou ovšem nachystány v databázových tabulkách i aplikačních objektech pro definici plánovací jednotky. Pomocí nich by bylo možné vytvořit rozvrh různých variant plánovací jednotky, obdobně jako jsou vytvářeny rozvrhy směn. Docházelo by tak k cyklickému opakování rozvrhu variant plánovací jednotky na základě vytvořené masky a kalendáře.

7.3 Propojení s přesčasy a filtrace na základě budoucí dostupnosti

Při plánování zaměstnanců na směny by bylo vhodné provádět kontrolu, zda nedochází k překračování možného počtu hodin práce. Při kontrole by se mohly zohledňovat počty hodin přesčasů, aby nedocházelo k jejich překročení a takového zaměstnance nedovolit přiřadit na plánovanou pozici. Pokrokem by bylo provádět kontrolu dostupnosti zaměstnanců na vybranou pozici a směnu ještě před samotným výpisem seznamu nabízených zaměstnanců. Bylo by tak možné vybírat pouze z těch, kteří prošli ověřením.

7.4 Trvalé přiřazování zaměstnanců

V současném řešení je trvalé přiřazování zaměstnanců na pozici řešeno společně s jednorázovým. Do budoucna by bylo vhodné toto přiřazování oddělit do samostatného dialogu, případně výchozí přiřazování zaměstnanců přesunout do aplikace Perm3. Díky tomu by nebylo možné tak jednoduše zasahovat do trvalých přiřazení. Plánování by tak bylo zajištěno přiřazením mimo zobrazované naplánování jednotek a primárně by docházelo jen k zadávání náhradních přiřazení a výjimek.

ZÁVĚR

Cílem této práce bylo vytvořit systém pro plánování zaměstnanců na pracovní pozice uvnitř pracovní jednotky a zapracovat jej do aplikací Perm3 a PermWeb. V rámci aplikace Perm3 byly navrženy a realizovány databázové tabulky a jejich obsluha aplikačními objekty, vytvořena aplikační okna pro tvorbu plánovacích jednotek a definic pracovních pozic. Dále byla sestavena DLL knihovna, která slouží pro zpracování přiřazení zaměstnanců na pracovní pozice a definice jednotek, na základě kterých je vytvářen plán jednotky. V návaznosti na funkcionalitu této knihovny bylo vytvořeno uživatelské rozhraní ve webové aplikaci PermWeb pro snadnou práci s plány jednotek. Také byly popsány možnosti budoucího vývoje, kterým by se dal systém rozšířit a zdokonalit.

V teoretické části byla nastíněna problematika plánování, a to jak z hlediska manažerské činnosti, tak plánování kapacit a lidských zdrojů. Důležitým poznatkem je, že se zdroji je třeba správně pracovat a vhodně plánovat jejich využití. Jedná se o velmi důležitou činnost, která vede k úspěchu a dosahování vytyčených cílů. Dále byly popsány technologie, které jsou použity nebo souvisí s řešením tvorby plánovacího systému, jejich možnosti a vlastnosti, pro které jsou použity.

Praktická část byla věnována tvorbě systému plánování kapacit, jako rozšíření aplikace Perm3 a webové aplikace PermWeb, pomocí technologií popsaných v teoretické části. Nejprve byl popsán návrh databázových tabulek společně s definicí, co to plánovací jednotka je a jakou má strukturu. Struktura plánovací jednotky byla navržena tak, aby sdružovala pracovní pozice, a na tyto pozice mohli být přiřazováni zaměstnanci. Jedná se základní stavební kámen systému. Princip vytváření jednotek spočívá v definici obecné jednotky, od které je možné odvozovat konkrétní jednotky s převzetím definice pracovních pozic. Díky tomu je možné provádět jednoduše změny definic. Datové struktury jsou spolu provázány, a každá tabulka tak odpovídá příslušným vlastnostem plánovací jednotky či přiřazení zaměstnance. Dále zde byly popsány aplikační objekty, které byly vytvořeny standardním způsobem pro aplikaci Perm3, v návaznosti na databázové tabulky, a zajišťují manipulaci s daty skrze celou aplikací. Jsou proto využity k prezentaci a zpracování úprav dat uvnitř aplikačních oken. Tvorba aplikačních oken je popsána v kapitole 4. Jedná se o grafické rozhraní pro vytváření plánovacích jednotek. Svou podobou a funkcionalitou zajišťují zapracování navrhnutých struktur a objektů do chodu aplikace Perm3.

Značná část je věnována detailnímu popisu DLL knihovny. Ta vznikla za účelem sestavování plánů jednotek. Pro svou práci používá data definic plánovacích jednotek a přiřazení zaměstnanců na pracovní pozice. Také využívá stávající rozvrhy směn zaměstnanců a jejich docházku, čímž je systém opět provázán s aplikací Perm3, aby docházelo ke správnému plánování. Dokáže zpracovávat vstupní změny a jako výstup poskytuje vypracované plány. Knihovna obsahuje velkou část know-how, které jsem systému dal, a je z hlediska plánování nejdůležitější částí této práce.

Následně byla popsána část rozhraní webové aplikace PermWeb, pro kterou byl vytvořen nový modul pro kapacitní plánování. Ten pro svou práci využívá funkcionalitu vytvořené knihovny. Rozhraní vzniklo za účelem přiřazování zaměstnanců na pracovní pozice plánovacích jednotek, zobrazení plánů a tvorbě změn přiřazení. Slouží jako manažerský nástroj pro plánování a snadné získání přehledu o dostupnosti zaměstnanců. Jedná se o poslední krok pro zapracování systému plánování do stávajícího řešení. V průběhu tvorby řešení se však zadání ukázalo jako daleko rozsáhlejší a časové náročnější. Zpracování tohoto úkolu se z časových důvodů projevilo jako nesplnitelné vznikem situace spojené s bezpečnostními opatřeními, kdy bylo daleko obtížnější spolupracovat s členy firmy, a větší komplexností řešeného problému a rozsáhlosti webové aplikace. Aby byl bod zadání včas zpracován, bylo nutné tuto situaci řešit jiným způsobem. Z toho důvodu nebyla část pro webové API a grafické webové rozhraní vytvořena individuálně, ale zpracována firmou, kdy jsem se podílel a spolupracoval na jejím vytvoření. Webová část je díky tomu zhotovena ve funkční podobě v návaznosti na vytvořenou DLL knihovnu, a řešení je tak kompletně zapracováno do webové aplikace PermWeb.

Poslední kapitola popisuje možnosti budoucího vývoje, kterými by bylo možné systém rozšířit. Jedná se o návrhy, které vznikly v průběhu tvorby systému. Také z toho důvodu jsou pro některé z nich již v systému nachystány prostředky pro možnou budoucí realizaci.

Systém umožňuje vytvářet v aplikaci Perm3 plánovací jednotky, definovat jejich pracovní pozice a tyto definice využívat pro tvorbu dalších jednotek. Pomocí rozhraní webové aplikace PermWeb je možné zobrazit strukturu vytvořené jednotky a přiřazovat zaměstnance na pracovní pozice, na základě čehož dochází k vytváření plánů jednotek. Plány je možné upravovat změnami přiřazení zaměstnanců na pracovní pozice. Výpočet plánů zajišťuje výkonná knihovna, která s využitím definic jednotek a přiřazení zaměstnanců zhotoví plány dle rozvrhů a dostupnosti zaměstnanců.

Závěrem lze říct, že systém jako celek funguje spolehlivě, má rychlou odezvu a poskytuje požadovanou funkcionalitu. Při načítání dat ve webovém klientu dochází k mírné prodlevě z důvodu spuštění funkce knihovny, která pro vykonání své činnosti načítá a zpracovává větší množství dat. Během tvorby systému nedocházelo k větším problémům, kromě řešení již zmiňované webové části. Podařilo se realizovat všechny části systému a reagovat na nově vzniklé požadavky. Systém byl vytvořen s využitím prostředků společnosti Kvasar a zapracován jako modul do stávajícího řešení, čímž bylo dosaženo požadovaného výsledku.

Vytvořený systém je plně funkční. Vzhledem k návrhům, které byly nastíněny v poslední kapitole, bych se následně chtěl věnovat jejich případné realizaci.

SEZNAM POUŽITÉ LITERATURY

[1] KRÁL, Bohumil. *Manažerské účetnictví*. 3., dopl. a aktualiz. vyd. Praha: Management Press, 2010. ISBN 978-80-7261-217-8.

[2] GONDÍKOVÁ, Bc. Vendula. *Kapacitní plánování výrobních dělníků* [online]. Plzeň, 2017 [cit. 2020-01-13]. Dostupné z: http://www.kiv.zcu.cz/~novyp/dip/dp_gondikova.pdf. Diplomová práce. Západočeská univerzita v Plzni.

[3] Plánování. *Kiwi.mendelu.cz: Uživatelský server Kiwi.mendelu.cz* [online]. Brno, 2014 [cit. 2020-01-13]. Dostupné z: http://user.mendelu.cz/xbadal/Studijni%20opory/Zaklady%20ekonomiky%20a%20podnikani%20(arbor.)/PL%C1NOV%C1N%CD.pdf

[4] ROMAN. Plánování. *Univerzita-Online.cz* [online]. 2012, 24. 2. 2012 [cit. 2020-01-13]. Dostupné z: http://www.univerzita-online.cz/mng/zaklady-managementu/planovani/

[5] SYNEK, Miloslav a Eva KISLINGEROVÁ. *Podniková ekonomika*. 5., přeprac. a dopl. vyd. Praha: C.H. Beck, 2010. Beckovy ekonomické učebnice. ISBN 978-80-7400-336-3.

[6] Managementmania: Prognózování (Forecasting). *Managementmania* [online]. 2015, 23. 10. 2015 [cit. 2020-01-13]. Dostupné z: https://managementmania.com/cs/prognozovani

[7] BUČKOVÁ, Šárka. *Rozpočet a systém plánů a rozpočtů – pojetí, tvorba a návrh, využití, zhodnocení* [online]. Znojmo, 2011 [cit. 2020-01-13]. Dostupné z: https://theses.cz/id/r84p3h. Bakalářská práce. SOUKROMÁ VYSOKÁ ŠKOLA EKONOMICKÁ ZNOJMO s.r.o.

[8] Management - Plánování. *Miraslebl* [online]. 2019, 24. 2. 2012 [cit. 2020-01-13]. Dostupné z: http://www.miras.cz/seminarky/management-planovani.php

[9] Personální plánování a strategie lidských zdrojů. *Managementmania* [online]. 2016 [cit. 2020-01-13]. Dostupné z: https://managementmania.com/cs/personalni-planovani-strategie-lz

[10] Proces plánování pracovníků. *Altaxo: Komplexní služby pro podnikatele* [online]. 2019 [cit. 2020-01-13]. Dostupné z: https://www.altaxo.cz/provoz-firmy/personalistika/rizeni-lidskych-zdroju/proces-planovani-pracovniku

[11] GAJIC, Zarko. Delphi History – from Pascal to Embarcadero Delphi XE 2: Delphi history: the Roots. *ThoughtCo: Lifelong learning* [online]. 17. 3. 2017 [cit. 2020-01-13]. Dostupné z: https://www.thoughtco.com/history-of-delphi-1056847

[12] RAD Studio Overview. Embarcadero [online]. 2018 [cit. 2020-01-13]. Dostupné z: https://www.embarcadero.com/products/rad-studio

[13] Proč je Embarcadero Delphi a ne Borland? *Delphi.cz: Komunitní portál Delphi* [online]. 2018 [cit. 2020-01-13]. Dostupné z: https://delphi.cz/page/Proc-je-Embarcadero-Delphi-a-ne-Borland.aspx

[14] KAMBURELIS, Michalis. Modern Object Pascal Introduction for Programmers. *Castle Game Engine* [online]. [cit. 2020-01-13]. Dostupné z: https://castle-engine.io/modern_pascal_introduction.html

[15] Form Designer. *Embarcadero: Product documentation Wikis* [online]. 2019 [cit. 2020-01-13]. Dostupné z: http://docwiki.embarcadero.com/RADStudio/Rio/en/Form_Designer

[16] SVOBODA, Josef, KOBELKA, Michal, ed. Object Pascal - 03: Úvod k OOP, datové typy 2. část. *Programujte.com* [online]. 2020, 24. 10. 2008 [cit. 2020-01-13]. Dostupné z: http://programujte.com/clanek/2008090100-object-pascal-03-uvod-k-oop-datove-typy-2-cast/

[17] ČERVINKA, Radek. Delphi Object Pascal pro začátečníky. *Delphi.cz: Komunitní portál Delphi* [online]. 2018, 18. 5. 2010 [cit. 2020-01-13]. Dostupné z: https://delphi.cz/post/Object-Pascal-zacatecnici.aspx

[18] Beginner's guide - Object Oriented Programming. *Dev: Dev community* [online]. 4. 8. 2018 [cit. 2020-01-13]. Dostupné z: https://dev.to/charanrajgolla/beginners-guide---object-oriented-programming

[19] ČÁPKA, David. Lekce 7 - Dědičnost a polymorfismus. *ITnetwork.cz* [online]. [cit. 2020-01-13]. Dostupné z: https://www.itnetwork.cz/csharp/oop/c-sharp-tutorial-dedicnost-a-polymorfismus

[20] RAD Studio EX2: The ultimate application development suite for Windows, Mac, mobile and Web. *Embarcadero* [online]. [cit. 2020-01-13]. Dostupné z: https://edn.embarcadero.com/article/41595

[21] Dynamic-Link Libraries. *Microsoft Docs* [online]. 31. 5. 2018 [cit. 2020-01-14]. Dostupné z: https://docs.microsoft.com/en-us/windows/win32/dlls/dynamic-link-libraries

[22] About Dynamic-Link Libraries. *Microsoft Docs* [online]. 31. 5. 2018 [cit. 2020-01-14]. Dostupné z: https://docs.microsoft.com/en-us/windows/win32/dlls/about-dynamic-link-libraries

[23] Advantages of Dynamic Linking. *Microsoft Docs* [online]. 31. 5. 2018 [cit. 2020-01-14]. Dostupné z: https://docs.microsoft.com/en-us/windows/win32/dlls/advantages-of-dynamic-linking

[24] Introduction to XML. *W3schools.com* [online]. 2020 [cit. 2020-01-15]. Dostupné z: https://www.w3schools.com/xml/xml_whatis.asp

[25] ROUSE, Margaret. Microsoft SQL Server. *Techtarget* [online]. Červen 2019 [cit. 2020-01-15]. Dostupné z: https://searchsqlserver.techtarget.com/definition/SQL-Server

[26] Programovací jazyky. *České vysoké učení technické v Praze: Fakulta dopravní* [online]. 1. 3. 2017 [cit. 2020-07-07]. Dostupné z: https://www.fd.cvut.cz/personal/jerabem1/14prg/14PRG_03_programovaci_jazyky.pdf

[27] PROKOPOVÁ, Zdenka. *Databázové systémy MySQL+PHP*. Ve Zlíně: Univerzita Tomáše Bati, 2006. ISBN 80-7318-486-9.

[28] SQL Server CURSOR. *SQL Server Tutorial* [online]. 2020 [cit. 2020-07-07]. Dostupné z: https://www.sqlservertutorial.net/sql-server-stored-procedures/sql-server-cursor/

[29] FIELDING, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures*. UNIVERSITY OF CALIFORNIA, IRVINE, 2000. Disertační práce. UNIVERSITY OF CALIFORNIA, IRVINE.

[30] *REST API Tutorial* [online]. [cit. 2020-01-17]. Dostupné z: https://restfulapi.net/

[31] MALÝ, Martin. REST: architektura pro webové API. *Zdroják.cz* [online]. 3. 8. 2009 [cit. 2020-01-17]. Dostupné z: https://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/

[32] DEERING, Sam. Do you know what a REST API is? *Sitepoint* [online]. 7. 12. 2012 [cit. 2020-01-17]. Dostupné z: https://www.sitepoint.com/developers-rest-api/

[33] HTTP. *MDN Web Docs: Resources for developers, by developers.* [online]. 2020, 18. 5. 2020 [cit. 2020-07-07]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP

[34] GOMES, Evandro. An awesome guide on how to build RESTful APIs with ASP.NET Core. *FreeCodeCamp* [online]. 21. 2 .2019 [cit. 2020-01-17]. Dostupné z: https://www.freecodecamp.org/news/an-awesome-guide-on-how-to-build-restful-apis-with-asp-net-core-87b818123e28/

[35] Co je spravovaný kód? *Microsoft Docs* [online]. 20. 6. 2016 [cit. 2020-05-11]. Dostupné z: https://docs.microsoft.com/cs-cz/dotnet/standard/managed-code

[36] Web Blog: AngularJS : Create a Web Site using AngularJS, ASP.NET MVC, Web API - PART 1: Introduction. *Web Blog* [online]. 2016, 31. 1. 2016 [cit. 2020-07-02]. Dostupné z: http://oniwebblog.blogspot.com/search/label/Angularjs

[37] Web Blog: General : CRUD application using AngularJs, NodeJs, MongoDB - Part 1 : introduction. *Web Blog* [online]. 2016, 31. 1. 2016 [cit. 2020-07-02]. Dostupné z: http://oniwebblog.blogspot.com/2016/01/general-crud-application-using.html

[38] Architecture overview. *Angular* [online]. 2019 [cit. 2020-02-03]. Dostupné z: https://angular.io/guide/architecture

[39] NWOSE, Lotanna. LAZY LOADING IN ANGULAR 7. *Pusher: Powering real-time experiences for mobile and web* [online]. 16. 8. 2019 [cit. 2020-04-07]. Dostupné z: https://pusher.com/tutorials/lazy-loading-angular-7

[40] TypeScript overview. *TutorialsPoint* [online]. 2020 [cit. 2020-02-03]. Dostupné z: https://www.tutorialspoint.com/typescript/typescript_overview.htm

[41] TypeScript. *TypeScript* [online]. 2020 [cit. 2020-02-03]. Dostupné z: https://www.typescriptlang.org/

[42] Angular Material Tutorial: Overview. *TutorialsPoint: SIMPLYEASYLEARNING* [online]. [cit. 2020-04-07]. Dostupné z: https://www.tutorialspoint.com/angular_material/angular_material_overview.htm

[43] *Angular Material: Material Design components for Angular* [online]. 2020 [cit. 2020-04-07]. Dostupné z: https://material.angular.io/

[44] The RxJS library. *Angular* [online]. 2019 [cit. 2020-02-03]. Dostupné z: https://angular.io/guide/rx-library

[45] Introduction. *RxJS* [online]. [cit. 2020-02-03]. Dostupné z: https://rxjs.dev/guide/overview

[46] What is RxJS. *Thinkster: A better way to learn* [online]. [cit. 2020-02-03]. Dostupné z: https://thinkster.io/tutorials/learn-rxjs-observables/what-is-rxjs

[47] Personalistika a mzdy - Perm3. *Kvasar* [online]. [cit. 2020-02-03]. Dostupné z: http://www.kvasar.cz/24729-personalistika-mzdy-perm3

[48] PermWeb - webová nadstavba systému PERM. *Kvasar* [online]. [cit. 2020-02-03]. Dostupné z: http://www.kvasar.cz/24766-permweb

[49] Access Tokens. *Auth0: Identity is Complex. Deal with it.* [online]. 2020, 2018 [cit. 2020-04-03]. Dostupné z: https://auth0.com/docs/tokens/concepts/access-tokens#access-token-lifetime

[50] Understanding Refresh Tokens: Learn about refresh tokens and the role they serve in the authorization process. *Auth0: Identity is Complex. Deal with it.* [online]. 2020, 2016 [cit. 2020-04-03]. Dostupné z: https://auth0.com/learn/refresh-tokens/

[51] *DB DESIGNER: Online Database Schema Design and Modeling Tool* [online]. 2020 [cit. 2020-04-07]. Dostupné z: https://www.dbdesigner.net/

[52] Circle Design. *Kissclipart: Free clipart & transparent image resources for everyone.* [online]. [cit. 2020-04-10]. Dostupné z: https://www.kissclipart.com/material-design-database-icon-clipart-database-com-pt6nhb/

[53] KOUŘIL, Jiří. *Implementace modulu plánování zdrojů pro systém Perm3*. Zlín: Univerzita Tomáše Bati ve Zlíně, 2017, 61 s. (77 230 znaků). Dostupné také z: http://hdl.handle.net/10563/43504. Univerzita Tomáše Bati ve Zlíně. Fakulta aplikované informatiky, Ústav automatizace a řídicí techniky. Vedoucí práce Dulík, Tomáš.

[54] LISCHNER, Ray. Delphi in a Nutshell by Ray Lischner. *O'Reilly: Get certified. Get ahead.* [online]. 2020 [cit. 2020-04-06]. Dostupné z: https://www.oreilly.com/library/view/delphi-in-a/1565926595/re306.html

[55] Icons. *Materal Design: Build beautiful products, faster.* [online]. [cit. 2020-04-16]. Dostupné z: https://material.io/resources/icons/?style=baseline

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

RAD	Rapid Application Development
pas	Přípona souborů Pascal
dfm	Přípona souborů obsahující vlastnosti formulářů vytvářených v Borland Delphi
DLL	Dynamic-link library
XML	Extensible Markup Language
HMTL	Hypertext Markup Language
MSSQL	Microsoft SQL Server
SQL	Structured Query Language
CRUD	Create, Read, Update, Delete
REST	Representational State Transfer
HTTP	Hypertext Transfer Protocol
API	Application Programming Interface
URI	Uniform Resource Identifier
JSON	JavaScript Object Notation
URL	Uniform Resource Locator
MVC	Model View Controller
RxJS	Reactive Extensions for JavaScript
spol s r. o.	Společnost s ručením omezeným
MS	Microsoft
N:N	N ku N, vazba několik k několika
+	Symbol pro vytvoření nového záznamu
^	Symbol pro úpravu vybraného záznamu
−	Symbol pro odstranění vybraného záznamu
tag	Značka
id	Identifikátor

SEZNAM OBRÁZKŮ

SEZNAM PŘÍLOH

PŘÍLOHA P I: OBSAH CD

Struktura obsahu přiloženého CD:

- Adresář **Text diplomové práce** – obsahuje text diplomové práce ve formátu PDF
- Adresář **Zdrojové kódy a soubory** – obsahuje vytvořené soubory pro systém kapacitního plánování se zdrojovými kódy. Také se zde nachází textový soubor s SQL dotazy pro vytvoření tabulek v databázi a vytvořená výkonná knihovna P3k_plankap.dll. Dále je zde umístěna ikona nové položky menu aplikace Perm3 pro Plánovací jednotky. Ve složkách Perm3 a PermWeb jsou umístěny příslušné soubory pro dané aplikace.

  Seznam souborů, na které je v textu diplomové práce odkazováno:

  [1] P3ObjPlanJed.pas

  [2] P3ObjPlanKap.pas

  [3] P3Q_PlanJed.pas

  [4] P3E_PlanJed.pas

  [5] P3V_PlanJedBunky.pas

  [6] P3E_PlanJedBunky .pas

  [7] P3E_PlanKap.pas

  [8] P3V_VybDefJed.pas

  [9] P3k_plankap.dll

  [10] k_plankap_in.xml

  [11] k_plankap_out.xml

  [12] P3Dplankap.pas