SW modul pro fakturaci

Bc. Martin Ofúkaný

Diplomová práce
2022

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2021/2022

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Martin Ofúkaný**
Osobní číslo:	**A20687**
Studijní program:	**N0613A140022 Informační technologie**
Specializace:	**Softwarové inženýrství**
Forma studia:	**Kombinovaná**
Téma práce:	**SW modul pro fakturaci**
Téma práce anglicky:	**SW Module for Invoicing**

Zásady pro vypracování

1. Vypracujte literární rešerši na dané téma.
2. Popište zdůvodnění potřebnosti navrhovaného řešení.
3. Navrhněte a vytvořte databázi webové aplikace.
4. Vytvořte strukturu a vzhled aplikace pomocí zvolených nástrojů.
5. Implementujte funkcionalitu na straně serveru a s využitím zvoleného jazyka také vytvořte dynamický obsah aplikace.
6. Věnujte pozornost zabezpečení aplikace.
7. Popište funkcionalitu vytvořené aplikace.
8. Proveďte závěr.

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

Seznam doporučené literatury:

1. ŽÁRA, Ondřej. *JavaScript: programátorské techniky a webové technologie*. Brno: Computer Press, 2015, 180 s. ISBN 9788025145739. Dostupné také z: http://knihy.cpress.cz/K2209
2. SUEHRING, Steve. *JavaScript: krok za krokem*. Brno: Computer Press, 2008, 335 s. Krok za krokem. ISBN 9788025122419.
3. MOLHANEC, Martin. *Webové metodiky*. Praha: Alfa Nakladatelství, 2014, 210 s. Informatika. Monografie. ISBN 9788087197844.
4. HOWARD, Michael a David LEBLANC. *Bezpečný kód: [techniky a strategie tvorby bezpečných webových aplikací]*. Brno: Computer Press, 2008, 895 s. ISBN 9788025120507.
5. FREEMAN, Adam. *Pro ASP.NET Core 3: Develop Cloud-Ready Web Applications Using MVC, Blazor, and Razor Pages*. 8th ed. 2020. Berkeley, CA: APress, 2020 ;, 1 online zdroj (XXIX, 1080 stran). ISBN 9781484254400.

Vedoucí diplomové práce: **doc. Ing. Roman Šenkeřík, Ph.D.**
Ústav informatiky a umělé inteligence

Konzultant diplomové práce: **Ing. Tomáš Vogeltanz, Ph.D.**
Ústav počítačových a komunikačních systémů

Datum zadání diplomové práce: **3. prosince 2021**
Termín odevzdání diplomové práce: **23. května 2022**

doc. Mgr. Milan Adámek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 24. ledna 2022

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 práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- 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 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 23. 5. 2022

Martin Ofúkaný, v. r.
podpis studenta

ABSTRAKT

Zameraním diplomovej práce je vytvorenie softvérového modulu, ktorého úlohou je zjednodušiť fakturačné procesy. Jeho hlavnou úlohou je vytváranie dokladov v obchodnom prípade napríklad cenových ponúk alebo dodacích listov. Softvér je webová aplikácia, vytvorená na platforme .NET pomocou technológie ASP.NET s využitou architektúrou klient – server. Práca ďalej popisuje vybrané sekcie kódu, použité pri zhotovení tejto webovej aplikácie.

Klíčová slova: užívateľské rozhranie, reťazec, požiadavka, funkcia, kód, aplikácia

ABSTRACT

The focus of the thesis is to create a software module whose task is to simplify invoicing processes. It's main task is to create documents in a business case, such as quotations or delivery notes. The software is a web application created on the .NET platform using ASP.NET technology using client-server architecture. The work further describes a selected sections of code used in creating this web application.

Keywords: user interface, string, request, function, code, application

Rád by som poďakoval mojím školiteľom doc. Ing. Romanovi Šenkeříkovi, Ph.D. a Ing. Tomášovi Vogeltanzovi, Ph.D. za cenné rady a odbornú pomoc, ktoré mi poskytli pri písaní diplomovej práce. Ďalej by som chcel poďakovať svojej rodine a priateľom za podporu a trpezlivosť.

Motto:

Marijn Haverbeke

Programovanie je ťažké. Základné pravidlá, na ktorých je všetko postavené, sú veľmi jednoduché, no ako sa program vyvíja, sám začína zavádzať svoje pravidlá a zákony. Programátor tak buduje labyrint, v ktorom sa môže stratiť aj on sám.

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

OBSAH

ÚVOD

V dnešnom, modernom svete rastie záujem o webové aplikácie každým dňom. Pre užívateľov, ale aj pre vývojárov, majú webové aplikácie množstvo výhod voči v minulosti najvyužívanejším desktopovým aplikáciám. Webové aplikácie fungujú v prostredí webových prehliadačov a vďaka tomu fungujú bez inštalácie akýchkoľvek doplnkov. Na miesto ukladania dát v užívateľskom zariadení, akým môže byť napr. počítač, webové aplikácie ukladajú dáta na vzdialenom servery, z čoho vyplýva, že nezaberajú žiadne miesto na diskoch užívateľských zariadení a šetria výkon týchto zariadení, nakoľko väčšina výpočtov prebieha na serveri. Webový prehliadač umožňuje iba prístup k aplikácií a jej obsahu, taktiež spúšťa skriptá, ktorých úlohou je oživiť dynamickú funkcionalitu aplikácie. Webové aplikácie sú multiplatformové, čo predstavuje jednu z najväčších výhod pre zákazníkov, nakoľko užívateľovi stačí na ich využívanie iba webový prehliadač, ktorý v dnešnej dobe výrobcovia implementujú do takmer každého zariadenia, ktoré môže byť pripojené k internetu. Pravidelné sťahovanie a inštalovanie aktualizácii znižuje užívateľom komfort, síce softvér si často dokáže aktualizácie stiahnuť sám, avšak užívateľ musí schváliť inštaláciu aktualizácie, chvíľu počkať a potom znovu spustiť desktopovú aplikáciu, pričom aktualizácie často prichádzajú v nevhodnom čase. Pri webových aplikáciách užívateľ pri každom spustení disponuje s najnovšou verziou.

Diplomová práca popisuje vytvorenie webovej aplikácie, ktorá má za úlohu zjednodušiť a zefektívniť fakturačné procesy. Zmyslom je vytvoriť softvérový modul, určený na fakturáciu pre klienta, ktorému nevyhovujú univerzálne, komerčné verzie. Zákazníci, ktorí využívajú aplikácie tohto typu, často pri svojom podnikaní objednávajú materiál od obchodných partnerov, prípadne využívajú kooperačné procesy. Tieto vzniknuté náklady potrebujú sprehľadniť pri následnom ocenení svojho produktu alebo analýze svojich zákaziek. Táto aplikácia umožňuje jednoduché a prehľadné priradenie týchto nákladov k jednotlivým produktom v prijatých objednávkach, následný import nákladov do odoslaných objednávok a následný prehľad ziskovosti jednotlivých objednávok.

Teoretická časť obsahuje základný rozbor použitých technológií, programovacích jazykov, knižníc a princípov, ktoré boli použité na vytvorenie aplikácie, obecné vysvetlenie spôsobu komunikácie aplikácie s klientom a následné spracovanie požiadaviek klienta. Súčasťou teoretickej časti je aj vysvetlenie funkcionality aplikácie, zamerané predovšetkým na odlišnosť tejto aplikácie od komerčných verzií.

Zmyslom praktickej časti je vytvorenie dokumentácie kódu aplikácie. Čitateľ sa oboznámi s princípom navrhnutia a vytvorenia databázy, vysvetlením princípu užívateľského rozhrania, zmyslom dynamického obsahu aplikácie a jeho následnou implementáciou, na ktorú následne nadväzuje obsluhovanie užívateľských požiadaviek. Praktická časť ďalej rozoberá možnosti zabezpečenia webových aplikácií, spôsoby obrany voči týmto hrozbám a príklad útoku na webovú aplikáciu, v zmysle pochopenia dôležitosti zabezpečenia webových aplikácií.

I. TEORETICKÁ ČÁST

1 ROZBOR POUŽITÝCH TECHNOLÓGIÍ

V tejto kapitole budú stručne popísané najdôležitejšie technológie, praktiky a spôsoby využité pri vytvorení tejto aplikácie. Taktiež budú stručne popísané jednotlivé programovacie jazyky a knižnice, ktoré budú využívané v praktickej časti.

1.1 Visual studio

Visual studio je vývojové prostredie od spoločnosti Microsoft a používa sa na rôzne typy vývoja softvéru, ako sú desktopové a webové aplikácie, webové stránky, mobilné aplikácie a mnoho ďalších. Nejde o špecifické vývojové prostredie pre konkrétny programovací jazyk, Visual studio je možné využívať na písanie kódu v jazykoch C#, VisualBasic, Python, Javascript a mnoho ďalších, celkovo poskytuje podporu pre 36 rôznych programovacích jazykov. Je k dispozícií pre operačné systémy Windows a macOS. [1]

1.1.1 Visual studio Community

Bezplatná verzia, ktorá vyšla v roku 2014. Pomocou tejto verzie môže vývojár vyvíjať svoje vlastné bezplatné alebo platené aplikácie. V podnikových organizáciách má toto vydanie určité obmedzenia, ak má táto organizácia viac ako 250 počítačov a ročný príjem vyšší ako 1 milión USD, stráca povolenie využívať túto verziu. [1]

1.1.2 Visual studio Professional

Komerčná verzia, spoločnosť Microsoft poskytuje bezplatnú skúšobnú verziu a po uplynutí si ju musí užívateľ zaplatiť. Hlavným účelom tejto verzie je poskytovať profesionálne vývojárske nástroje na vytváranie ľubovoľného typu aplikácií, výkonné funkcie na produktivitu a nástroje na agilné plánovanie projektov. [1]

1.1.3 Visual studio Enterprise

Táto verzie predstavuje integrované komplexné riešenie pre tímy akejkoľvek veľkosti s náročnými požiadavkami na kvalitu a rozsah. Verzia je spoplatnená s 90 dňovým skúšobným obdobím. [1]

1.2 ASP.NET

.NET je vývojárska platforma spoločnosti Microsoft, ktorá pozostáva z programovacích jazykov a knižníc. Poskytuje integráciu jazykov HTML, CSS a javascript. Umožňuje vývojárom vytvárať webové stránky, aplikácie a služby. ASP.NET rozširuje platformu .NET o nástroje a knižnice špeciálne na vytváranie webových aplikácií. [2]

1.2.1 Master Pages

Hlavné súbory užívateľského rozhrania ASP.NET sú označované príponou master. Umožňujú vytvoriť konzistentné rozloženie stránok v aplikácií, resp. definujú vzhľad a správanie pre celý súbor stránok v aplikácií. Táto predloha poskytuje šablónu pre webové formuláre so zdieľaným rozložením a funkciami. Hlavná stránka definuje zástupné symboly, ktoré môžu byť prepísané webovými formulármi. [3]

1.2.2 ASPX Pages

Súbory s príponou ASPX sú webové formuláre širšieho charakteru, ktoré vytvárajú nižšiu vrstvu užívateľského rozhrania. Výstupom je kombinácia mastrov a webových formulárov. [4]

1.3 SQL Server Management Studio

SQL Server Management Studio (SSMS) je integrované prostredie, ktoré sa dnes už bežne využíva na konfiguráciu a správu všetkých komponentov Microsoft SQL Servera. Umožňuje správcom databáz a vývojárom konfigurovať a spravovať všetky komponenty v rámci SQL servera. Jeho hlavnou funkcionalitou je vytváranie entít (tabuliek), vykonávanie SQL dotazov na vkladanie, aktualizáciu alebo odstraňovanie dát, vytváranie a spravovanie spúšťačov, pohľadov alebo kurzorov. Umožňuje aj nastaviť zabezpečenie k databázam a ich objektom. [5]

1.3.1 Jazyk SQL

Jazyk SQL je štandardný jazyk pre systémy správy relačných databáz. Príkazy jazyka SQL sa využívajú na vykonávanie úloh ako napríklad aktualizácia údajov v databáze, načítanie údajov z databázy, mazanie údajov alebo definíciu relačných vzťahov. SQL dokáže vykonávať mnoho ďalších operácií vrátane optimalizácie a údržby databáz. [6]

V tabuľke 1 sú definované najčastejšie využívané príkazy jazyka SQL, ktoré sa budú objavovať v tejto práci.

Tabuľka 1. Zoznam najčastejšie používaných SQL príkazov [6]

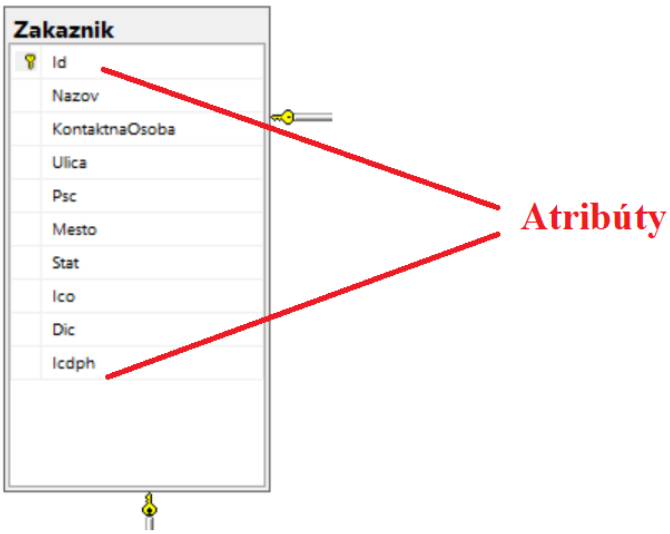
CREATE	Definuje schému štruktúry databázy
INSERT	Vkladá údaje do tabuľky
UPDATE	Aktualizuje údaje v tabuľke
DELETE	Odstraňuje riadky v tabuľke
SELECT	Vyberie atribút na základe podmienky v klauzule WHERE
DROP	Odstraňuje tabuľky z databázy

1.3.2 Entita

Pod pojmom entita si môžeme predstaviť živú alebo neživú vec zo skutočného sveta, ktorá sa ukladá do databázy. Entity sú v databáze vyjadrené formou tabuľky. V tejto práci budú entity predstavovať napr. dopyty, faktúry alebo objednávky. [7]

1.3.3 Atribút entity

Atribúty popisujú charakteristiku a vlastnosti entít, napríklad na obrázku 1 sú znázornené príklady atribútov entity zákazník. [7]



Obrázok 1. Atribúty entít

1.3.4 Relácia entít

Definuje vzťahy medzi entitami.

1. Jeden k jednému (1:1)
   Jedna entita môže byť spojená najviac s jednou entitou, napríklad jedna osoba má jedno rodné číslo a rodné číslo patrí len jednej osobe.
2. Jeden k mnohým (1:N)
   Jedna entita môže byť spojená s viacerými entitami, napríklad do jednej triedy, patrí viacero študentov.
3. Mnoho k mnohým (M:N)
   Jedna entita môže byť spojená viac ako s jednou entitou a naopak, napríklad študent môže mať viacero predmetov a na predmet môže chodiť viacero študentov. Problémom je, že relačné databázové systémy neumožňujú spojiť dve entity vzťahom M:N, riešením je oddelenie týchto dvoch entít treťou pretínajúcou entitou a vytvoriť medzi nimi dva vzťahy 1:N, pričom pretínajúca entita zvyčajne obsahuje atribúty z oboch spájajúcich sa entít.
   [8][9]

1.3.5 View (pohľad)

Pohľady v jazyku SQL sú virtuálne tabuľky, vytvorené z riadkov a stĺpcov jednej alebo viacerých skutočných tabuliek. Pohľady nevyžadujú žiadne uloženie v databáze, pretože fyzicky ani neexistujú, ale napriek tomu je možné dopytovať sa na ne rovnako, ako aj na fyzické tabuľky. Dáta, ktoré sú aktualizované v pohľadoch, sú automaticky aktualizované aj vo fyzických tabuľkách. [10]

1.3.6 Databázová schéma

Databázová schéma predstavuje množinu entít uložených v databáze a ich relácie (vzťahy). Taktiež pomáha vysvetliť logickú štruktúru databázy. [11]

1.4 Entity framework

Entity framework je objektovo-relačný mapér, ktorý podporuje vývoj dátovo orientovaných softvérových aplikácií. Vývojárom umožňuje pracovať s údajmi vo forme objektov a vlastnosťami špecifických pre doménu ako sú zákazníci a adresy zákazníkov, bez potreby zaoberať sa základnými tabuľkami a stĺpcami databázy, v ktorých sú dáta uložené. Entity

framewok taktiež umožňuje vývojárom pracovať na vyššej úrovni abstrakcie a znižuje rozsiahlosť kódu. [12]

1.4.1 Db Model Builder

Db Model Builder je kódovo orientovaný prístup, ktorý sa využíva na mapovanie tried do databázovej schémy. [13]

1.5 Frontend a Backend

Na internete je možné nájsť množstvo definícií rozdielov medzi frontend a backend programovaním, z môjho pohľadu najzrozumiteľnejšie vysvetlenie je, že frontend je všetko to, čo vidí používateľ danej služby a pri webových aplikáciách sa vykonáva na strane klienta, teda vo webovom prehliadači a „vykresľuje“ vzhľad aplikácie. Backend je používateľom neprístupný kód, ktorý beží na strane servera, pričom používateľovi posiela len výsledky. [14]

1.6 Skriptovacie jazyky

Skriptovacie jazyky sú programovacie jazyky, ktoré vykonávajú úlohy v špeciálnom prostredí pomocou tlmočníka namiesto kompilátora. Sú zvyčajne krátke, rýchle a interpretované zo zdrojového kódu alebo bajtkódu. Zvyčajne sa interpretujú za behu, ako pri kompilácií kódu. Skriptovacie jazyky sú multiplatformové, teda na spustenie nevyžadujú inštaláciu špeciálneho softvéru (samozrejme okrem webového prehliadača). Skriptovacie jazyky môžu „bežať“ na strane serveru (backend), tento prístup je bezpečnejší, pretože hlavná časť kódu beží na severy a návštevník ju nevidí. V tejto práci budú identifikovať užívateľské požiadavky a odosielať ich na server. Druhý variant je na strane klienta (frontend). V tomto prípade už celý kód beží vo webovom prehliadači a návštevník má prístup k celému kódu. Tento princíp bude v praktickej časti dynamicky meniť užívateľské rozhranie z dôsledku užívateľských aktivít. [15]

1.6.1 JavaScript

JavaScript je programovací jazyk, ktorý sa používa na strane klienta, ale aj servera. A dáva webovým aplikáciám interaktívne prvky, ako napríklad interaktívne mapy, animácie a mnoho ďalších. Neexistujú žiadne obmedzenia pre veci, ktoré môžeme robiť

s JavaScriptom na webovej stránke, napr.: kliknutím na tlačidlo je možné zobraziť alebo skryť informácie, prehrávať zvuk alebo video na webovej stránke, zobrazovať animácie a mnoho ďalších. Taktiež môžeme pomocou JavaScriptu overovať užívateľské vstupy na webovú aplikáciu, vďaka tomu nesprávne vstupy od užívateľov nezaťažujú server. [16]

1.6.2 jQuery

Jedná sa o malú multiplatformovú knižnicu JavaScriptu, ktorá je bohatá na funkcie. Jej úlohou je zjednodušiť skriptovanie na strane klienta. Programovanie v JavaScripte často vyžaduje veľké množstvo kódu. Vďaka tejto knižnici, si môžeme jednoducho zavolať mnohé funkcie pomocou jedného riadku kódu. Umožňuje vývojárom vytvárať abstrakcie, pokročilé efekty a tematické widgety na vysokej úrovni. [17]

1.7 Regulárne výrazy

Regulárne výrazy sú sekvencie znakov, ktoré slúžia ako vzor popisujúci text. V tejto práci budú pomocou regulárnych výrazov overované vstupy aplikácie. Ich úlohou bude eliminovať nesprávne vstupy, napríklad cena nesmie obsahovať iné znaky ako číslice, s výnimkou znamienok „-,.“. [18]

1.8 Jazyk C#

Jazyk C# je moderný objektovo orientovaný a typovo bezpečný programovací jazyk, ktorý umožňuje vývojárom vytvárať robustné aplikácie, ktoré bežia v architektúre .NET. Bol vytvorený spoločnosťou Microsoft. Pochádza z rodiny jazykov C. C# dokáže automaticky získavať späť nevyužitú obsadenú pamäť. [19]

1.8.1 LINQ

Language Integrated Query je súbor technológií založených na integrácii možností dopytovania priamo z jazyka C#. LINQ umožňuje prístup k dátam, modifikáciu dát a mnoho ďalších funkcií, bez ohľadu na ich zdroj, ktorým môžu byť napríklad entity databázy alebo XML súbor. [20]

1.8.2 PDF sharp

PDF sharp je knižnica OpenSource, ktorá dokáže vytvoriť a spracovať dokumenty typu PDF za chodu akéhokoľvek jazyka v architektúre .NET. Taktiež dokáže kresliť do doku-

mentov rôzne útvary, alebo odoslať výstup na tlačiareň. PDF sharp spadá pod licenciu MIT a jej využitie je bezplatné. [21]

1.9 API

API sú konštrukcie dostupné v programovacích jazykoch, ktoré umožňujú vývojárom jednoduchšie vytvárať komplexné funkcie, a ďalej vývojárom poskytujú miesto zložitého kódu, jednoduchšiu syntax. [22]

1.9.1 Web API

Web API je aplikačné programovacie rozhranie pre web, ktoré rozširuje funkčnosť webového prehliadača. Dnes už všetky prehliadače obsahujú sadu vstavaných API na podporu zložitých operácií a uľahčenie prístupu k dátam. [23]

1.10 Značkovacie jazyky

Značkovacie jazyky obsahujú skôr štandardné slová, než programovaciu syntax a ich úlohou je daný text vykresliť v aplikácií pomocou značiek, ktoré sa prelínajú s primárnym textom. Značky sú zodpovedné za to, kde sa prvok začína a kde končí. Tieto značky sa objavujú v dokumentoch iba vtedy, keď autor píše daný text vo vývojovom prostredí. Keď aplikácia spracuje „značkovaný kód“, zobrazí sa užívateľovi formátovaný a upravený text. [24]

1.10.1 Jazyk HTML

Jazyk HTML je jedným z najčastejšie využívaných značkovacích jazykov v prehliadačoch. Nepovažuje sa za programovací jazyk, pretože neumožňuje vytvárať dynamickú funkcionalitu aplikácie, umožňuje vytvárať štruktúru webových stránok a ich prepojenie. Taktiež umožňuje oddeľovať odseky, označovať nadpisy, zvýrazniť písmo a pod. Značky informujú webový prehliadač o tom, kde sa prvok nachádza, a kde končí, zatiaľ čo atribút popisuje vlastnosti prvku. Jazyk HTML je považovaný za oficiálny webový štandard. [24]

1.10.2 Kaskádové štýly

Kaskádový štýl alebo CSS je programovací jazyk, ktorý sa využíva na úpravu dokumentov vytvorených pomocou značkovacích jazykov, resp. popisuje, ako by sa mali zobrazovať prvky HTML v užívateľskom rozhraní. Umožňuje napríklad ovládať farbu textu alebo štýl písma. Definíciu jazyka CSS je možné napísať do tried a tieto triedy nie je problém nadefi-

novať na viacerých stránkach HTML dokumentu, taktiež každý prvok v jednom HTML dokumente, môže mať inú triedu. Na obrázku 2 je znázornený príklad definície jazyka CSS. [25]

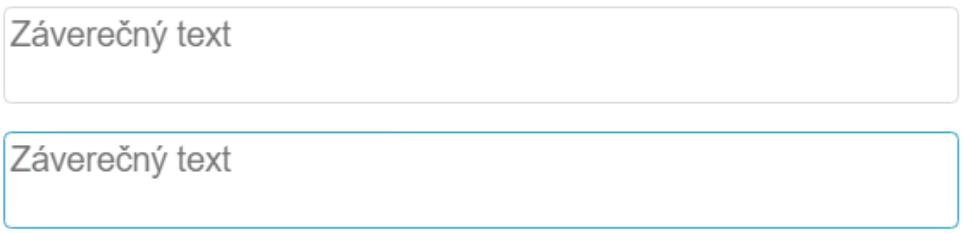
```
67 .zaverecny-text {
68     border: 1px solid #dadada;
69     padding: 2px;
70     overflow: hidden;
71 }
```

Obrázok 2. Definícia triedy v jazyku CSS

CSS umožňuje jednoduchú dynamiku užívateľského rozhrania podobne ako javascript, využíva sa predovšetkým na jednoduché animácie a grafické zmeny užívateľského rozhrania, zvyčajne ako dôsledok užívateľskej aktivity. Názorný príklad na obrázku 3 ukazuje, ako definícia triedy v jazyku CSS dokáže zvýrazniť textový vstup aplikácie, v prípade, ak užívateľ klikne na daný vstup. Výsledná dynamika je znázornená na obrázku 4.

```
72
73     .zaverecny-text:focus {
74         border-color: rgb(42, 164, 207);
75     }
```

Obrázok 3. Definícia dynamiky v triede jazyka CSS

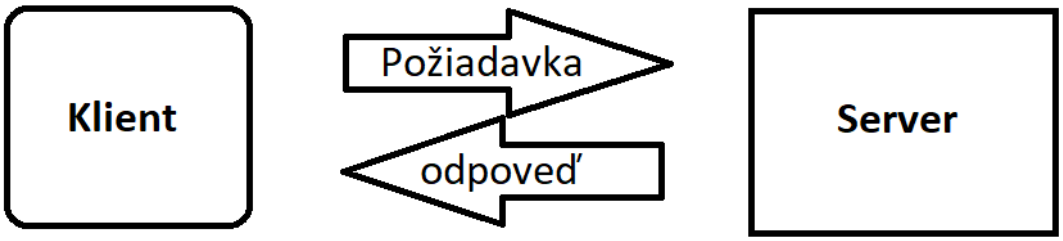


Obrázok 4. Výsledok triedy jazyka CSS

1.11 HTTP protokol

HTTP je protokol aplikačnej vrstvy, ktorý umožňuje komunikáciu medzi klientom a webovým serverom. Komunikácia prebieha formou odosielania HTTP požiadaviek a prijímania HTTP odpovedí, znázornená je na obrázku 5. [26]

1. Klient odošle na web HTTP požiadavku
2. Webový server príjme požiadavku
3. Server spracuje požiadavku
4. Server odošle klientovi HTTP odpoveď



Obrázok 5. Komunikácia klient-server

1.11.1 Metóda GET

Dáta odosielané metódou GET sú pripojené na požiadavku, pričom sú viditeľné pre všetkých, resp. všetky názvy premenných a hodnoty sú viditeľné v URL adrese, z tohto dôvodu nie je vhodné používať metódu GET na odosielanie citlivých údajov, ako sú napríklad heslá. [27]

1.11.2 Metóda POST

Dáta odosielané metódou POST sú obsiahnuté v hlavičke HTTP, pričom nie sú viditeľné v URL adrese. V tabuľke 2 sa nachádza porovnanie metódy GET a POST. [27]

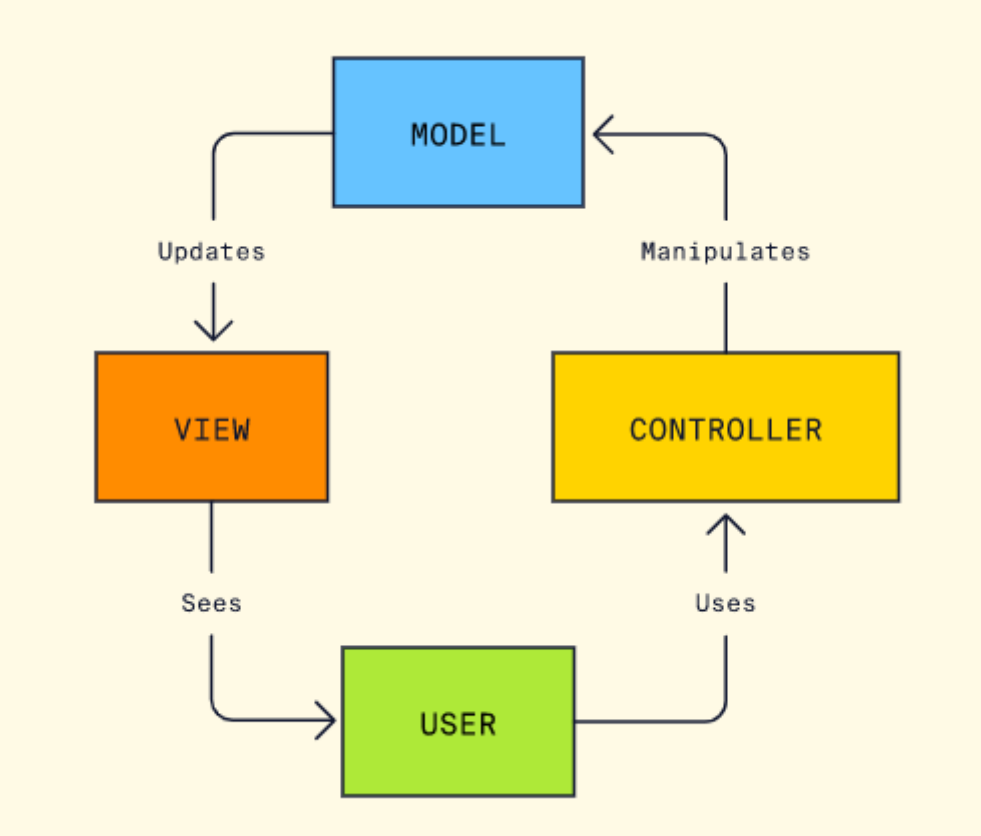
Tabuľka 2. Rozdiel medzi metódou GET a POST [27]

GET	POST
Odosielané dáta sú obmedzené na maximálne 1500 znakov.	Odosielané dáta nemajú obmedzenú veľkosť.
Využíva sa na odosielanie dát, ktoré nie sú citlivé.	Využíva sa aj na odosielanie citlivých dát.
Nie je príliš bezpečná.	Bezpečnejšia ako GET.
Stránku je možné uložiť ako záložku.	Stránku nie je možné uložiť ako záložku.

1.12 Model – View – Controller (MVC)

MVC je architektúra, ktorá sa využíva pre vytvorenie spôsobu organizácie kódu. Každá časť kódu má svoj účel a tieto účely sú odlišné. Jedna časť archivuje dáta, druhá časť kreslí štruktúru a vzhľad a tretia časť riadi funkcionalitu aplikácie. [28]

MVC umožňuje prehľadne usporiadať tieto sekcie kódu, čím robí vývoj aplikácií podstatne jednoduchším. Na obrázku 6 je znázornená schéma architektúry MVC. [28]



Obrázok 6. Schéma MVC [28]

Model

Zodpovedá za všetky dáta, spojené s touto logikou, resp. údaje prenášané medzi komponentmi View a Controller. Model môžeme charakterizovať aj ako backend, ktorý obsahuje všetku dátovú logiku. [28][29]

View

View sa využíva pre celú logiku používateľského rozhrania aplikácie. Napríklad textové polia alebo rozbaľovacie zoznamy, resp. všetky elementy užívateľského rozhrania, s ktorými užívateľ komunikuje. [28]

Controller

Controller funguje ako riadiaca jednotka medzi komponentmi Model a View na spracovanie prenosovej logiky požiadaviek, manipuláciu s údajmi v komponente Model a interakciu s komponentom View pre vykreslenie konečného výstupu. [28]

API Controller

Úlohou API Controllera je reagovať na HTTP požiadavky na získanie, odoslanie, vloženie a odstránenie od klientov formou API. Na obrázku 7 je možné vidieť príklady spracovania HTTP požiadaviek. [30]

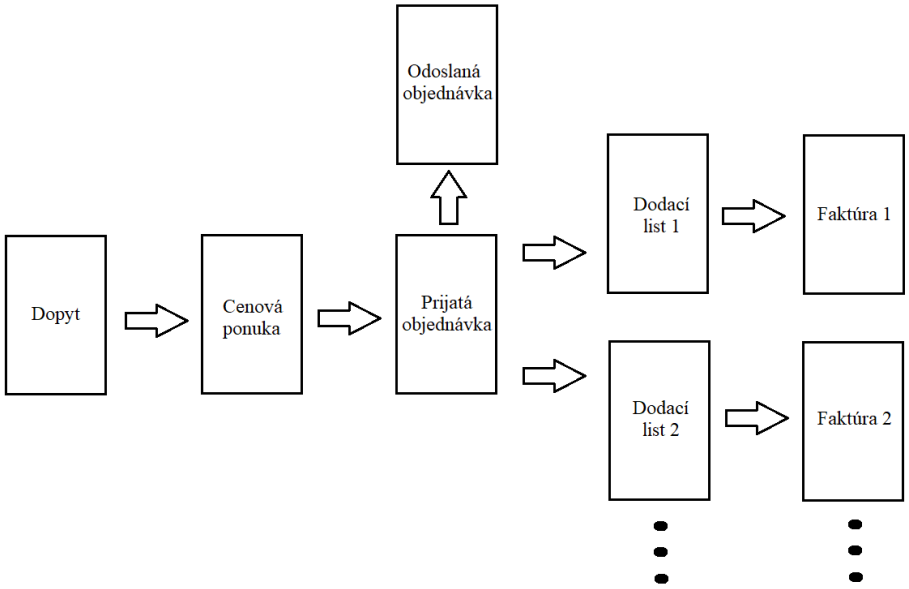
Obrázok 7. Metódy HTTP požiadaviek [30]

2 VÝZNAM SOFTWÉROVÉHO MODULU

Hlavnou úlohou webovej aplikácie je zjednodušiť fakturačné procesy formou jednoduchého vytvárania a modifikovania obchodných dokumentov, resp. dopyty, cenové ponuky, prijaté objednávky, odoslané objednávky, dodacie listy a faktúry priamo v aplikácií. Aplikácia je univerzálna z pohľadu možnosti realizácie kompletného fakturačného procesu, či už sa jedná o predaj výrobkov, alebo poskytovanie služieb. Dnes už je dostupných množstvo komerčných verzií podobných aplikácií, napríklad KROS alebo SunSoft. Avšak tieto verzie sú zvyčajne spoplatnené a nie sú optimálnym riešením pre klienta.

2.1 Funkcionalita aplikácie

Užívateľ v aplikácií vytvára nový dopyt, z ktorého si následne postupne generuje ostatné doklady vyššej vrstvy. Položky sú vždy približnou kópiou predchádzajúceho dokladu, to ale neznamená, že sa nemôžu napr. v prijatej objednávke upravovať. Užívateľ si môže taktiež pozrieť prehľad zameraný na výpočet ziskov z predajov, vygenerovať PDF dokument z dokladov a pod. Schéma na obrázku 8 popisuje postup generovania dokladov.

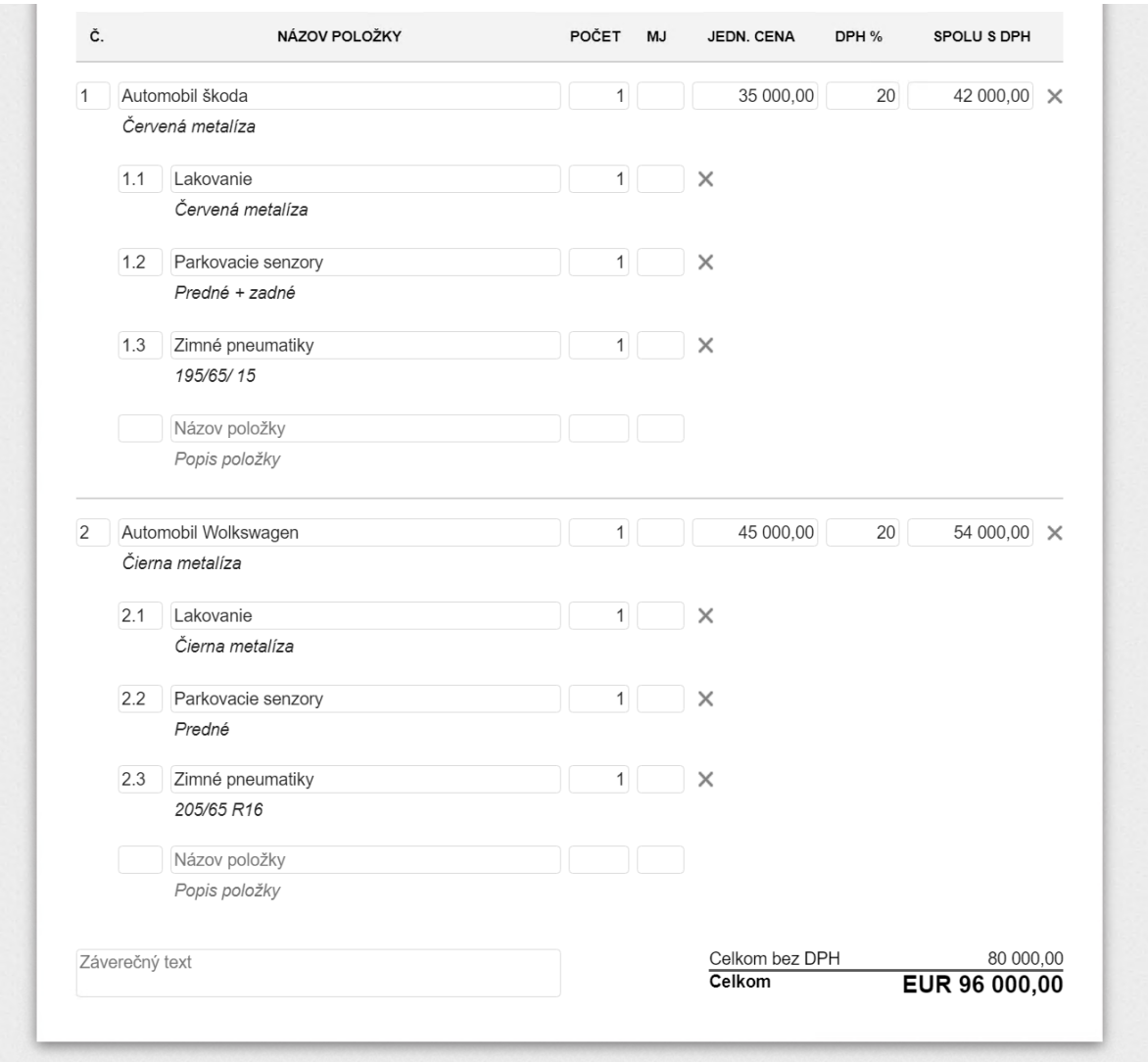


Obrázok 8. Schéma funkcionality aplikácie

2.2 Originalita vytvorenej aplikácie

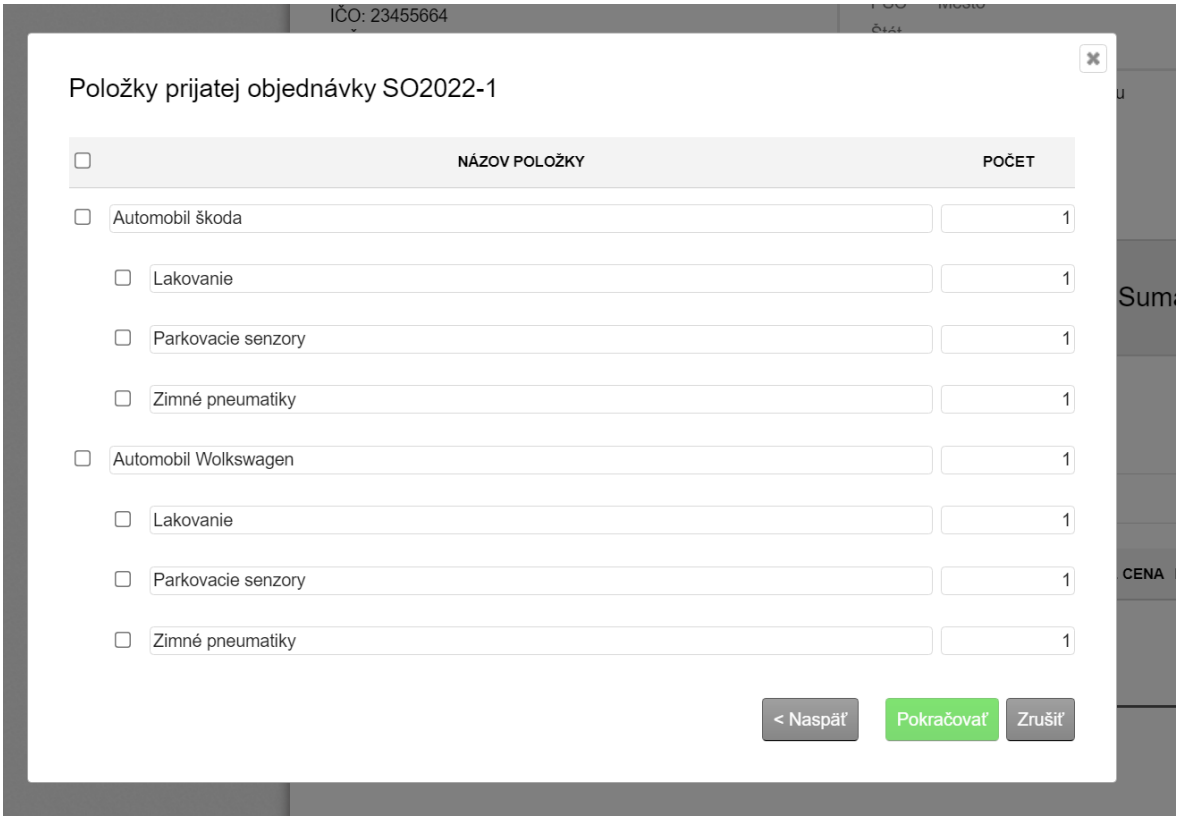
Unikátnosť vytvorenej aplikácie spočíva v spôsobe riadenia nákladov a obchodných kooperácií. Obchodné kooperácie sa využívajú v prípade, ak užívateľ softvérového modulu pre zhotovenie produktu, ktorý následne odošle na trh, využíva služby iných obchodných partnerov.

Užívateľ môže v prijatých objednávkach pridávať k položkám podpoložky, ktorých úlohou je sprehľadniť náklady a prípadné obchodné kooperácie k daným položkám. Podpoložky sú viditeľné iba v prijatých objednávkach, prípadne v odoslaných objednávkach. Zákazník k týmto informáciám nemá prístup, pri generovaní faktúr a dodacích listov zanikajú. Na obrázku 9 je znázornený príklad podpoložiek prijatej objednávky.



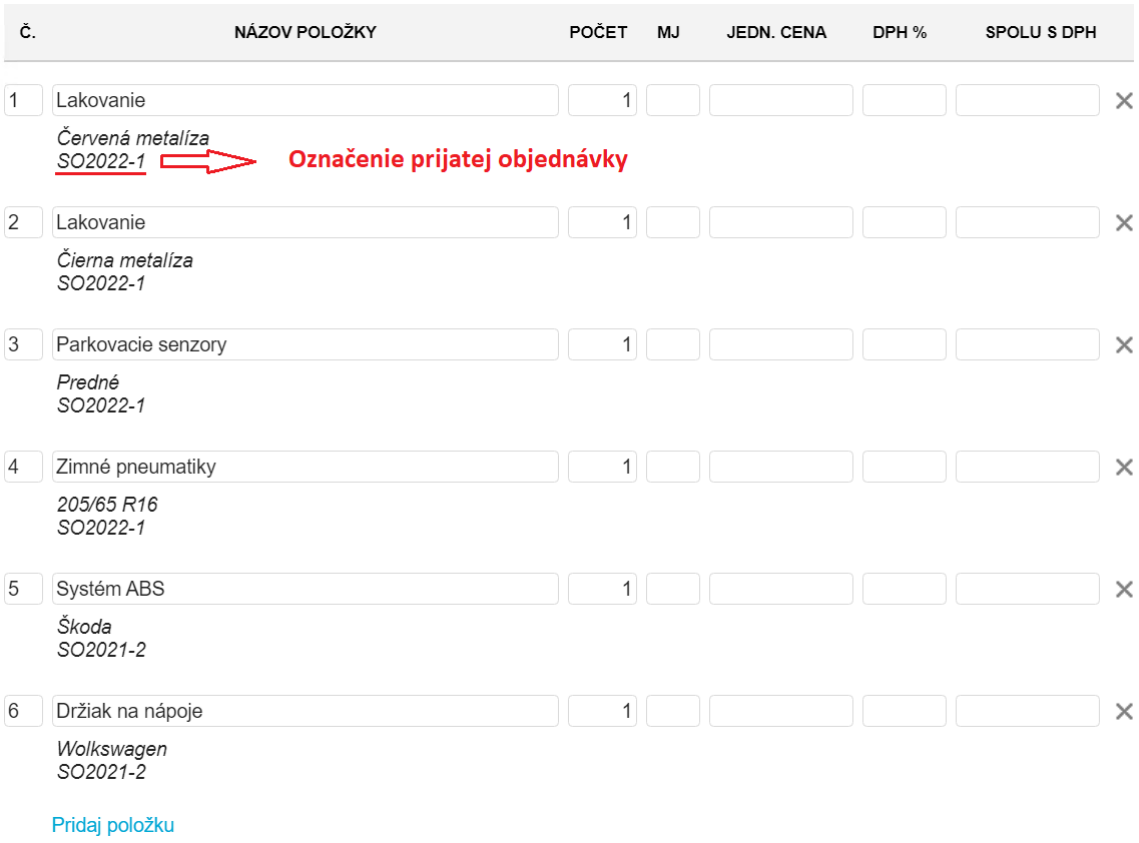
Obrázok 9. Vkladanie položiek a podpoložiek do prijatej objednávky

Tieto podpoložky môže užívateľ následne jednoduchým spôsobom importovať do odoslaných objednávok označením check boxov na obrázku 10. Importovať môže aj samotné položky, v prípade ak by chcel celý obchod realizovať cez kooperácie.



Obrázok 10. Import podpoložiek z prijatej do odoslanej objednávky

Všetky pridané položky v odoslanej objednávke nesú označenie prijatej objednávky. Vďaka tomu užívateľ vidí, z ktorej prijatej objednávky pochádzajú, ako je znázornené na obrázku 11.



Obrázok 11. Položky odoslanej objednávky

2.3 Prehľad stavu prijatých objednávok

V aplikácii, ktorá bola vytvorená v tejto práci je prehľad určený pre užívateľa, ktorý všetky položky aj podpoložky prijatej objednávky objednáva z iných zdrojov prostredníctvom odoslaných objednávok. Užívateľ si následne môže v prehľade pozrieť aktuálny stav prijatej objednávky.

Obrázok 12. Prehľad stavu prijatých objednávok

Zelený bod na obrázku 12 znamená, že prijatá objednávka bola kompletne spracovaná, resp. všetky položky aj podpoložky boli importované do odoslanej objednávky, teda boli objednané prostredníctvom odoslanej objednávky.

Červený bod na obrázku 12 znamená, že prijatá objednávka je v priebehu spracovania, resp. aspoň jedna položka alebo podpoložka bola importovaná do odoslanej objednávky.

Sivý bod na obrázku 12 znamená, že v danej prijatej objednávke spracovanie zatiaľ neprebehlo vôbec, resp. ani jedna položka alebo podpoložka, nebola importovaná do odoslanej objednávky.

Prehľad nákladov

Užívateľ si môže v prehľadoch vybrať prijatú objednávku a zobraziť si skutočné náklady, ktoré sú vypočítane z položiek v odoslaných objednávkach, ktoré patria k danej prijatej objednávke. Na obrázku 13 je znázornený príklad prehľadu nákladov.

Prijatá objednávka SO2022-1	
Suma na objednávke	1 640,00 €
Skutočné náklady	686,00 €

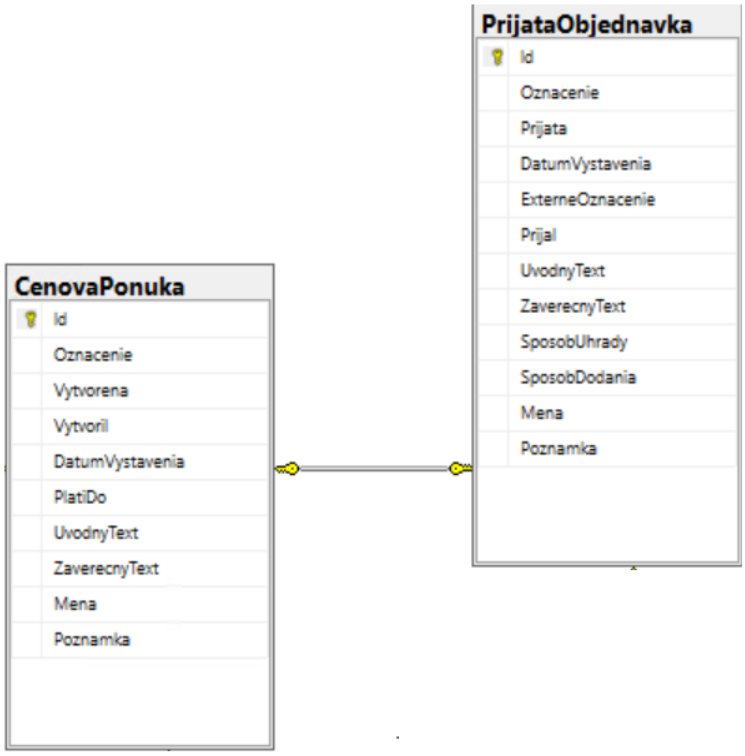
Obrázok 13. Prehľad nákladov prijatej objednávky

II. PRAKTICKÁ ČÁST

3 VYTVORENIE DATABÁZY

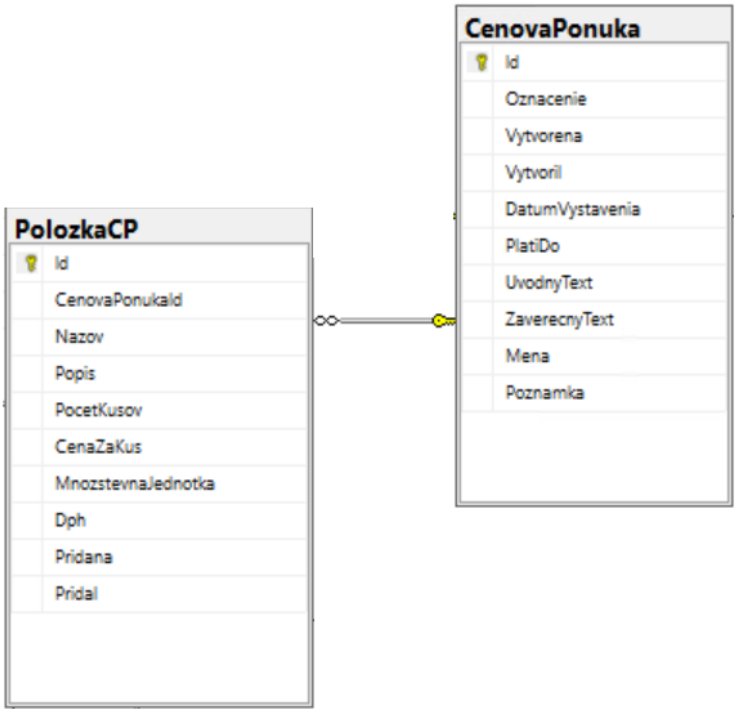
Pre vytvorenie databázy bol využitý softvér SQL Server Management Studio. Databáza obsahuje 13 entít, každá entita má svoje atribúty. Napríklad entita „PrijataObjednavka“ obsahuje atribúty: Id, označenie, dátum prijatia, dátum vystavenia a ďalšie, ktoré sú zobrazené na obrázku 14.

Entity sú navzájom pospájané pomocou relácií. Napríklad k cenovej ponuke na obrázku 14 môže byť vytvorená vždy len jedna prijatá objednávka, pričom aj prijatá objednávka patrí vždy iba k jednej cenovej ponuke.



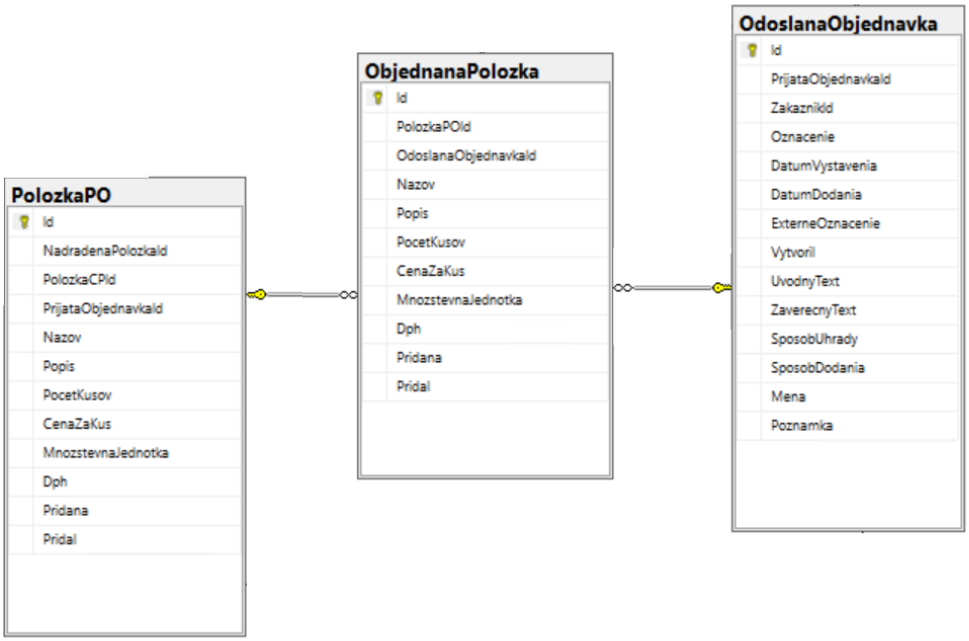
Obrázok 14. Relácia entít 1:1

Cenová ponuka na obrázku 15 môže mať viacero položiek, ale samotná položka patrí vždy iba k jednej cenovej ponuke.



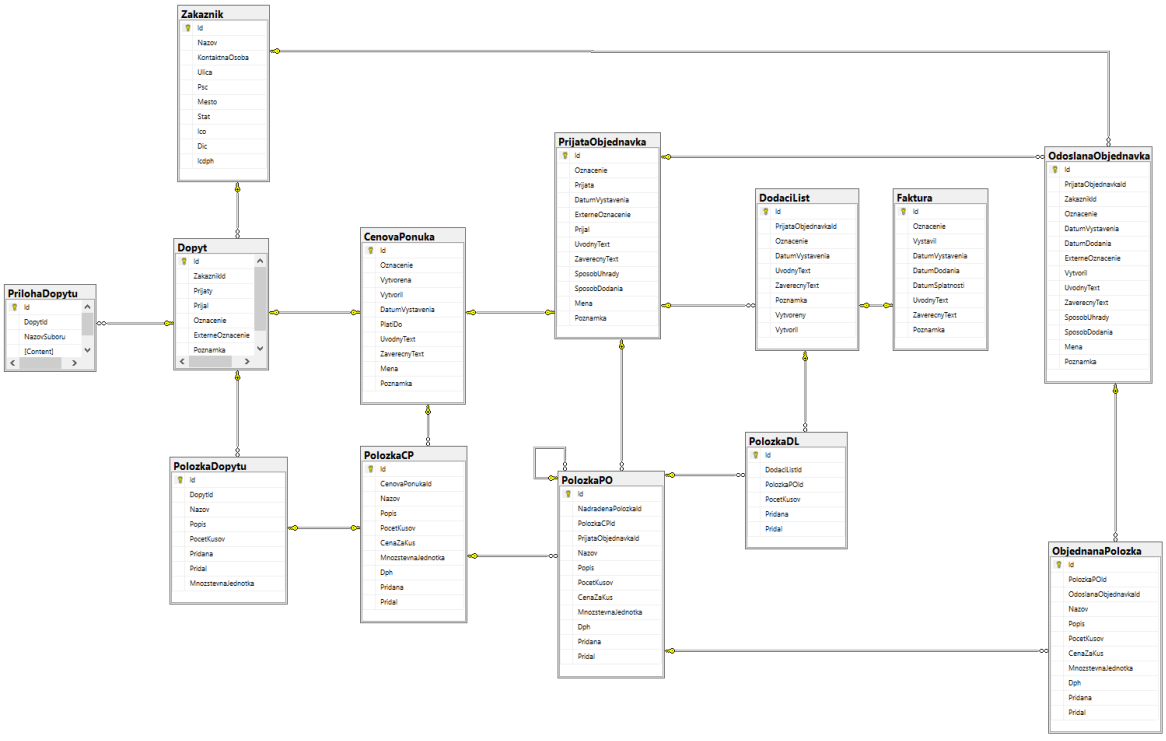
Obrázok 15. Relácia entít 1:N

Ako už bolo spomenuté v kapitole 1.3.4, jazyk SQL neumožňuje vytvoriť reláciu M:N medzi 2 entitami, v mojom prípade napríklad položka prijatej objednávky na obrázku 16, môže byť obsiahnutá vo viacerých odoslaných objednávkach, a zároveň odoslaná objednávka, môže obsahovať viacero položiek prijatej objednávky. V tomto prípade bola vytvorená pretínajúca tabuľka „ObjednanaPolozka“.



Obrázok 16. Relácia entít M:N

Na obrázku 17 je znázornená kompletná databázová schéma aplikácie.



Obrázok 17. Schéma databázy

Každá entita popisujúca doklad má priradenú entitu položka. Táto redundancia je nevyhnutná z dôvodu nezhody atribútov položiek medzi jednotlivými dokladmi. Napríklad „ObjednanaPolozka“ je importovaná z prijatej objednávky do odoslanej objednávky, resp. je vytvorená z „PolozkyPO“, avšak atribúty budú rozdielne, pretože danú položku užívateľ v odoslanej objednávke nakupuje od dodávateľa, a v prijatej objednávke predáva odberateľovi.

3.1 Označenie dokladov

Obrázok 18 obsahuje funkciu, ktorá automaticky generuje označenia dokladov.

```
ALTER FUNCTION [dbo].[GenerujOznacenieCP]()
RETURNS nvarchar(12)
AS
BEGIN
    DECLARE @oznacenieCP nvarchar(12);
    DECLARE @posledneOznacenie nvarchar(12) = (SELECT TOP 1 [Oznacenie] FROM [dbo].[CenovaPonuka] ORDER BY [Id] DESC);
    DECLARE @rok int = YEAR(GETDATE());
    IF(@posledneOznacenie IS NULL)
    BEGIN
        SET @oznacenieCP = CONCAT(N'QTN', @rok, N'-1');
    END
    ELSE
    BEGIN
        DECLARE @poslednyRok int = SUBSTRING(@posledneOznacenie, 4, 4);
        IF(@rok != @poslednyRok)
        BEGIN
            SET @oznacenieCP = CONCAT(N'QTN', @rok, N'-1');
        END
        ELSE
        BEGIN
            DECLARE @poradoveCislo int = SUBSTRING(@posledneOznacenie, 9, LEN(@posledneOznacenie)-8);
            SET @oznacenieCP = CONCAT(N'QTN', @rok, N'-', @poradoveCislo+1);
        END
    END
    RETURN @oznacenieCP;
END
GO
```

Obrázok 18. Funkcia generujúca označenie dokladu

3.2 Stav prijatej objednávky

Aktuálny stav prijatej objednávky v prehľade je počítaný priamo v databáze pomocou pohľadu a znázornený na obrázku 19. Skupina dotazov jazyka SQL najskôr zistí počet položiek a podpoložiek prijatej objednávky, ktoré už boli importované do odoslaných objednávok.

```
ALTER VIEW [dbo].[PolozkaStavView]
AS
SELECT [a].[Id],[a].[PrijataObjednavkaId],(SELECT COUNT([Id]) FROM [dbo].[ObjednanaPolozka] WHERE [PolozkaPOId]=[a].[Id]) AS [PocetObjednani]
FROM [dbo].[PolozkaPO] [a]
WHERE [a].[NadradenaPolozkaId] IS NULL AND [a].[Id] NOT IN (SELECT [NadradenaPolozkaId] FROM [dbo].[PolozkaPO] WHERE [NadradenaPolozkaId] IS NOT NULL)
UNION
SELECT [a].[Id],[a].[PrijataObjednavkaId],(SELECT COUNT([Id]) FROM [dbo].[ObjednanaPolozka] WHERE [PolozkaPOId]=[a].[Id]) AS [PocetObjednani]
FROM [dbo].[PolozkaPO] [a]
WHERE [a].[NadradenaPolozkaId] IS NOT NULL
GO
```

Obrázok 19. Sčítanie importovaných položiek do odoslaných objednávok

Následne v druhom pohľade na obrázku 20 je porovnaný počet importovaných položiek a podpoložiek do odoslanej objednávky s celkovým počtom položiek a podpoložiek v prijatej objednávke. Podľa výsledku je prijatej objednávke priradený stav „ukončená“, „rozpracovaná“ alebo „prijatá“.

```
ALTER VIEW [dbo].[POZoznamView]
AS
SELECT [a].[Id],[a].[Prijata],[a].[Oznacenie],[a].[Poznamka],[c].[Nazov]AS[NazovZakaznika],
(CASE
    WHEN ((SELECT COUNT([Id]) FROM [dbo].[PolozkaStavView] WHERE [PrijataObjednavkaId]=[a].[Id] AND [PocetObjednani] > 0) =
    (SELECT COUNT([Id]) FROM [dbo].[PolozkaStavView] WHERE [PrijataObjednavkaId]=[a].[Id])) THEN N'Ukončená'
    WHEN ((SELECT COUNT([Id]) FROM [dbo].[PolozkaStavView] WHERE [PrijataObjednavkaId]=[a].[Id] AND [PocetObjednani] > 0) > 0) THEN N'Rozpracovaná'
    ELSE 'Prijatá'
END) AS Stav
FROM [dbo].[PrijataObjednavka] [a]
JOIN [dbo].[Dopyt] [b] ON [a].[Id]=[b].[Id]
LEFT JOIN [dbo].[Zakaznik] [c] ON [b].[ZakaznikId]=[c].[Id]

GO
```

Obrázok 20. Priradenie stavu prijatej objednávke

3.3 Výpočet nákladov prijatej objednávky

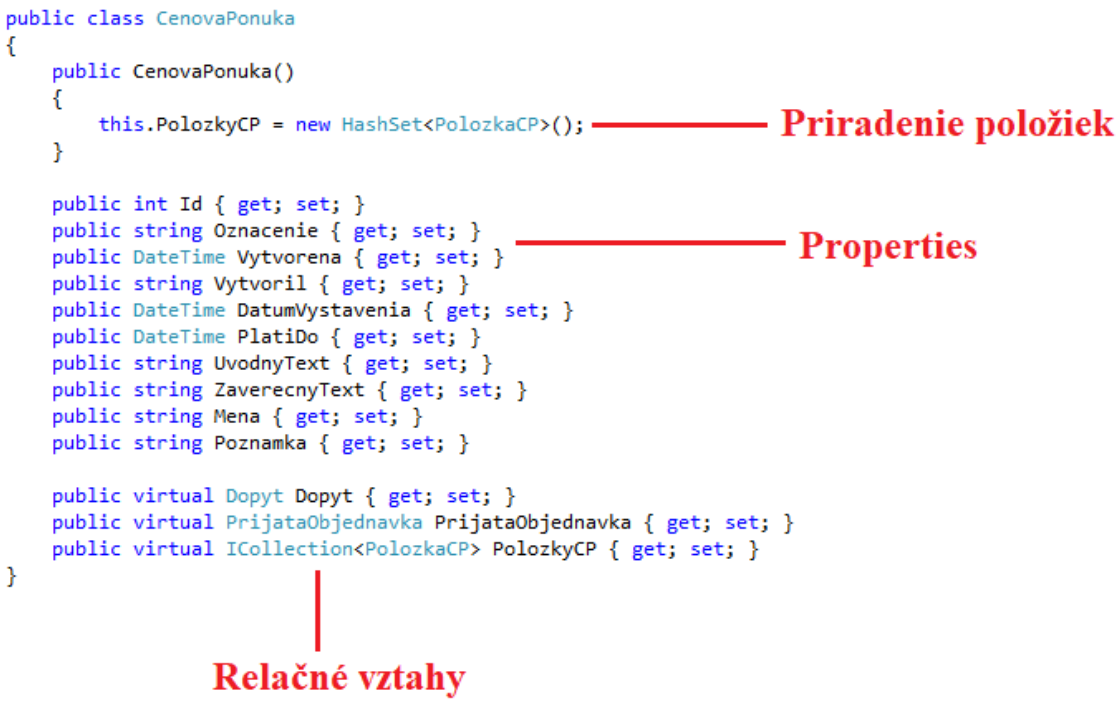
Náklady prijatých objednávok sú taktiež počítané v pohľadoch, na obrázku 21 sa najskôr vypočíta suma za jednotlivé položky objednaných položiek a podpoložiek, resp. položiek a podpoložiek prijatej objednávky importovaných do odoslanej objednávky podľa počtu kusov, a následne sa vypočítajú celkové náklady prijatej objednávky.

```
ALTER VIEW [dbo].[CelkoveNakladyObjednavkaView]
AS
SELECT [b].[PrijataObjednavkaId], SUM(ROUND([a].[CenaZaKus] * [a].[PocetKusov], 2, 0)) AS [Suma]
FROM [dbo].[ObjednanaPolozka] [a]
JOIN [dbo].[PolozkaPO] [b] ON [a].[PolozkaPOId]=[b].[Id]
GROUP BY [b].[PrijataObjednavkaId]
GO
```

Obrázok 21. Celkové náklady prijatej objednávky

4 MAPOVANIE TRIED

Pre spojenie aplikácie s databázou boli vytvorené objektové triedy pomocou jazyka C#, v ktorých je potrebné definovať aj relačné vzťahy. Na obrázku 22 je možné vidieť objektovú triedu cenovej ponuky.



Obrázok 22. Vytvorenie objektovej triedy

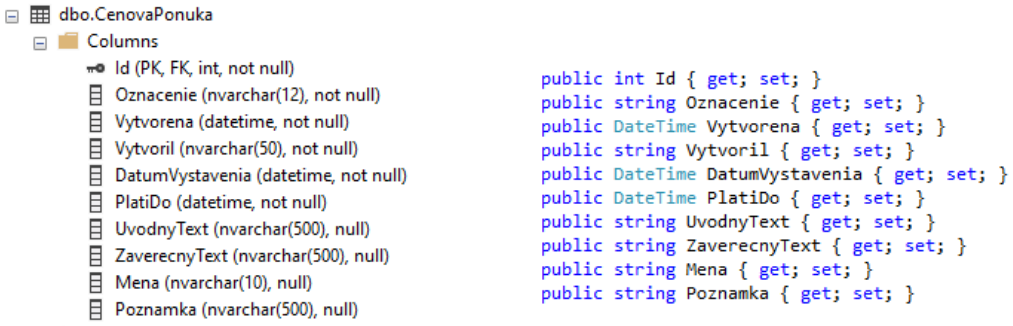
Properties ID a označenie budú generované automaticky v databáze, zvyšné properties bude užívateľ zadávať pomocou užívateľského rozhrania aplikácie, prípadne budú automaticky generované v jazyku C#.

Následne na obrázku 23 je znázornený príklad mapovania objektovej triedy na príslušnú entitu v databáze.

```
modelBuilder.Entity<CenovaPonuka>().ToTable("CenovaPonuka", "dbo");
```

Obrázok 23. Mapovanie entít na objektové triedy

Mapovanie properties a atribútov entít vytvára entity framework automaticky, avšak je dôležité, aby dátové typy a názvy properties v objektovej triede, boli totožné s atribútmi entít v databáze. Názorný príklad ukazuje obrázok 24.



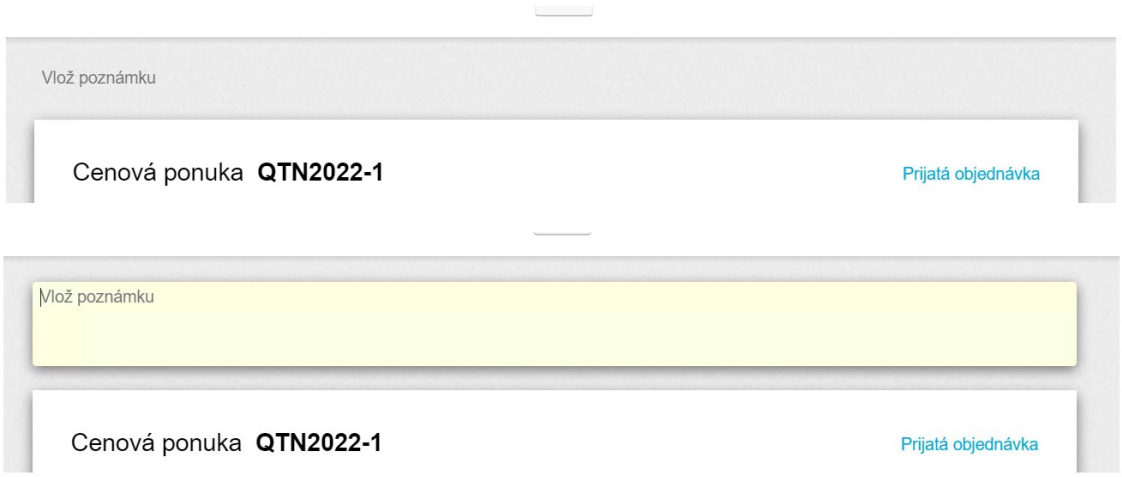
Obrázok 24. Porovnanie atribútov a properties

5 UŽÍVATEĽSKÉ ROZHRANIE

Užívateľské rozhranie je vytvorené v jazyku HTML, ktorý pomocou jednoduchých elementov, vytvára základnú štruktúru webovej stránky a jazyka CSS, ktorého úlohou je elementy vytvorené v jazyku HTML upraviť, resp. definovať ich vzhľad pomocou tried, ktoré sa následne implementujú do elementov v jazyku HTML. Obrázok 25 demonštruje vytvorenie elementu užívateľského rozhrania.

Obrázok 25. Vytvorenie elementu užívateľského rozhrania

Do vzhľadu užívateľského rozhrania často zasahujú aj funkcie javascriptu. Jednou z úloh javascriptu je dynamicky meniť užívateľské rozhranie počas behu aplikácie, napríklad v dôsledku užívateľskej aktivity. Tieto zmeny dynamiky sú najčastejšie definované v jazyku CSS, v javascripte je možné definovať riadenie tejto dynamiky. Na obrázku 26 je znázornená zmena dynamiky užívateľského rozhrania, ktorá je definovaná na obrázku 27.



Obrázok 26. Príklad dynamiky užívateľského rozhrania

```
$('[data-inputname="poznamka"]').on("change", function (e) {
    if ($(this).val().isEmptyOrWhiteSpace()) {
        $(this).removeClass("poznamka-vyplnena");
        $(this).val("");
    } else {
        $(this).addClass("poznamka-vyplnena");
    }
});


if (typeof data.Poznamka !== 'undefined') {
    parentElement = $("input[value='" + data.id + "']").prev();
    element = $(parentElement).find(".valpoznamka");
    if (data.Poznamka.length > 0) {
        $(element).text(data.Poznamka);
        $(element).closest("tr").css('display', 'table-row');
    } else {
        $(element).text("");
        $(element).closest("tr").css('display', 'none');
    }
}
```

Obrázok 27. Riadenie dynamiky webovej aplikácie v javascripte

Základ užívateľského rozhrania tejto aplikácie pozostáva z dvoch súborov „masterpage“. Zvyšná časť užívateľského rozhrania je vytvorená pomocou webových formulárov.

5.1 Webapp.Master

Úlohou tohto mastera je vytvoriť „lištu“ v hornej časti užívateľského rozhrania. Táto lišta je zobrazená na obrázku 28 a slúži užívateľovi ako navigácia. Pomocou nej môže užívateľ

otvárať všetky webové formuláre a bude zobrazená počas celého behu aplikácie, bez ohľadu, na ktorom formulári sa užívateľ nachádza.

Obrázok 28. Navigácia užívateľského rozhrania

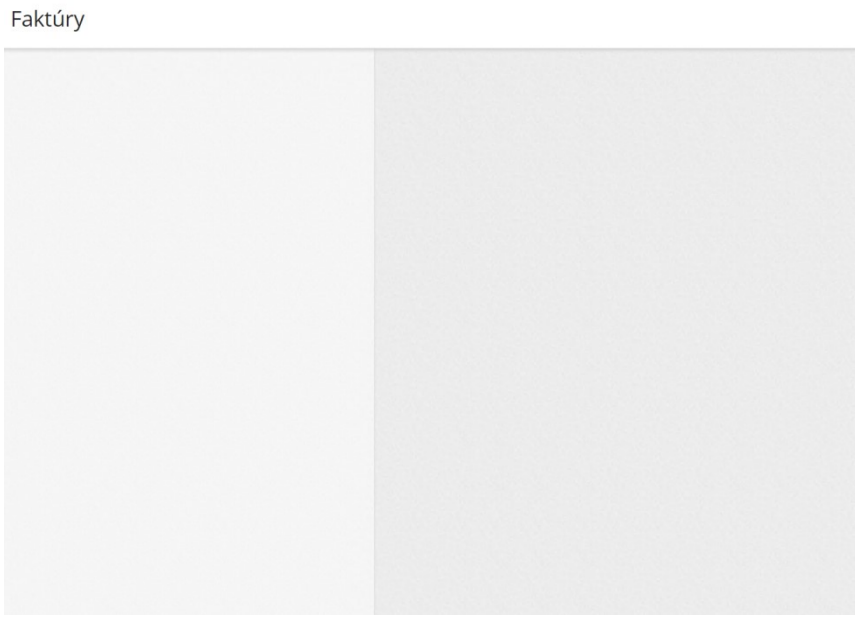
Menu obsiahnuté v lište je vytvorené na obrázku 29 pomocou jazyka HTML spolu s mapovaním na triedy jazyka CSS, ktorých úlohou je vykresliť vzhľad jednotlivých prvkov.

```
<nav id="main-nav">
    <ul class="main-nav-menu">
        <li class="<%# Prehlady %>">
            <a href="/prehlad"><span>Prehľad</span></a>
        </li>
        <li class="<%# Faktury %>">
            <a href="/faktury"><span>Faktúry</span></a>
        </li>
        <li class="<%# Obchod %>" data-dropdown-name="ddObchod">
            <div class="mainmenu-item-wrapper">
                <div>Obchod</div>
                <div class="icon-dropdown"></div>
            </div>
        </li>
    </ul>
</nav>

<div id="ddObchod" class="dropdown-main">
    <ul>
        <li><a class="<%# Dopyty %>" href="/dopyty"><span>Dopyty</span></a></li>
        <li><a class="<%# CenovePonuky %>" href="/cenove-ponuky"><span>Cenové ponuky</span></a></li>
        <li><a class="<%# PrijateObjednavky %>" href="/prijate-objednavky"><span>Prijaté objednávky</span></a></li>
        <li><a class="<%# OdoslaneObjednavky %>" href="/odoslane-objednavky"><span>Odoslané objednávky</span></a></li>
        <li><a class="<%# DodacieListy %>" href="/dodacie-listy"><span>Dodacie listy</span></a></li>
    </ul>
</div>
```

Obrázok 29. Kód generujúci navigáciu užívateľského rozhrania

5.2 TwoColumns.Master

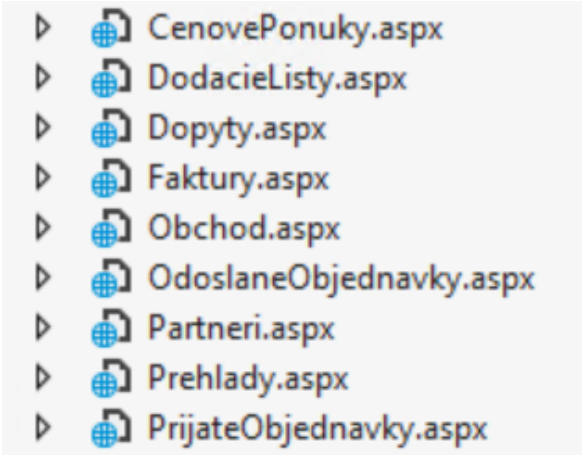
Už z nadpisu vyplýva že tento master bude vytvárať 2 stĺpce. Stĺpce predstavujú dve oddelené prostredia v užívateľskom rozhraní, do ktorých budú následne implementované funkcie užívateľom zvoleného webového formuláru. Ľavý stĺpec na obrázku 30 obsahuje vyhľadávacie pole a všetky obchodné dokumenty uložené v databáze. Užívateľ si môže zvoliť konkrétny doklad, ktorý sa následne zobrazí v pravom stĺpci.



Obrázok 30. Podklad webových formulárov

5.3 Webové formuláre

Zvyšné časti užívateľského rozhrania sú vytvorené v súboroch s príponou ASPX, tieto webové formuláre na obrázku 31 vykresľujú užívateľom zvolené doklady, ktoré sú uložené v databáze. V tomto prípade sú prvky jazyka HTML závislé na časti aplikácie, v ktorej sa užívateľ momentálne nachádza. Každý užívateľom zvolený prípad je generovaný vlastným webovým formulárom.



Obrázok 31. Stránky webových formulárov

Štruktúra formulárov je vytvorená v jazyku HTML s mapovaním na triedy jazyka CSS, rovnako ako aj pri mastroch.

6 ČÍTANIE DÁT Z DATABÁZY

Proces čítania dát z databázy je neodmysliteľnou súčasťou pri spustení aplikácie, ale aj následne počas behu aplikácie. Často je nevyhnutné čítať dáta z viacerých tabuliek databázy súčasne. V tejto kapitole opíšem spôsob, akým aplikácia načítava dáta z databázy pri prvom spustení.

Po spustení aplikácie sa v objektových triedach spustí metóda Page_Load, ktorá načíta v ľavom stĺpci zoznam všetkých dokladov rovnakého druhu, kde sú zobrazené iba základné informácie. V pravom stĺpci sa načítajú detaily dokladu, ktorý je v databáze uložený na poslednom mieste. Na obrázku 32 je znázornená metóda Page_Load cenovej ponuky.

```
protected void Page_Load(object sender, EventArgs e)
{
    MovisWebApp.Pages.TwoColumns master = (MovisWebApp.Pages.TwoColumns)this.Master;
    MovisWebApp.Pages.WebApp topLevelMaster = (MovisWebApp.Pages.WebApp)master.Master;

    master.PdfMethod = "generujPdf()";

    topLevelMaster.ContentName = "Cenové ponuky";

    //
    int ponukaId = 0;

    //
    if (RouteData.Values.ContainsKey("ponukaId"))
    {
        ponukaId = Convert.ToInt32(RouteData.Values["ponukaId"]);
        //
        DbErrorOccurred = VytvorCenovuPonuku(ponukaId);
    }

    //
    if (!DbErrorOccurred)
    {
        if ((this.ZoznamPonuk = NacitajPonuky()) == null)
        {
            DbErrorOccurred = true;
        }
        else
        {
            if (ponukaId == 0 && this.ZoznamPonuk.Count > 0)
            {
                ponukaId = this.ZoznamPonuk.First<PolozkaZoznamView>().Id;
            }
        }
    }

    //
    if (!DbErrorOccurred)
    {
        if (this.ZoznamPonuk.Count > 0)
        {
            this.ZvolenaPonuka = NacitajSuvisiace(ponukaId);
            if (this.ZvolenaPonuka == null)
                this.ZoznamPonuk = null;
        }
    }
}
```

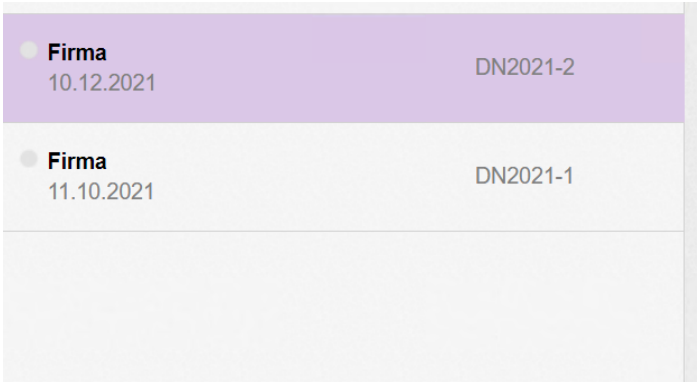
Obrázok 32. Metóda Page_Load

6.1 Načítanie zoznamu všetkých dokladov

Načítanie dát z databázy na obrázku 33 je realizované pomocou objektu triedy SqlConnection, ktorý následne dokáže čítať dáta z databázy pomocou triedy SqlDataReader.

```
List<PolozkaZoznamView> result = null;

string connectionString = ConfigurationManager.ConnectionStrings["MovisContext"].ConnectionString;
using (SqlConnection connection = new SqlConnection(connectionString))
{
    using (SqlCommand cmd = new SqlCommand("SELECT [Id],[Vytvorena],[Oznacenie] FROM [dbo].[CPZoznamView] ORDER BY [Id] DESC;", connection))
    {
        try
        {
            connection.Open();

            using (SqlDataReader reader = cmd.ExecuteReader())
            {
                result = new List<PolozkaZoznamView>();

                while (reader.Read())
                {
                    result.Add(new PolozkaZoznamView
                    {
                        Id = reader.GetInt32(0),
                        Datum = reader.GetDateTime(1),
                        Oznacenie = reader.GetString(2),
                        Poznamka = reader.IsDBNull(3) ? string.Empty : reader.GetString(3),
                        NazovZakaznika = reader.IsDBNull(4) ? string.Empty : reader.GetString(4)
                    });
                }
            }
        }
        catch (Exception) { result = null; }
    }
}
```

Obrázok 33. Načítanie dát z databázy



Obrázok 34. Zoznam dokladov z užívateľského rozhrania

Užívateľ si následne môže na obrázku 34 zobraziť detaily aj ďalších dokladov, avšak k tomu je nutná aktivita aj dynamického obsahu aplikácie.

6.2 Načítanie posledného dokladu

Načítanie detailov posledného dokladu funguje na rovnakom princípe, ako aj načítanie zoznamu všetkých dokladov, avšak vyžaduje vytiahnutie dát z viacerých entít a pohľadov

databázy. Obrázok 35 znázorňuje príklad vyťahovania dát z viacerých entít a pohľadov databázy.

```
CPView result = null;

string connectionString = ConfigurationManager.ConnectionStrings["MovisContext"].ConnectionString;
using (SqlConnection connection = new SqlConnection(connectionString))
{
    string queryString = "SELECT [Id],[Oznacenie],[DatumVystavenia],[PlatiDo] FROM [dbo].[CPView] WHERE [Id]=@ponukaId;";
    queryString += "SELECT [ZakaznikId],[Nazov],[KontaktnaOsoba] FROM [dbo].[CPZakaznikView] WHERE [CenovaPonukaId]=@ponukaId;";
    queryString += "SELECT [Id],[CenaZaKus],[PocetKusov],[MnozstevnaJednotka] FROM [dbo].[PolozkaCP] WHERE [CenovaPonukaId]=@ponukaId;";
    queryString += "SELECT [Oznacenie],[DokladId] FROM [dbo].[PrepojeneDokladyView] WHERE [PripadId]=@ponukaId ORDER BY [Oznacenie];";

    using (SqlCommand cmd = new SqlCommand(queryString, connection))
    {
        cmd.Parameters.AddWithValue("@ponukaId", ponukaId);

        try
        {
            connection.Open();

            using (SqlDataReader reader = cmd.ExecuteReader())
            {
                while (reader.Read())
                {
                    result = new CPView
                    {
                        Id = reader.GetInt32(0),
                        Oznacenie = reader.GetString(1),
                        DatumVystavenia = reader.GetDateTime(2),
                        PlatiDo = reader.GetDateTime(3),
                        Poznamka = reader.IsDBNull(4) ? null : (string)reader.GetValue(4),
                        UvodnyText = reader.IsDBNull(5) ? null : (string)reader.GetValue(5),
                        ZaverecnyText = reader.IsDBNull(6) ? null : (string)reader.GetValue(6),
                        Mena = reader.IsDBNull(7) ? null : (string)reader.GetValue(7),
                        MaObjednavku = reader.GetBoolean(8),
                        OznacenieDopytu = reader.GetString(9),
                    };
                }
                reader.NextResult();
```

Obrázok 35. Načítanie detailov dokladu pri spustení aplikácie

6.2.1 Výpočet cien

Pri načítaní dokladov prebieha na strane servera prepočet celkových cien za doklad. Celkové ceny nie sú uložené v databáze, výpočet prebieha iba v zmysle projekcie cien do užívateľského rozhrania. Najskôr je na obrázku 36 spočítaná cena za jednu položku bez DPH, resp. cena vynásobená počtom kusov. Následne je k cene pripočítaná hodnota DPH. Týmto postupom prejdú všetky položky.

```
public double? CenaSpoluBezDph
{
    get
    {
        if (_cenaZaKus != null && _dph != null)
        {
            return _cenaZaKus * _pocetKusov;
        }
        return null;
    }
}


public double? CenaSpoluDph
{
    get
    {
        var cenaSpoluBezDph = this.CenaSpoluBezDph;

        if (cenaSpoluBezDph != null)
        {
            return Math.Round(((double)(cenaSpoluBezDph * (_dph / 100 + 1))), 2, MidpointRounding.AwayFromZero);
        }
        return null;
    }
}
```

Obrázok 36. Výpočet ceny za jednu položku

Ceny jednotlivých položiek sú následne sčítané na obrázku 37 pre získanie celkovej sumy za doklad.

```
public string SpoluDph
{
    get
    {
        return this.Polozky.Where<PolozkaCenaView>(x => x.CenaSpoluDph != null).Select<PolozkaCenaView,
            double>(x => (double)x.CenaSpoluDph).Sum().ToFixedTwo();
    }

}
public string SpoluBezDph
{
    get
    {
        return this.Polozky.Where<PolozkaCenaView>(x => x.CenaSpoluBezDph != null).Select<PolozkaCenaView,
            double>(x => (double)x.CenaSpoluBezDph).Sum().ToFixedTwo();
    }
```

Obrázok 37. Výpočet celkovej sumy za doklad

7 DYNAMICKÝ OBSAH APLIKÁCIE

Asi najdôležitejšou úlohou dynamickej časti aplikácie je obsluhovať užívateľské požiadavky. V tejto kapitole bude vysvetlené, akým spôsobom sú tieto udalosti zachytávané a následne odosielané na spracovanie. Dynamika aplikácie má taktiež za úlohu validovať užívateľské vstupy, realizovať niektoré operácie a výpočty na strane klienta a dynamicky prispôsobovať užívateľské rozhranie potrebám užívateľa, počas behu aplikácie.

7.1 Vytváranie dokladov

Ako už bolo spomenuté v kapitole 2.1, vytvárajú sa iba dopyty, ostatné doklady sú generované z dokladov nižšej vrstvy.

7.1.1 Vytvorenie dopytu

Vytvoriť je možné iba prázdny dopyt, ostatné dáta ako napr. informácie o zákazníkovi alebo položky, je možné vkladať do dopytov až po vytvorení prázdneho dopytu formou úpravy dokladov.

Užívateľ klikne na tlačidlo „nový dopyt“. Túto aktivitu na obrázku 38 automaticky deteguje dynamická funkcionalita aplikácie, ktorá odošle asynchrónnu HTTP požiadavku na server. Controller na serveri vytvorí “prázdny“ dopyt, následne odošle klientovi odpoveď spolu so základnými dátami, ktoré vznikli pri vytváraní nového dopytu priamo na servery, alebo v databáze.

```
$(".nova-polozka").on("click", function (e) {
    $.ajax({
        headers: {
            Accept: "application/json"
        },
        url: "/api/dopyty/novy",
        type: "POST",
        success: function (data) {
            data = JSON.parse(data);
            $(".no-document").hide();
            $(".error-notif").hide();
            $(".document-placeholder").show();
            //
            var zoznam = $(".simplebar-content");
            var polozka = $(zoznam).children().first();

            if ($(polozka).css('display') === 'none') {
                $(polozka).css('display', 'block');
            } else {
                polozka = $(polozka).clone(true);
                $(zoznam).prepend(polozka);
            }
            //
```

Obrázok 38. Požiadavka na vytvorenie nového dopytu

Na záver zvyšná časť tela tejto funkcie na obrázku 39 vyplní formulár dopytu v užívateľskom rozhraní základnými dátami, tieto dáta boli obsiahnuté v odpovedi zo strany servera.

```
$(polozka).find(".valZakaznik").text("<Neznámy zákazník!>");
$(polozka).find(".valZakaznik").removeClass("nazov-zakaznika").addClass("warn");
$(polozka).find(".valOznacenieDopytu").text(data.OznacenieDopytu);
$(polozka).find(".valDatum").text(data.Prijaty);
$(polozka).find(".valId").val(data.Id);
//
vyplnFormular(data);
nastavTriedu(polozka);
zvolenyDopytId = data.Id;
window.history.pushState({ 'dopytId': zvolenyDopytId }, null, "/dopyty/" + zvolenyDopytId);
```

Obrázok 39. Vyplnenie dopytu v užívateľskom rozhraní

7.2 Úprava dokladov

Modifikácia dát v dokladoch sa vykonáva v užívateľskom rozhraní pochopiteľne počas behu aplikácie, je teda nevyhnutné zapojiť do tejto aktivity dynamickú stránku aplikácie. Ukladanie samotných dát prebieha automaticky potom, ako užívateľ opustí textové pole.

7.2.1 Funkcia „uprav doklad“

Ak niektoré vstupné pole užívateľského rozhrania stratí focus, resp. užívateľ ho opustí, príkaz javascriptu na obrázku 40 zistí, či bolo pred tým upravené. V prípade zistenej zmeny zavolá funkciu, ktorá sprostredkuje prepis dát v databáze formou HTTP požiadavky na controller.

```
$('[data-autosave="text"], .mena-select').on("change", upravPonuku);
```

Obrázok 40. Automatické ukladanie vstupných dát

Funkcia na obrázku 41 zistí, ktoré textové pole bolo upravené.

```
function upravPonuku() {
    var formData = new FormData();
    var inputname = $(this).data("inputname");
    formData.append(inputname, $(this).val());

    if (inputname === "nazovPolozky" || inputname === "popisPolozky" || inputname === "pocetKusov"
        || inputname === "cenaZaKus" || inputname === "dph" || inputname === "mnozstevnaJednotka") {
        var riadok;
        if (inputname === "popisPolozky") {
            riadok = $(this).closest("tr").prev();
        } else {
            riadok = $(this).closest("tr");
        }

        formData.append("polozkaid", $(riadok).attr("data-polozkaid"));
    }
```

Obrázok 41. Identifikácia upraveného vstupu

Ak bolo dané pole upravené, funkcia na obrázku 42 odošle HTTP požiadavku na server, kde ju následne spracuje controller pre daný doklad.

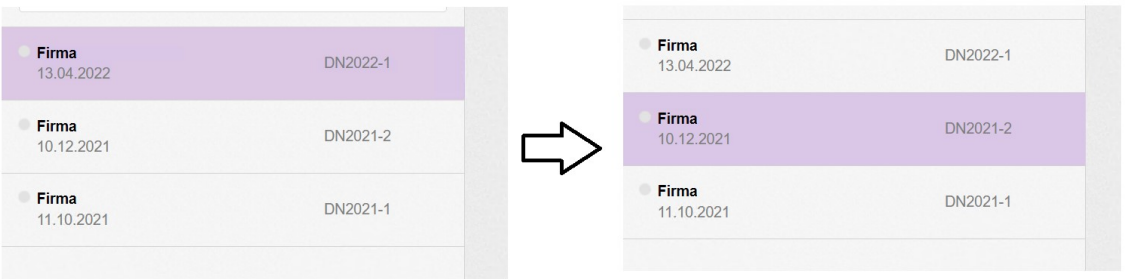
```
$.ajax({
    headers: {
        Accept: "application/json",
    },
    url: "/api/cenove-ponuky/update/" + zvolenaPonukaId,
    type: "POST",
    contentType: false,
    processData: false,
    data: formData,
    success: function (data) {
        var parentElement;
        var element;

        if (typeof data.Poznamka !== 'undefined') {
            parentElement = $("input[value='" + data.Id + "']").prev();
            element = $(parentElement).find(".valPoznamka");
            if (data.Poznamka.length > 0) {
                $(element).text(data.Poznamka);
                $(element).closest("tr").css('display', 'table-row');
            } else {
                $(element).text("");
                $(element).closest("tr").css('display', 'none');
            }
        }

        //
        $('.total-mena').text($('.mena-select').val().toUpperCase());
    },
    error: function () {
    }
});
```

Obrázok 42. Odoslanie požiadavky na server

7.3 Načítanie zvoleného dokladu

Aby si užívateľ mohol zobraziť detaily zvoleného dokladu, teda iného dokladu ako posledného v databáze, ktorý sa zobrazí automaticky pri načítaní webového formuláru, je taktiež nevyhnutné, aby dynamická stránka aplikácie odoslala HTTP požiadavku na server.

Užívateľ si môže v ľavom stĺpci na obrázku 43 zvoliť ľubovoľný doklad rovnakého druhu pre zobrazenie detailov.



Obrázok 43. Zvolenie požadovaného dokladu

Funkcia na obrázku 44 následne zachytí užívateľskú aktivitu v ľavom stĺpci, resp. požiadavku na zobrazenie detailov iného dokladu, pričom zistí ID požadovaného dokladu.

```
$(".polozkazoznamu-obal").on("click", function (e) {
    zvolenaPonukaId = $(this).children(".valId").val();
    //
    nastavTriedu(this);
    //
    nacitajPonuku();
    //
    window.history.pushState({ 'ponukaId': zvolenaPonukaId }, null, "/cenove-ponuky/" + zvolenaPonukaId);
});
```

Obrázok 44. Identifikácia užívateľskej aktivity

Následne zavolá funkciu na obrázku 45, ktorá odošle HTTP požiadavku na server, resp. na controller.

```
function nacitajPonuku() {
    $.ajax({
        headers: {
            Accept: "application/json"
        },
        url: "/api/cenove-ponuky/" + zvolenaPonukaId,
        type: "GET",
        success: function (data) {
            data = JSON.parse(data);
            $(".error-notif").hide();
            $(".document-placeholder").show();
            vyplnFormular(data);
        },
        error: function () {
            $(".document-placeholder").hide();
            $(".error-notif").show();
        }
    });
}
```

Obrázok 45. Odoslanie požiadavky na server

Controller odpovie na HTTP požiadavku, pričom odpoveď obsahuje požadované dáta, následne funkcia javascriptu na obrázku 46 vyplní formulár v užívateľskom rozhraní.

```
function vyplnFormular(data) {
    $(".oznacenie-objednavky").text(data.Oznacenie);
    $('[data-inputname="poznamka"]').val(data.Poznamka);
    if (data.Poznamka.length === 0) {
        $('[data-inputname="poznamka"]').removeClass("poznamka-vyplnena");
    } else {
        $('[data-inputname="poznamka"]').addClass("poznamka-vyplnena");
    }
    $('.po-btn').attr("data-maobjednavku", data.MaObjednavku);
    $('[data-inputname="vystavena"]').val(data.DatumVystavenia);
    $('[data-inputname="plati"]').val(data.PlatiDo);
    $('[data-inputname="cislo-dopytu"]').val(data.OznacenieDopytu);


    if (!data.Zakaznik) {
        $('[data-inputname="nazovFirmy"]').val("");
        $('[data-inputname="kontaktnaOsoba"]').val("");
        $('[data-inputname="ulica"]').val("");
        $('[data-inputname="psc"]').val("");
        $('[data-inputname="mesto"]').val("");
        $('[data-inputname="stat"]').val("");
        $('[data-inputname="ico"]').val("");
        $('[data-inputname="dic"]').val("");
        $('[data-inputname="icdph"]').val("");
        $('[data-zakaznikid]').attr('data-zakaznikid', "");
    } else {
        $('[data-inputname="nazovFirmy"]').val(data.Zakaznik.Nazov);
        $('[data-inputname="kontaktnaOsoba"]').val(data.Zakaznik.KontaktnaOsoba);
        $('[data-inputname="ulica"]').val(data.Zakaznik.Ulica);
        $('[data-inputname="psc"]').val(data.Zakaznik.Psc);
```

Obrázok 46. Vyplnenie formuláru v užívateľskom rozhraní

7.4 Pridávanie položiek do odoslanej objednávky

Ako už bolo spomenuté v kapitole 2.2, položky odoslanej objednávky sa importujú z položiek a podpoložiek prijatej objednávky. Užívateľ klikne na tlačidlo „pridaj položku“, funkcia javascriptu na obrázku 47 deteguje túto aktivitu a vytvorí webový formulár s možnosťou vyhľadávania prijatých objednávok.

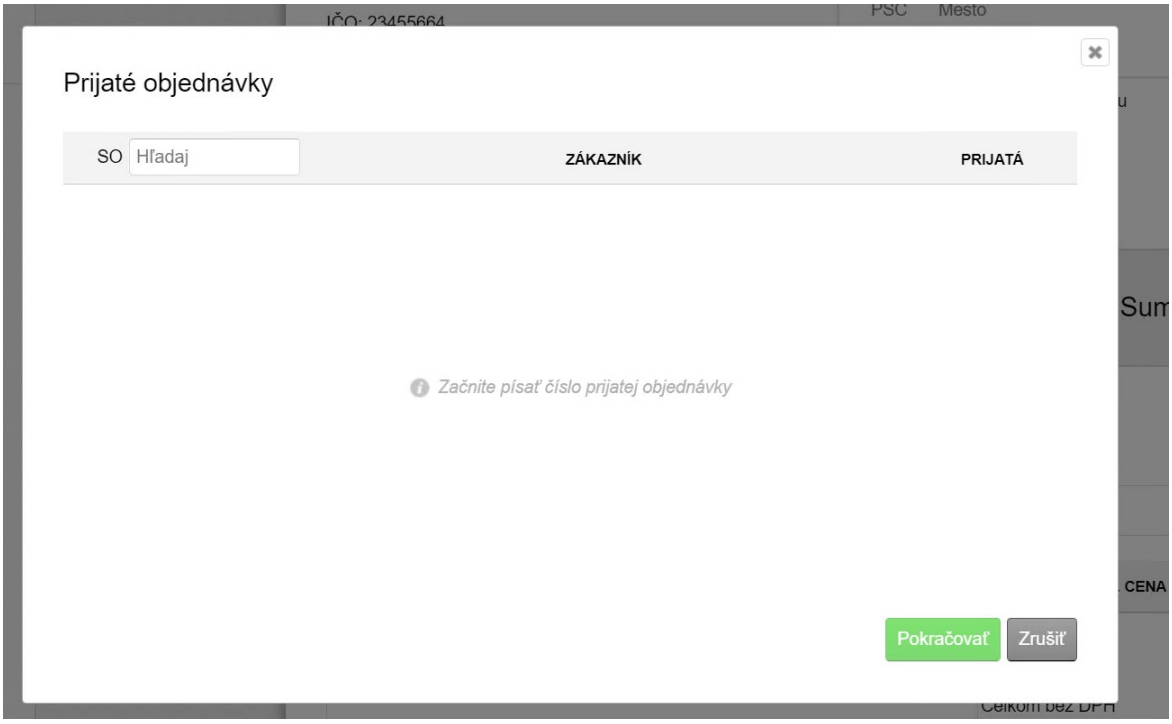
```
$(".np-btn").on("click", function (e) {
    var tpl = $('.riadok-dostupne').first();
    var parent = $(tpl).parent();
    $(parent).children().not(':first-child').remove();
    //
    $(".vyhladavanie").val("");
    $("#prehlad-dostupneobj .continuebutton").prop("disabled", true);
    //
    $('.filter-info').css("display", "flex");
    $('.filter-info').find('span').text("Začnite písať číslo prijatej objednávky");
    //
    //
```

Obrázok 47. Vytvorenie formuláru pre vyhľadanie prijatej objednávky

Vytvorený formulár na obrázku 49 je následne uložený na stred obrazovky, pomocou kódu na obrázku 48.

```
var x = $(window).width();
x = x - $('#draggable1').width();
x = x / 2;
$('#prehlad-dostupneobj').fadeIn(150);
var y = $(window).height();
y = y - $('#draggable1').height();
y = y / 2;
$('#draggable1').css('top', y + "px");
$('#draggable1').css('left', x + "px");
```

Obrázok 48. Uloženie formuláru na stred obrazovky



Obrázok 49. Výstup funkcie javascriptu (Obrázok 35-36)

Užívateľ začne písať externé označenie prijatej objednávky od tretieho znaku, pretože prvé dva znaky majú všetky prijaté objednávky rovnaké, počas toho funkcia javascriptu na obrázku 50 zachytí tento vyhľadávaný výraz a odošle HTTP požiadavku na controller pre vrátenie vyhovujúcich prijatých objednávok. Užívateľ nemusí zadať celé externé označenie prijatej objednávky, požiadavka smeruje na controller automaticky, ako náhle vyhľadávacie pole obsahuje aspoň jeden znak, pričom užívateľovi vracia vyhovujúce prijaté objednávky. Tento cyklus sa zopakuje vždy, keď sa zmení reťazec znakov vo vyhľadávacom poli.

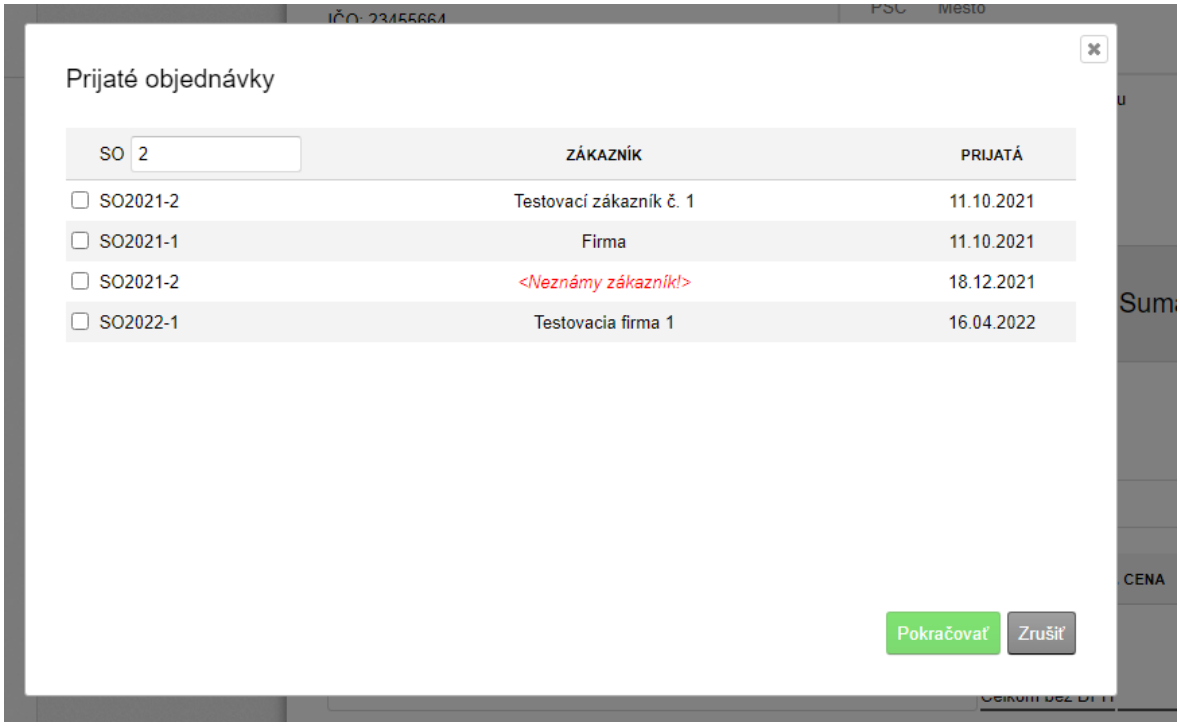
```
$('.vyhladavanie').keyup(delay(function (e) {
    $('.riadok-dostupne').not(':first-child').remove();
    $("#prehlad-dostupneobj .continuebutton").prop("disabled", true);
    $('#prehlad-dostupneobj').data("objednavkaid", "");
    $('.filter-info').css("display", "flex");

    var exp = $(".vyhladavanie").val();
    //
    if (exp.length !== 0) {
        $('.filter-info').find('span').text("Vyhľadávam ...");
        //
        var formData = new FormData();
        formData.append("vyraz", exp);
        $.ajax({
            headers: {
                Accept: "application/json"
            },
            url: "/api/odoslane-objednavky/zoznampo",
            type: "POST",
            contentType: false,
            processData: false,
            data: formData,
            success: function (data) {
                zobrazZoznamPO(data);
            },
            error: function () {
            }
        });
    } else {
        $('.filter-info').find('span').text("Začnite písať číslo prijatej objednávky");
    }
}, 500));
```

Obrázok 50. Funkcia javascriptu pre vyhľadávanie dostupných objednávok

Súčasne pri každej zmene reťazca znakov vo vyhľadávacom poli funkcia javascriptu na obrázku 51 vypisuje vyhovujúce prijaté objednávky do užívateľského rozhrania.

```
function zobrazZoznamPO(data) {
    //
    var tpl = $('.riadok-dostupne').first();
    var parent = $(tpl).parent();

    //
    $.each(data, function (index, value) {
        var polozka = $(tpl).clone(true);
        $(polozka).find('.dr-oznacenie').text(value.Oznacenie);
        if (!value.Zakaznik) {
            $(polozka).find('.dr-zakaznik').text("<Neznámy zákazník!>").css("font-style", "italic").css("color", "red");
        } else {
            $(polozka).find('.dr-zakaznik').text(value.Zakaznik);
        }
        $(polozka).find('.dr-prijata').text(value.Prijata);
        $(polozka).data("objednavkaid", value.Id);
        $(polozka).css("display", "flex");
        $(parent).append(polozka);
    });

    if (data.length === 0) {
        $('.filter-info').css("display", "flex");
        $('.filter-info').find('span').text("Zvolenému filtru nezodpovedajú žiadne položky");
    } else {
        $('.filter-info').css("display", "none");
    }
}
```

Obrázok 51. Funkcia na zobrazenie vyhovujúcich prijatých objednávok



Obrázok 52. Výstup v užívateľskom rozhraní

Užívateľ následne vyberie požadovanú prijatú objednávku na obrázku 52 kliknutím na check box. Funkcia javascriptu na obrázku 53 uvoľní tlačidlo „pokračovať“, zistí ID prijatej objednávky, ktorej patrí zvolený check box. Funkcia taktiež zabezpečí, aby mohla byť

zvolená iba jedna prijatá objednávka, ak je označený jeden z check boxov a užívateľ klikne na druhý, pôvodný check box stratí označenie.

```
$(".dr-chk").on("click", function (e) {
    if ($(this).is(":checked")) {
        $('.dr-chk').not(this).prop('checked', false);
        $("#prehlad-dostupneobj .continuebutton").prop("disabled", false);
        //
        var objednavkaid = $(this).closest('.riadok-dostupne').data("objednavkaid");
        $('#prehlad-dostupneobj').data("objednavkaid", objednavkaid);
    } else {
        $("#prehlad-dostupneobj .continuebutton").prop("disabled", true);
    }
    e.stopPropagation();
});
```

Obrázok 53. Výber prijatej objednávky

Ak užívateľ klikne na tlačidlo pokračovať, javascript na obrázku 54 odošle HTTP požiadavku na controller, pre vrátenie objednávaných položiek, resp. položiek a podpoložiek zvolenej prijatej objednávky.

```
$('#prehlad-dostupneobj .continuebutton').on("click", function (e) {
    var formData = new FormData();
    formData.append("poid", $('#prehlad-dostupneobj').data("objednavkaid"));
    formData.append("ooid", zvolenaObjednavkaId);

    $.ajax({
        headers: {
            Accept: "application/json",
        },
        url: "/api/odoslane-objednavky/zoznamop",
        type: "POST",
        contentType: false,
        processData: false,
        data: formData,
        success: function (data) {
            zobrazZoznamOP(data);
        },
        error: function () {
        }
    });
});
```

Obrázok 54. HTTP požiadavka pre vrátenie objednávaných položiek

Následne sa vypíšu všetky položky a podpoložky zvolenej prijatej objednávky v užívateľskom rozhraní na obrázku 7. Túto aktivitu sprostredkuje funkcia na obrázku 55.

```
function zobrazZoznamOP(data) {
    $("#prehlad-dostupneobj").fadeOut(150, function () {
        var tpl = $('.riadok-dp').first();
        $('.riadok-dp').not(tpl).remove();
        //
        $.each(data.Polozky, function (index, value) {
            var polozka = $(tpl).clone(true).css("display", "flex");
            $(polozka).data("polozkaid", value.PolozkaId);
            $(polozka).find('.dp-nazov').val(value.Nazov);
            $(polozka).find('.dp-pocet').val(value.PocetKusov);
            $(polozka).insertBefore("#prehlad-dostupnepolozky .dr-control-panel");

            $.each(value.PodradenePolozky, function (idx, podradenaPolozka) {
                polozka = $(tpl).clone(true).css("display", "flex");
                $(polozka).css("padding-left", "30px");
                $(polozka).data("polozkaid", podradenaPolozka.PolozkaId);
                $(polozka).find('.dp-nazov').val(podradenaPolozka.Nazov);
                $(polozka).find('.dp-pocet').val(podradenaPolozka.PocetKusov);
                $(polozka).insertBefore("#prehlad-dostupnepolozky .dr-control-panel");
            });
        });
        $(".oznaceniepo").text(data.Oznacenie);
        $('.dp-chkall').prop('checked', false);
        $("#prehlad-dostupnepolozky .continuebutton").prop("disabled", true);
        //
```

Obrázok 55. Výpis položiek a podpoložiek zvolenej prijatej objednávky

Následne užívateľ zvolí požadované položky a podpoložky kliknutím na check boxy. V tomto prípade už môže zvoliť viacero možností. Túto aktivitu deteguje a zabezpečuje funkcia na obrázku 56.

```
$(".dp-chk").on("click", function (e) {
    if ($('.dp-chk:checked').length === 0) {
        $('.dp-chkall').prop('checked', false);
        $("#prehlad-dostupnepolozky .continuebutton").prop("disabled", true);
    } else {
        $("#prehlad-dostupnepolozky .continuebutton").prop("disabled", false);
        if ($('.dp-chk:checked').length === ($('.dp-chk').length - 1)) {
            $('.dp-chkall').prop('checked', true);
        }
    }
});
```

Obrázok 56. Výber objednávaných položiek

Prípadne jedným kliknutí vybrať všetky položky aj podpoložky. Tento princíp je definovaný na obrázku 57.

```
$('.dp-chkall').change(function () {
    if (this.checked) {
        $('.riadok-dp').slice(1).each(function (index, element) {
            $(this).find('.dp-chk').prop('checked', true);
        });
        //
        if ($('.dp-chk:checked').length > 0) {
            $("#prehlad-dostupnepolozky .continuebutton").prop("disabled", false);
        } else {
            $("#prehlad-dostupnepolozky .continuebutton").prop("disabled", true);
        }
    } else {
        $('.dp-chk').prop('checked', false);
        $("#prehlad-dostupnepolozky .continuebutton").prop("disabled", true);
    }
});
```

Obrázok 57. Voľba všetkých položiek a podpoložiek

V ďalšom kroku sú všetky zvolené položky importované do jedného textového reťazca v tvare ID položky 1, počet kusov; ID položky 2, počet kusov; a tento reťazec následne smeruje na controller, ktorý tieto položky zapíše do databázy pomocou funkcie na obrázkoch 58 a 59.

```
$('#prehlad-dostupnepolozky .continuebutton').on("click", function (e) {
    $(this).prop("disabled", true);
    //
    var formData = new FormData();
    //
    var str = "";
    $('.riadok-dp').slice(1).each(function (index, element) {
        if ($(this).find('.dp-chk').is(":checked")) {
            if (str.length !== 0) {
                str = str + ";";
            }
            //
            var polozkaid = $(element).data("polozkaid");
            if (typeof polozkaid !== 'undefined') {
                str = str + "polozkaid:" + polozkaid + ",pocet:" + $(this).find('.dp-pocet').val();
            }
        }
    });
```

Obrázok 58. Vytvorenie reťazca z položiek odoslanej objednávky

```
        $.ajax({
            headers: {
                Accept: "application/json"
            },
            url: "/api/odoslane-objednavky/novapolozka",
            type: "POST",
            contentType: false,
            processData: false,
            data: formData,
            success: function (data) {
                doplnPolozky(data);
            },
            error: function () {
                $("#prehlad-dostupnepolozky .continuebutton").prop("disabled", false);
            }
        });
    });
```

Obrázok 59. HTTP požiadavka pre zapísanie položiek do databázy

Ak všetko prebehlo v poriadku, položky sa vypíšu do odoslanej objednávky v užívateľskom rozhraní, pomocou funkcie na obrázku 60.

```
function doplnPolozky(data) {
    var tplPolozka = $('.riadok-oo').first();
    var tplPopis = $(tplPolozka).next();
    var tplPridaj = $(tplPolozka).parent().children().last();

    $.each(data, function (index, value) {
        var polozka = $(tplPolozka).clone(true).css("display", "table-row");
        $(polozka).attr("data-polozkaid", value.PolozkaId);
        $(polozka).find("input").first().val($('.riadok-oo').length);
        $(polozka).find('[data-inputname="nazovPolozky"]').val(value.Nazov);
        $(polozka).find('[data-inputname="pocetKusov"]').val(value.PocetKusov);
        $(polozka).find('[data-inputname="mnozstevnaJednotka"]').val(value.MnozstevnaJednotka);
        $(polozka).find('[data-inputname="cenaZaKus"]').val(value.CenaZaKusStr);
        $(polozka).find('[data-inputname="dph"]').val(value.DphStr);
        $(polozka).find('.valCenaSpolu').val(value.CenaSpoluDphStr);
        $(polozka).insertBefore(tplPridaj);
        var polozkaPopis = $(tplPopis).clone(true).css("display", "table-row");
        $(polozkaPopis).find('[data-inputname="popisPolozky"]').val(value.Popis);
        $(polozkaPopis).insertBefore(tplPridaj);
        var ta = $(polozkaPopis).find('[data-inputname="popisPolozky"]');
        autosize($(ta));
    });

    //
    spocitajCelkom();

    //
    $("#prehlad-dostupnepolozky").fadeOut(150);
}
```

Obrázok 60. Výpis položiek odoslanej objednávky

7.5 Generovanie PDF

Pri generovaní PDF má za úlohu javascript iba sledovať, kedy užívateľ klikne na tlačidlo pre vytvorenie PDF dokumentu a následne odošle na controller HTTP požiadavku, v ktorej je podstatné len ID dokumentu a URL adresa, znázornená na obrázku 61.

```
function generujPdf() {
    var pdfUrl = "/api/prijate-objednavky/pdf/" + zvolenaObjednavkaId;

    $("#pdfDownloader").attr("href", pdfUrl);
    $("#pdfDownloader")[0].click();
}
```

Obrázok 61. Odoslanie požiadavky na server

7.6 Validácia vstupov

Úlohou validácie vstupov je eliminovať nesprávne alebo nezmyselné vstupy, ktoré by v niektorých prípadoch mohli spôsobovať v aplikácií aj problémy, pretože niektoré vstupy sú následne využívané na aritmetické alebo logické operácie. Validácia týchto vstupov prebieha na strane užívateľa, priamo v jeho prehliadači. Vstupy sú porovnávané s regulárnym výrazom vo funkcii jazyka javascript, z dôvodu aby nesprávne vstupy zbytočne nezaťažovali server.

7.6.1 Číslicové vstupy

Aplikácia obsahuje vstupné polia aj výlučne číslicového charakteru, ktoré vypĺňa užívateľ. Tieto vstupy sú validované na strane klienta ale aj na strane servera. Napr. na obrázku 62, do vstupu počet kusov, nie je možné zadať iné znaky ako číslice 0-9, pričom prvá číslica nesmie byť 0.

```
$('[data-inputname="pocetKusov"]').inputFilter(function (value) {
    return (/^[1-9][0-9]*$/).test(value);
});
```

Regulárny výraz

Obrázok 62. Validácia číslicových vstupov na strane klienta

7.6.2 Prázdne vstupy

Funkcia na obrázku 63 skontroluje vstup a zistí, či nie je prázdny, alebo neobsahuje len medzery.

```
String.prototype.isEmptyOrWhiteSpace = function () {
    if (this.length !== 0) {
        var str = this.replace(/^\s+|\s+$/gm, '');
        if (str.length === 0) {
            return true;
        } else {
            return false;
        }
    } else {
        return true;
    }
};
```

Obrázok 63. Eliminácia prázdnych vstupov

7.7 Prepočet cien

Ak užívateľ v dokladoch pridá novú položku, ktorá predstavuje druh tovaru alebo služby, musí zadať množstvo, jednotkovú cenu a výšku DPH v %. Cena položky s DPH a celková cena za doklad sú vypočítané na strane servera, ale aj na strane klienta, aby sa zmeny automaticky zobrazili po každej úprave, nie len pri načítaní dokladu. Príklad je znázornený na obrázku 64.

Č.	NÁZOV POLOŽKY	POČET	MJ	JEDN. CENA	DPH %	SPOLU S DPH
1	Moja polozka c.1 dopytu	8	ks	45,00	20	432,00
	Popis polozky c.1 dopytu					
2	Moja polozka c.2 dopytu	6	kg	70,00	20	504,00
	Popis polozky c.2 dopytu					
3	Moja polozka c.3 dopytu	25	m	80,00	20	2 400,00
	Popis polozky c.3 dopytu					
4	Moja polozka c.4 dopytu	8	l	98,00	20	940,80
	Popis polozky c.4 dopytu					
5	Moja polozka c.5 dopytu	15	kg	87,00	20	1 566,00
	Popis polozky c.5 dopytu					

Môj záverečný text

Celkom bez DPH 4 869,00

Celkom **EUR 5 842,80**

Obrázok 64. Príklad oceňovania položiek

Ak užívateľ pridá novú položku s cenou, počtom kusov a DPH, funkcia v jazyku javascript automaticky vypočíta celkovú cenu za danú položku spolu s DPH a súčasne túto cenu pripočíta k celkovým cenám za doklad (s DPH a bez DPH).

Najskôr je vytvorená funkcia na obrázku 65, ktorá vypočíta cenu bez DPH pre jednu položku, resp. pre jeden riadok.

```
function vypocitajCenuBezDph(element) {
    var cenaZaKus = $(element).find('[data-inputname="cenaZaKus"]').val();
    cenaZaKus = cenaZaKus.replace(/,/g, '.');
    cenaZaKus = cenaZaKus.replace(/\s/g, '');
    cenaZaKus = parseFloat(cenaZaKus);
    var pocetKusov = parseInt($(element).find('[data-inputname="pocetKusov"]').val());
    if (!isNaN(cenaZaKus) && !isNaN(pocetKusov)) {
        return (pocetKusov * cenaZaKus);
    } else {
        return Number.NaN;
    }
}
```

Obrázok 65. Výpočet ceny bez DPH pre jednu položku

Následne vypočíta cenu s DPH pre jeden riadok, podľa funkcie na obrázku 66 a upraví ju na požadovaný tvar.

```
function prepocitajCenu() {
    var riadok = $(this).closest("tr");
    var cenaBezDph = vypocitajCenuBezDph(riadok);
    var dph = $(riadok).find('[data-inputname="dph"]').val();
    dph = dph.replace(/,/g, '.');
    dph = parseFloat(dph);
    if (!isNaN(cenaBezDph) && !isNaN(dph)) {
        dph = dph / 100 + 1;
        var temp = cenaBezDph * dph;
        var cenaSpolu = Math.round((temp + Number.EPSILON) * 100) / 100;
        cenaSpolu = cenaSpolu.toFixed(2);
        cenaSpolu = cenaSpolu.replace(/(\d)(?=(\d{3})+\.)/g, '$1 ');
        cenaSpolu = cenaSpolu.replace(/\./g, ',');
        $(riadok).find('.valCenaSpolu').val(cenaSpolu);
    } else {
        $(riadok).find('.valCenaSpolu').val("");
    }

    //
```

Obrázok 66. Výpočet ceny s DPH pre jednu položku

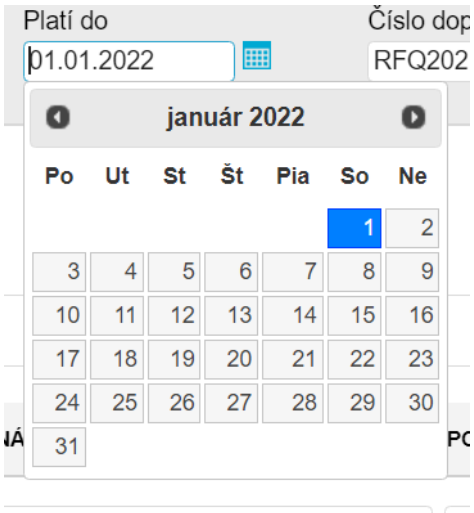
Následne riadok pričíta k celkovej cene za doklad, čím vypočíta cenu za celý doklad s DPH a bez DPH, v zmysle kódu na obrázku 67.

```
//
var total = 0;
var totalBezDph = 0;
$('.valCenaSpolu').each(function (index, element) {
    var cenaSpolu = $(this).val().replace(/,/g, '.');
    cenaSpolu = cenaSpolu.replace(/\s/g, '');
    cenaSpolu = parseFloat(cenaSpolu);
    if (!isNaN(cenaSpolu)) {
        total = total + cenaSpolu;
    }
    cenaBezDph = vypocitajCenuBezDph($(this).closest("tr"));
    if (!isNaN(cenaBezDph)) {
        totalBezDph = totalBezDph + cenaBezDph;
    }
});
total = total.toFixed(2);
total = total.replace(/(\d)(?=(\d{3})+\.)/g, '$1 ');
total = total.replace(/\./g, ',');
$('#total1').text(total);
$('#total2').text(total);
//
totalBezDph = totalBezDph.toFixed(2);
totalBezDph = totalBezDph.replace(/(\d)(?=(\d{3})+\.)/g, '$1 ');
totalBezDph = totalBezDph.replace(/\./g, ',');
$('#total3').text(totalBezDph);
```

Obrázok 67. Výpočet ceny s DPH a bez DPH

7.8 Datepicker

Datepicker je mini aplikácia knižnice jQuery, ktorá umožňuje užívateľovi vypĺňať „dátumové vstupy“ vizuálnou formou. Užívateľ si jednoduchým kliknutím zvolí dátum, ktorý táto mini aplikácia vloží do vybraného vstupu. Príklad datepickeru je znázornený na obrázku 68.



Obrázok 68. Datepicker v užívateľskom rozhraní

Celá mini aplikácia datepickeru je obsiahnutá v knižnici jQuery, programátor musí len implementovať túto mini aplikáciu do svojho kódu a nastaviť užívateľské rozhranie do vhodného tvaru, ako na obrázku 69. Na internete je dokonca možné nájsť rôzne verzie nastavenia tohto rozhrania v rôznych jazykoch.

```
}(function (datepicker) {

    datepicker.regional.sk = {
        closeText: "Zavrieť",
        prevText: "&#x3C;Predchádzajúci",
        nextText: "Nasledujúci&#x3E;",
        currentText: "Dnes",
        monthNames: ["január", "február", "marec", "apríl", "máj", "jún",
        "júl", "august", "september", "október", "november", "december"],
        monthNamesShort: ["Jan", "Feb", "Mar", "Apr", "Máj", "Jún",
        "Júl", "Aug", "Sep", "Okt", "Nov", "Dec"],
        dayNames: ["nedeľa", "pondelok", "utorok", "streda", "štvrtok", "piatok", "sobota"],
        dayNamesShort: ["Ned", "Pon", "Uto", "Str", "Štv", "Pia", "Sob"],
        dayNamesMin: ["Ne", "Po", "Ut", "St", "Št", "Pia", "So"],
        weekHeader: "Ty",
        dateFormat: "dd.mm.yy",
        firstDay: 1,
        isRTL: false,
        showMonthAfterYear: false,
        yearSuffix: ""
    };
    datepicker.setDefaults(datepicker.regional.sk);

    return datepicker.regional.sk;
```

Obrázok 69. Nastavenie užívateľského rozhrania datepickeru

8 COTROLLER

V tejto práci je úlohou Controllerov obsluhovať užívateľské požiadavky, ktoré musia byť spracované na strane servera formou Web API. Controller príjme HTTP požiadavku, ktorá je reprezentovaná triedou HttpRequest, spracuje ju a odošle klientovi objekt triedy HttpResposneMessage reprezentujúci HTTP odpoveď. Spracovanie požiadavky môže znamenať vkladanie dát do databázy, zobrazenie dát z databázy a pod. Každý druh dokladov má svoj vlastný Controller.

Controller na obrázku 70 v prvom kroku zistí, s ktorým dokladom bude pracovať podľa ID. Toto ID je súčasťou požiadavky.

```
[Route("api/cenove-ponuky/update/{ponukaId}")]
[HttpPost]
public HttpResponseMessage UpravPonuku(int ponukaId)
{
```

Obrázok 70. Hlavička triedy HttpResponseMessage

8.1 Vytvorenie dopytu

Všetky novovytvorené dopyty sú rovnaké, líšia sa len v ID, označení a samozrejme dátume. Controller na obrázku 71 zachytí HTTP požiadavku na vytvorenie nového dopytu, vygeneruje aktuálny dátum a odošle príkaz do databázy. Ako už bolo avizované v kapitole 7.1.1, databáza novému dopytu pridelí ID a označenie. Na záver Controller odošle odpoveď na požiadavku spolu s vygenerovanými dátami.

```
[Route("api/dopyty/novy")]
[HttpPost]
public HttpResponseMessage Novy()
{
    HttpResponseMessage response = null;

    using (IMovisContext context = new MovisContext())
    {
        try
        {
            Dopyt dopyt = new Dopyt()
            {
                Prijaty = DateTime.Now
            };
            context.Dopyty.Add(dopyt);
            context.Save();
            //
            DopytView result = new DopytView
            {
                Id = dopyt.Id,
                Oznacenie = dopyt.Oznacenie,
                Prijaty = dopyt.Prijaty
            };

            //
            response = Request.CreateResponse(HttpStatusCode.OK, result.ToJSON());
        }
        catch (Exception)
        {
            response = Request.CreateResponse(HttpStatusCode.BadRequest);
        }
    }

    return response;
}
```

Obrázok 71. Generovanie dopytu na strane servera

8.2 Upravenie dokladu

Do controllera prichádza HTTP požiadavka, ktorá nesie sadu kľúčov, identifikátor upravovaného vstupu, upravenú hodnotu a pod. Trieda HttpResponseMessage najskôr zistí, ktorý vstup bol upravený na novú hodnotu tohto vstupu. V prípade úpravy položiek musí zistiť, o ktorú položku sa jedná, a následne sa odkázať na danú položku. Príklad uvedeného spracovania HTTP požiadavky je znázornený na obrázku 72.

```
try
{
    foreach (string key in req.Form.AllKeys)
    {
        switch (key)
        {
            case "polozkaid":
                int polozkaid = Int32.Parse(req["polozkaid"]);
                polozka = context.PolozkyCP.Single<PolozkaCP>(x => x.Id == polozkaid);
                break;
            default:
                inputName = key;
                valueToUpdate = req[inputName];
                break;
        }
    }
```

Obrázok 72. Spracovanie kľúčov HTTP požiadavky

Následne už len controller na obrázku 73 prepíše hodnotu vstupu v databáze. V prípade číslicových vstupov porovná znovu daný reťazec s regulárnym výrazom, zároveň číslicové vstupy prichádzajú do controllera v typoch string, preto je nevyhnutné tieto vstupy pretypovať.

```
switch (inputName)
{
    case "nazovPolozky":
        polozka.Nazov = valueToUpdate.Trim() == string.Empty ? null : valueToUpdate.Trim();
        break;
    case "popisPolozky":
        polozka.Popis = valueToUpdate.Trim() == string.Empty ? null : valueToUpdate.Trim();
        break;
    case "uvodny-text":
        context.CenovePonuky.Single<CenovaPonuka>(x => x.Id == ponukaId).UvodnyText =
            valueToUpdate.Trim() == string.Empty ? null : valueToUpdate.Trim();
        break;
    case "zaverecny-text":
        context.CenovePonuky.Single<CenovaPonuka>(x => x.Id == ponukaId).ZaverecnyText =
            valueToUpdate.Trim() == string.Empty ? null : valueToUpdate.Trim();
        break;
    case "pocetKusov":
        Regex regexPocet = new Regex(@"^[1-9][0-9]*$");
        if (regexPocet.IsMatch(valueToUpdate.Trim()))
        {
            polozka.PocetKusov = Int32.Parse(valueToUpdate.Trim(), CultureInfo.InvariantCulture);
        }
        else
        {
            throw new Exception("Počet kusov nie je v správnom tvare!");
        }
        break;
    case "cenaZaKus":
        Regex regexCena = new Regex(@"^-?(\d+[\.,]?\d{0,2})?$");

        if (regexCena.IsMatch(valueToUpdate))
        {
            valueToUpdate = valueToUpdate.Replace(",", ".");
            valueToUpdate = valueToUpdate.Replace(" ", "");
            if (!string.IsNullOrEmpty(valueToUpdate))
            {
                double cenaZaKus = Convert.ToDouble(valueToUpdate, CultureInfo.InvariantCulture);
                polozka.CenaZaKus = cenaZaKus;
            }
            else
            {
                polozka.CenaZaKus = null;
            }
        }
        else {
            throw new Exception("Cena za kus nie je v správnom tvare!");
        }
        break;
```

Obrázok 73. Prepis dát v databáze

8.3 Načítanie zvoleného dokladu

Pri načítaní zvoleného dokladu controller zaujíma iba ID tohto dokladu, nakoľko pri tejto akcii nie sú užívateľom vykonávané žiadne zmeny, modifikované vstupy a pod. Načítanie

prebieha rovnakým spôsobom, ako pri spustení aplikácie, načítanie vlastností dokladu uloženého v databáze na poslednom mieste spolu s prepočtom celkových cien za celý doklad.

8.4 Vyhľadávanie prijatých objednávok

Controller odoslaných objednávok na obrázku 74 príjme HTTP požiadavku s priloženým vyhľadávaným výrazom, pomocou ktorého vytvorí dotaz v jazyku SQL. Tento dotaz vracia všetky vyhovujúce prijaté objednávky, ktoré sú uložené v databáze.

```
public HttpResponseMessage NacitajZoznamPO()
{
    HttpResponseMessage response = null;

    string vyraz = HttpContext.Current.Request["vyraz"];
    vyraz = "SO" + vyraz + "%";

    string connectionString = ConfigurationManager.ConnectionStrings["MovisContext"].ConnectionString;
    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        string queryString = "SELECT [Id],[Oznacenie],[Nazov],[Prijata] FROM [dbo].[DostupnePOView] WHERE [Oznacenie] LIKE @vyraz;";

        using (SqlCommand cmd = new SqlCommand(queryString, connection))
        {
            cmd.Parameters.AddWithValue("@vyraz", vyraz);

            try
            {
                connection.Open();

                using (SqlDataReader reader = cmd.ExecuteReader())
                {
                    List<Object> dostupneObjednavky = new List<object>();

                    while (reader.Read())
                    {
                        dostupneObjednavky.Add(new
                        {
                            Id = reader.GetInt32(0),
                            Oznacenie = reader.GetString(1),
                            Zakaznik = reader.IsDBNull(2) ? null : (string)reader.GetValue(2),
                            Prijata = reader.GetDateTime(3).ToString("dd.MM.yyyy")
                        });
                    }

                    response = Request.CreateResponse(HttpStatusCode.OK, dostupneObjednavky);
                }
            }
            catch (Exception)
            {
                response = Request.CreateResponse(HttpStatusCode.BadRequest);
            }
        }
    }

    return response;
}
```

Obrázok 74. Vyhľadávanie prijatých objednávok podľa užívateľom zadaného výrazu

8.5 Spracovanie položiek odoslanej objednávky

Textový reťazec s položkami na pridanie do odoslanej objednávky, ktorý príde v HTTP požiadavke na obrázku 75, musí byť pred zapísaním do databázy rozdelený na samostatné položky. Tým vznikne pole položiek, v ktorom každá položka obsahuje 2 atribúty. Z prvkov poľa, resp. položiek, sú extrahované atribúty, aby ich bolo možné zapísať do databázy.

```
public HttpResponseMessage PridajOP()
{
    HttpResponseMessage response = null;

    int objednavkaId = Int32.Parse(HttpContext.Current.Request["objednavkaId"], CultureInfo.InvariantCulture);
    string str = HttpContext.Current.Request["polozky"];
    string[] polozky = str.Split(';');
    List<ObjednanaPolozka> polozkyNaPridanie = new List<ObjednanaPolozka>();

    using (IMovisContext context = new MovisContext())
    {
        try
        {
            foreach (string riadokPolozky in polozky)
            {
                string[] polozkyRiadku = riadokPolozky.Split(',');
                int polozkapoid = Int32.Parse(polozkyRiadku[0].Split(':')[1], CultureInfo.InvariantCulture);
                int pocetKusov = Int32.Parse(polozkyRiadku[1].Split(':')[1], CultureInfo.InvariantCulture);

                PolozkaPO polozkapo = context.PolozkyPO.Single<PolozkaPO>(x => x.Id == polozkapoid);
                PrijataObjednavka po = polozkapo.PrijataObjednavka;
                ObjednanaPolozka objednanaPolozka = new ObjednanaPolozka
                {
                    PolozkaPOId = polozkapoid,
                    OdoslanaObjednavkaId = objednavkaId,
                    Nazov = polozkapo.Nazov,
                    Popis = string.IsNullOrWhiteSpace(polozkapo.Popis) ? po.Oznacenie : (polozkapo.Popis + "\r\n" + po.Oznacenie),
                    PocetKusov = pocetKusov,
                    MnozstevnaJednotka = polozkapo.MnozstevnaJednotka,
                    Pridana = DateTime.Now,
                };
                polozkyNaPridanie.Add(objednanaPolozka);
                context.ObjednanePolozky.Add(objednanaPolozka);
            }

            context.Save();
```

Obrázok 75. Spracovanie položiek odoslanej objednávky

8.6 Generovanie PDF

Generovanie PDF je len projekcia textových reťazcov a dát z databázy na čistú stranu PDF dokumentu. Problémom je, že veľkosť týchto dát nie je známa, teda nie je možné vedieť, koľko bude mať dokument PDF strán, nakoľko doklady majú neobmedzené množstvo položiek a podpoložiek. Je nevyhnutné v každom kroku sledovať, v akej výške strany sa projektovaný text nachádza.

V prvom kroku je potrebné načítať dáta z databázy pre zvolený dokument. Následne vytvoriť nový dokument na obrázku 76.

```
PdfDocument document = new PdfDocument();
```

Obrázok 76. Vytvorenie nového PDF dokumentu

Pridať čistú stranu a vytvoriť triedu, pomocou ktorej budú projektované dáta na danej strane. Príklad je znázornený na obrázku 77.

```
PdfPage page = document.AddPage();
XGraphics gfx = XGraphics.FromPdfPage(page);
```

Obrázok 77. Pridanie čistej strany

Vytvorenie fontov popisuje obrázok 78.

```
XFont font = new XFont("Arial", 19, XFontStyle.Bold);
XFont font2 = new XFont("Arial", 13, XFontStyle.Bold);
XFont font3 = new XFont("Arial", 11, XFontStyle.Bold);
XFont font4 = new XFont("Arial", 10, XFontStyle.Regular);
```

Obrázok 78. Vytvorenie fontov

Následne je možné na obrázku 79 začať písať údaje a importovať dáta z databázy na čistú stranu, príkazom DrawString. Tieto údaje sa umiestňujú pomocou „súradníc“ výška a šírka strany.

```
gfx.DrawString("Prijatá objednávka", font, XBrushes.Black,
    new XRect(50, 30, page.Width, page.Height), XStringFormats.TopLeft);
gfx.DrawString(zvolenaObjednavka.Oznacenie, font, XBrushes.Black,
    new XRect(226, 30, page.Width, page.Height), XStringFormats.TopLeft);
```

Obrázok 79. Vypisovanie údajov a import dát z databázy

Týmto spôsobom sú vypísané všetky základné údaje v danom dokumente o dodávateľovi, odberateľovi a pod. Následne je možné začať vypisovať jednotlivé položky dokladov, v tomto prípade sa nedá určiť počet týchto položiek, logicky bude v každom dokumente iný. Preto po pridaní každej položky, je nevyhnutné sledovať zvislú súradnicu strany. Maximálna výška strany pre položky bola nastavená na hodnotu 680, pretože na konci poslednej strany bude potrebné voľné miesto na záverečné údaje dokladu. Položky sa začnú vypisovať na hodnote 440. Po vypísaní jednej položky sa posunie zvislá súradnicu o hodnotu 40, týmto spôsobom sa dostane posledná položka na hodnotu 680 a potom sa vytvorí nová strana. Tento postup je zobrazený na obrázku 80.

```
int y = 440;
int z = 455;
int cisloPolozky = 1;
int MaxPageHeight = 680;

foreach (PolozkaPOView polozka in zvolenaObjednavka.PolozkyPO)
{
    if (y == MaxPageHeight)
    {
        PdfPage page2 = document.AddPage();
        XGraphics gfx2 = XGraphics.FromPdfPage(page2);
        gfx = gfx2;
        y = 40;
        z = 55;
    }
    gfx.DrawString(cisloPolozky.ToString() + ". " + polozka.Nazov, fontPolozka, XBrushes.Black,
        new XRect(50, y, page.Width, page.Height), XStringFormats.TopLeft);
    gfx.DrawString(polozka.Popis, fontPopisPolozky, XBrushes.Black,
        new XRect(50, z, page.Width, page.Height), XStringFormats.TopLeft);
    gfx.DrawString(polozka.PocetKusov.ToString(), fontPolozkaHodnoty, XBrushes.Black,
        new XRect(270, y, page.Width, page.Height), XStringFormats.TopLeft);
    gfx.DrawString(polozka.MnozstevnaJednotka, fontPolozkaHodnoty, XBrushes.Black,
        new XRect(300, y, page.Width, page.Height), XStringFormats.TopLeft);
    gfx.DrawString(polozka.CenaZaKusStr.ToString(), fontPolozkaHodnoty, XBrushes.Black,
        new XRect(360, y, page.Width, page.Height), XStringFormats.TopLeft);
    gfx.DrawString(polozka.DphStr.ToString() + "%", fontPolozkaHodnoty, XBrushes.Black,
        new XRect(420, y, page.Width, page.Height), XStringFormats.TopLeft);
    gfx.DrawString(polozka.CenaSpoluDphStr.ToString(), fontPolozkaHodnoty, XBrushes.Black,
        new XRect(480, y, page.Width, page.Height), XStringFormats.TopLeft);
    gfx.DrawLine(XPens.Black, 545, z + 15, 50, z + 15);

    z += 40;
    y += 40;
    cisloPolozky++;
```

Obrázok 80. Import položiek dokladu do PDF dokumentu

Záver poslednej strany obsahuje záverečný text, celkovú cenu a cenu bez DPH. Na obrázku 81 je možné vidieť, že tieto dáta sú umiestňované podľa premennej „y“, ktorá definuje umiestnenie poslednej položky.

```
gfx.DrawString(zvolenaObjednavka.ZaverecnyText, font4, XBrushes.Black,
    new XRect(50, y + 5, page.Width, page.Height), XStringFormats.TopLeft);

gfx.DrawLine(XPens.Black, 250, y + 48, 490, y + 48);
gfx.DrawString("Celkom bez DPH:", fontCenaText, XBrushes.Black,
    new XRect(250, y + 30, page.Width, page.Height), XStringFormats.TopLeft);
gfx.DrawString(zvolenaObjednavka.SpoluBezDph + " " + zvolenaObjednavka.Mena.ToUpper(), fontCenaText, XBrushes.Black,
    new XRect(390, y + 30, page.Width, page.Height), XStringFormats.TopLeft);
gfx.DrawString("Celkom:", fontCena, XBrushes.Black,
    new XRect(250, y + 52, page.Width, page.Height), XStringFormats.TopLeft);
gfx.DrawString(zvolenaObjednavka.SpoluDph + " " + zvolenaObjednavka.Mena.ToUpper(), fontCena, XBrushes.Black,
    new XRect(390, y + 52, page.Width, page.Height), XStringFormats.TopLeft);
```

Obrázok 81. Import záverečných údajov do PDF dokumentu

Ostáva už len miesto pre pečiatku, ktorá bude v prípade splnenia podmienky na obrázku 82 umiestnená na novej strane.

```
if (y >= 640)
{
    PdfPage page3 = document.AddPage();
    XGraphics gfx3 = XGraphics.FromPdfPage(page3);
    gfx = gfx3;
    y = -30;
}

gfx.DrawLine(XPens.Black, 545, y + 195, 335, y + 195);
gfx.DrawString("Pečiatka a podpis", font4, XBrushes.Black,
    new XRect(400, y + 200, page.Width, page.Height), XStringFormats.TopLeft);
```

Obrázok 82. Nakreslenie miesta pre pečiatku a podpis

Na obrázku 83 je možné vidieť ukážku PDF dokumentu cenovej ponuky.

Cenová ponuka QTN2021-1

Dodávateľ

Meno dodávateľa
Adresa dodávateľa
Mesto dodávateľa
Krajina dodávateľa

IČO: 156578656
DIČ: 989855655
IČ DPH: 99862555

Odberateľ

Firma
Adresa
Mesto
PSČ
Štát

IČO: 155574
DIČ: 6515654546
IČ DPH: 5616516565

Dátum vystavenia: 11.10.2021

Platí do: 25.10.2021

Suma: **5 842,80 EUR**

Môj úvodný text

Názov položky	Počet	MJ	Cena bez DPH	DPH	Spolu s DPH
1. Moja polozka c.1 dopytu Popis polozky c.1 dopytu	8	ks	45,00	20%	432,00
2. Moja polozka c.2 dopytu Popis polozky c.2 dopytu	6	kg	70,00	20%	504,00
3. Moja polozka c.3 dopytu Popis polozky c.3 dopytu	25	m	80,00	20%	2 400,00
4. Moja polozka c.4 dopytu Popis polozky c.4 dopytu	8	l	98,00	20%	940,80
5. Moja polozka c.5 dopytu Popis polozky c.5 dopytu	15	kg	87,00	20%	1 566,00

Môj záverečný text

Celkom bez DPH: 4 869,00 EUR
Celkom: **5 842,80 EUR**

Pečiatka a podpis

Obrázok 83. Vygenerované PDF

9 ZABEZPEČENIE APLIKÁCIE

Zabezpečenie webových aplikácií je veľmi dôležité. Na zabezpečenie aplikácií treba dbať už pri vývoji, následne pri nasadení do prevádzky, ale aj počas celej životnosti aplikácie. Útoky na webové aplikácie najčastejšie smerujú ku krádežiam dát, alebo poškodeniu celej databázy, prípadne len jej časti. V tejto kapitole bude vysvetlené, akým spôsobom je možné zabezpečiť webové aplikácie. Súčasťou kapitoly bude aj testovanie odolnosti tejto aplikácie voči jednému z najznámejších a najčastejších útokov.

9.1 SQL injection

SQL injection je útok zameraný na webové aplikácie, ktorý pozostáva z vloženia dotazu jazyka SQL, prostredníctvom vstupných údajov od klienta do aplikácie. Tento útok umožňuje neoprávnené čítanie, upravovanie (vkladanie, aktualizáciu, mazanie) údajov databázy, vykonávať administratívne operácie v databáze (ako napríklad vypnutie databázového systému) a v niektorých prípadoch vydávať príkazy operačnému systému. [31]

Útočník pri väčšine útokov typu SQL injection potrebuje poznať názvy entít databázy. Túto znalosť by s najväčšou pravdepodobnosťou nemal, avšak nemusí byť moc zložité tieto informácie odhadnúť z funkcionality aplikácie.

9.1.1 Parametrizácia dotazov

Parametrizácia dotazov je jedným z najúčinnejších spôsobov ochrany voči útoku SQL injection. Kód na obrázku 84 zabezpečuje import vstupných reťazcov z užívateľského rozhrania do databázy kontrolovaným spôsobom, resp. vstupné reťazce sú spracovávané len ako hodnoty, ktoré je potrebné uložiť do databázy nie ako dotazy jazyka SQL. [32]

```
txtNam = getRequestString("CustomerName");
txtAdd = getRequestString("Address");
txtCit = getRequestString("City");
txtSQL = "INSERT INTO Customers (CustomerName,Address,City) Values(@0,@1,@2)";
db.Execute(txtSQL,txtNam,txtAdd,txtCit);
```

Obrázok 84. Príklad parametrických dotazov [32]

Vyššie uvedený príklad transformuje 3 vstupné reťazce (txtNam, txtAdd, txtCit) na parametre (@0, @1, @,2) a až potom ich importuje do databázy.

Dnes už väčšina vývojových technológií, automaticky bez vedomia vývojárov využíva parametrické dotazy. V tejto aplikácii predstavuje riziko len úmyselné poškodenie štruktúry databázy, napr. zmazanie entít, pretože aplikácia nie je vybavená autentizáciou užívateľa, z čoho vyplýva, že užívatelia nemajú rozdielne oprávnenia a môžu pristupovať relatívne k všetkým dátam uloženým v databáze. Problém potencionálneho rizika poškodenia databázy rieši technológia LINQ to SQL, ktorá automaticky mení vstupy užívateľského rozhrania na parametre, bez implementácie ošetrenia pomocou parametrických dotazov v kóde. [33]

9.1.2 Ošetrenie vstupov

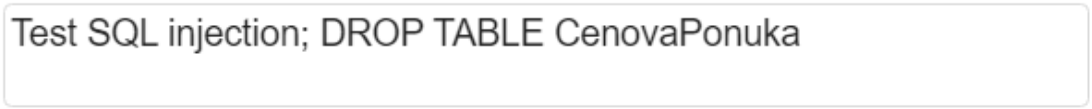
Eliminovať alebo aspoň minimalizovať riziko útoku SQL injection je možné aj pomocou ošetrenia vstupov, prostredníctvom regulárnych výrazov. V tomto prípade je nevyhnutné vedieť odhadnúť presnú množinu znakov, ktorá je pre dané vstupy potrebná a následne vytvoriť regulárny výraz, ktorý akceptuje iba túto množinu znakov. Prípadne aspoň eliminovať najčastejšie sa využívajúce znaky pri útoku SQL injection (“, =, ;), ak je to možné.

Ošetrenie vstupov je možné implementovať na strane klienta v jazyku javascript. V tomto prípade by toto zabezpečenie mohol útočník jednoducho obísť. Ak by v prehliadači zakázal využívanie javascriptu, v tom prípade by validácia vstupov neprebehla vôbec.

Druhou možnosťou je ošetriť vstupy rovnakým spôsobom na servery, resp. porovnávať vstupy s regulárny výrazom až na servery. V tomto prípade už útočník nedokáže obísť toto zabezpečenie tak jednoducho.

9.1.3 Test SQL injection

Funkčnosť automatickej parametrizácie dotazov technológie LINQ to SQL bola otestovaná vytvorením útoku SQL injection na vlastnú databázu. V užívateľskom rozhraní cenových ponúk bol vložený do vstupného poľa za “užitočný vstup“ škodlivý príkaz, ktorý by mal zmazať celú entitu cenových ponúk.



Obrázok 85. Vloženie škodlivého kódu do užívateľského vstupu

Vstup na obrázku 85 by mal vytvoriť dva dotazy, definované na obrázku 86 samozrejme iba v prípade nefunkčnosti parametrizácie dotazov.

```
UPDATE CenovaPonuka SET ZaverecnyText = 'Test SQL injection' WHERE Id = 341;
DROP TABLE CenovaPonuka;
```

Obrázok 86. SQL dotazy pri útoku typu SQL injection

Prvý riadok na obrázku 86 predstavuje "užitočný" dotaz, ktorý vloží do databázy požadovanú hodnotu, druhý riadok predstavuje škodlivý príkaz, ktorý zmaže celú entitu cenových ponúk a s ňou samozrejme aj všetky dáta.

Results Messages

	Id	Oznacenie	Vytvorena	Vytvoril	UvodnyText	ZaverecnyText	Mena	Poznamka
1	330	QTN2021-2	2021-10-11 10:27:08.823	Martin Oúkaný	Môj úvodný text	Môj záverečný text	czk	NULL
2	331	QTN2021-1	2021-10-11 10:09:52.290	Martin Oúkaný	Môj úvodný text	Môj záverečný text	eur	NULL
3	333	QTN2021-2	2021-12-18 18:30:25.010	Martin Oúkaný	NULL	test	eur	NULL
4	341	QTN2022-1	2022-04-16 15:37:01.077	Martin Oúkaný	NULL	Test SQL injection; DROP TABLE CenovaPonuka	eur	NULL

Obrázok 87. Zobrazenie dát v SQL server mangemet studiu po útoku SQL injection

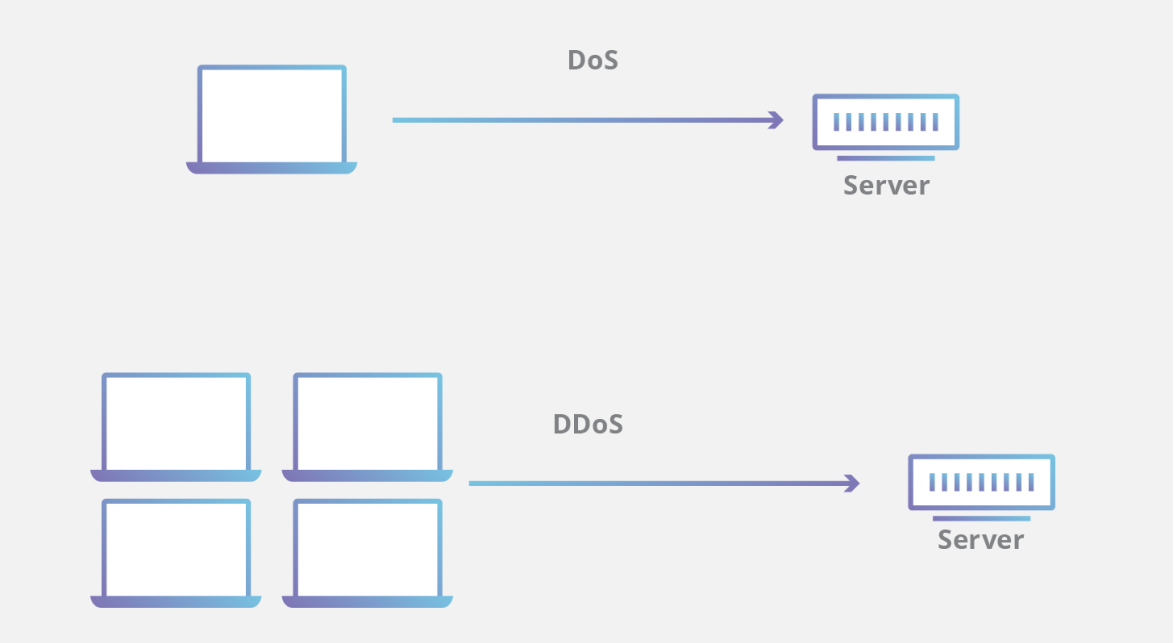
Z obrázku 87 vyplýva, že ochrana voči SQL injection v technológií LINQ to SQL funguje a celý vstup bol zapísaný do databázy ako parameter, teda vstup sa nepodarilo rozdeliť na dva SQL dotazy.

9.2 DOS a DDOS

Jedná sa o útoky, ktoré znemožňujú alebo obmedzujú prístup k systému legitímnym užívateľom. Pri webových aplikáciách útočník zahltí server veľkým množstvom požiadaviek, útočník z neho väčšinou nemá priamy profit, ide hlavne o spôsobenie škody. DOS je útok z jedného zdroja, DDOS je distribuovaný útok z viacerých zdrojov. Ich rozdiel je znázornený na obrázku 88. [34]

Príznaky DOS a DDOS útoku: [34]

- Spadnutie systému
- Pomalá odozva systému
- Nedostupnosť časti systému
- Strata dát zo systému



Obrázok 88. DOS a DDOS útok [35]

9.2.1 Obrana voči DOS a DDOS útokom

Nakoľko tieto útoky sú smerované väčšinou iba na to, aby napadli infraštruktúru, nepoužívajú sa na ich obranu bezpečnostné praktiky pri vývoji softvéru. Webové aplikácie môže útočník využiť ako sprostredkovateľa na zahltenie serveru požiadavkami. Obrana spočíva skôr v konfigurácií zariadení sieťovej infraštruktúry, prípadne pridaním špeciálnych zariadení, ktoré dokážu eliminovať tento typ útoku do sieťovej infraštruktúry. Pri útoku typu DOS je možné obmedziť požiadavky na server v časovom intervale z jednej IP adresy, avšak tento spôsob je pri útoku DDOS neúčinný.

ZÁVER

Mojím cieľom v diplomovej práci bolo vytvorenie softvérového modulu pre fakturáciu, ktorý umožňuje užívateľovi využívať aj funkcie, ktoré nie sú bežne dostupné v komerčných verziách, napríklad spôsob riadenia nákladov a obchodných kooperácií v kapitole 2.2.

V prvej kapitole tejto práce som obecne vysvetlil princípy a technológie, ktoré by mali čitateľovi zjednodušiť pochopenie praktickej časti a samotného vývoja aplikácie. V druhej kapitole som pomocou schémy funkcionality aplikácie opísal spôsob využitia tejto aplikácie a zdôraznil jej zmysel voči komerčným verziám na praktickom príklade.

Tretia kapitola opisuje databázu v SQL server managemet studiu, vytvorenie jednotlivých entít spolu s atribútmi a ich následné pospájanie pomocou relačných vzťahov. Záver tejto kapitoly znázorňuje kompletnú schému databázy a spôsob generovania označení dokladov. Štvrtá kapitola znázorňuje spôsob mapovania jednotlivých entít na objektové triedy jazyka C# a prepojenie atribútov databázy na properties. Úvod piatej kapitoly je venovaný jazykom HTML a CSS, ktoré svojou vzájomnou spoluprácou efektívne vytvárajú užívateľské rozhranie a úvodu do jazyka javascript, ktorý riadi dynamiku užívateľského rozhrania. Zvyšná časť kapitoly dokumentuje stavebné bloky užívateľského rozhrania, vytvorené pomocou hlavných súborov a webových formulárov. Šiesta kapitola vysvetľuje princíp načítania dát z databázy pri spustení aplikácie. Úlohou siedmej kapitoly je čitateľovi vysvetliť princíp komunikácie klienta so serverom a spôsoby detekcie užívateľských požiadaviek s ich následným odosielaním na server. Táto kapitola taktiež znázorňuje princíp validácie užívateľských vstupov a implementáciu výpočtov, ktoré sú vykonávané na strane klienta. Ôsma kapitola ukazuje spôsoby spracovania a vybavenia užívateľských požiadaviek na strane servera, s následnou odpoveďou, ktorá nesie požadovaný výsledok. Posledná kapitola oboznamuje čitateľa s bezpečnostnými hrozbami, ktorým čelia dnešné webové aplikácie, pričom tieto hrozby často smerujú prostredníctvom webových aplikácií na samotný server. Kapitola ďalej predstavuje spôsoby obrany voči týmto hrozbám.

Aplikácia umožňuje užívateľovi pohodlne realizovať všetky fakturačné procesy, mimo iného využívať aj ďalšiu funkcionalitu, ako napríklad analytický prehľad, možnosť generovania PDF dokumentov a pod. Samozrejme, existuje veľa možností doplnenia aplikácie o ďalšie funkcionality. Užívatelia by určite ocenili podrobnejší prehľad, resp. pridanie ďalších webových formulárov k prehľadom, ktoré by mohli analyticky znázorniť grafickou formou úspech podnikania poslednej doby, porovnanie s dobou minulou a pod. Pri samot-

ných faktúrach by mohla zákazníkovi zjednodušiť spôsob platby metóda „scan to pay“, ktorá umožňuje zaplatiť faktúru pomocou naskenovania QR kódu.

SEZNAM POUŽITÉ LITERATURY

[1] [online]. [cit. 2022-1-3]. Dostupné z: https://www.geeksforgeeks.org/introduction-to-visual-studio/

[2] [online]. [cit. 2022-1-3]. Dostupné z: https://www.javatpoint.com/asp-net-introduction

[3] [online]. [cit. 2022-1-12]. Dostupné z: http://www-db.deis.unibo.it/courses/TW/DOCS/w3schools/aspnet/aspnet_masterpages.asp.html

[4] [online]. [cit. 2022-1-3]. Dostupné z: https://fileinfo.com/extension/aspx

[5] [online]. [cit. 2022-1-12]. Dostupné z: https://www.javatpoint.com/sql-server-management-studio

[6] [online]. [cit. 2022-1-6]. Dostupné z: https://www.guru99.com/what-is-sql.html

[7] [online]. [cit. 2022-2-8]. Dostupné z: https://www.guru99.com/er-diagram-tutorial-dbms.html

[8] [online]. [cit. 2022-1-12]. Dostupné z: https://www.guru99.com/er-diagram-tutorial-dbms.html

[9] [online]. [cit. 2022-2-8]. Dostupné z: https://www.ibm.com/docs/en/informix-servers/12.10?topic=SSGU8G_12.1.0/com.ibm.ddi.doc/ids_ddi_186.htm

[10] [online]. [cit. 2022-2-8]. Dostupné z: https://www.w3schools.com/sql/sql_view.asp

[11] [online]. [cit. 2022-2-8]. Dostupné z: https://www.guru99.com/er-diagram-tutorial-dbms.html

[12] [online]. [cit. 2022-2-12]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ef/overview

[13] [online]. [cit. 2022-2-12]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/api/system.data.entity.dbmodelbuilder?view=entity-framework-6.2.0

[14] [online]. [cit. 2022-3-5]. Dostupné z: https://www.czechitas.cz/blog/frontend-vs-backend

[15] [online]. [cit. 2022-3-5]. Dostupné z: https://rockcontent.com/blog/scripting-languages/

[16] [online]. [cit. 2022-3-25]. Dostupné z: https://www.hackreactor.com/blog/what-is-javascript-used-for

[17] [online]. [cit. 2022-3-25]. Dostupné z: https://www.javatpoint.com/what-is-jquery

[18] [online]. [cit. 2022-3-26]. Dostupné z: https://web.archive.org/web/20161101212501/http://www.regular-expressions.info/tutorial.html

[19] [online]. [cit. 2022-4-2]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/

[20] [online]. [cit. 2022-4-2]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/

[21] [online]. [cit. 2022-4-2]. Dostupné z: http://www.pdfsharp.net/

[22] [online]. [cit. 2022-4-2]. Dostupné z: https://developer-mozilla-org.translate.goog/en-US/docs/Learn/JavaScript/Client-si-de_web_APIs/Introduction?_x_tr_sl=en&_x_tr_tl=sk&_x_tr_hl=sk&_x_tr_pto=sc

[23] [online]. [cit. 2022-4-16]. Dostupné z: https://www-w3schools-com.translate.goog/js/js_api_intro.asp?_x_tr_sl=en&_x_tr_tl=sk&_x_tr_hl=sk&_x_tr_pto=sc

[24] [online]. [cit. 2022-4-16]. Dostupné z: https://www.howtogeek.com/721685/what-is-a-markup-language/

[25] [online]. [cit. 2022-4-16]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS

[26] [online]. [cit. 2022-4-16]. Dostupné z: https://www.w3schools.com/whatis/whatis_http.asp

[27] [online]. [cit. 2022-4-26]. Dostupné z: https://sk.sawakinome.com/articles/technology/difference-between-get-and-post-method-in-php-2.html

[28] [online]. [cit. 2022-4-26]. Dostupné z: https://www.codecademy.com/article/mvc

[29] [online]. [cit. 2022-5-1]. Dostupné z: https://www.freecodecamp.org/news/the-model-view-controller-pattern-mvc-architecture-and-frameworks-explained/

[30] [online]. [cit. 2022-5-1]. Dostupné z: https://www.davidhayden.me/blog/asp.net-mvc-4-web-api-routes-and-apicontroller

[31] [online]. [cit. 2022-5-5]. Dostupné z: https://owasp.org/www-community/attacks/SQL_Injection

[32] [online]. [cit. 2022-5-5]. Dostupné z: https://www.w3schools.com/sql/sql_injection.asp

[33] [online]. [cit. 2022-5-6]. Dostupné z: https://entityframework.net/linq-prevent-sql-injection

[34] [online]. [cit. 2022-5-5]. Dostupné z: https://www.w3schools.in/ethical-hacking/dos-attacks-and-its-prevention/

[35] [online]. [cit. 2022-5-5]. Dostupné z: https://www.hostinger.com/tutorials/what-is-html

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

ASPX	Prípona súboru pre webový formulár
C	Objektovo orientovaný jazyk
C#	Objektovo orientovaný jazyk
CSS	Kaskádové štýly
DOS	Útok odmietnutia služby
DDOS	Distribuovaný útok odmietnutia služby
DPH	Daň z pridanej hodnoty
ER diagram	Diagram vzťahov medzi entitami
GET	Dopytovacia metóda http protokolu
HTML	Hypertextový značkovací jazyk
HTTP	Hypertextový prenosový protokol
ID	Jedinečný identifikátor
macOS	Operačný systém od spoločnosti Apple
MIT	Massachusetts Institute of Technology
PDF	Formát súboru
POST	Dopytovacia metóda HTTP protokolu
QR	Dvojrozmerný čiarový kód
SQL	Štandardný jazyk, pre prístup a manipuláciu s databázami
SSMS	SQL Server Management Studio
URL	Odkaz na webový zdroj
USD	Americký dolár
Web API	Aplikačné programovacie rozhranie pre web
XML	Značkovací jazyk

SEZNAM OBRÁZKŮ

ZOZNAM TABULIEK