**EKONOMICKÁ UNIVERZITA V BRATISLAVE**

**FAKULTA Hospodárskej informatiky**

Evidenčné číslo: 103004/I/2023/421000214229

**ASP .NET XML WEBOVÁ SLUŽBA POSKYTUJÚCA USPORIADANÉ INFORMÁCIE O KNIHÁCH ELEKTRONICKÉHO KNÍHKUPECTVA**

Diplomová práca

**2023 Bc. Martin Jankech**

**EKONOMICKÁ UNIVERZITA V BRATISLAVE**

**FAKULTA Hospodárskej informatiky**

**ASP .NET XML WEBOVÁ SLUŽBA POSKYTUJÚCA USPORIADANÉ INFORMÁCIE O KNIHÁCH ELEKTRONICKÉHO KNÍHKUPECTVA**

Diplomová práca

**Študijný program:** Informačný manažment

**Študijný odbor:** Ekonómia a manažment

**Školiace pracovisko:** Katedra aplikovanej informatiky

**Vedúci záverečnej práce:** Ing. Igor Koštál, PhD.

**Bratislava 2023 Martin Jankech**



Ekonomická univerzita v Bratislave

Fakulta hospodárskej informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

**Meno a priezvisko študenta:** Bc.Martin Jankech

**Študijný program:** informačný manažment (Jednoodborové štúdium, inžiniersky II. st., denná forma)

**Študijný odbor:** ekonómia a manažment

**Typ záverečnej práce:** Inžinierska záverečná práca

**Jazyk záverečnej práce:** slovenský

**Sekundárny jazyk:** anglický

**Názov:** ASP .NET XML webová služba poskytujúca usporiadané informácie o knihách elektronického kníhkupectva

**Anotácia:** Diplomant v práci zanalyzuje možnosti použitia XML webových služieb na sledovanie parametrov kníh elektronického kníhkupectva vo vybratom období v elektronickom informačnom systéme a porovná ich použitie s doterajším spôsobom sledovania týchto parametrov vo vybraných kníhkupectvách. V rámci diplomovej práce diplomant vo vybratom riadenom programovacom jazyku vytvorí ASP .NET XML webovú službu poskytujúcu prostredníctvom svojej funkcionality jej klientovi podľa vybraných kritérií usporiadané informácie o parametroch kníh elektronického kníhkupectva v sledovanom období, ktorými môžu byť celkový počet kníh v tomto kníhkupectve na začiatku a konci sledovaného obdobia, počty a zoznamy kníh s ich základnými dátami od jednotlivých vydavateľov, autorov, počty a zoznamy kníh s ich základnými dátami s najväčším a najmenším predajom na začiatku a konci sledovaného obdobia a iné ich parametre.

**Vedúci:** Ing. Igor Košťál, PhD.

**Katedra:** KAI FHI - Katedra aplikovanej informatiky

**Vedúci katedry:** Ing. Mgr. Peter Schmidt, PhD.

**Dátum zadania:** 25.10.2021

**Dátum schválenia:** 17.04.2023 prof. Ing. Ivan Brezina, CSc

osoba zodpovedná za realizáciu študijného programu

**Čestné vyhlásenie**

Vyhlasujem, že som celú diplomovú prácu vypracoval samostatne s použitím uvedenej odbornej literatúry.

 Bratislava dňa 28.04.2023 .................................

Podpis študenta

**Poďakovanie**

Touto cestou, by som sa chcel poďakovať môjmu vedúcemu diplomovej práce Ing. Igorovi Koštálovi, PhD. za odbornú pomoc, vedenie, konzultácie, užitočné rady a pripomienky pri vypracovaní mojej diplomovej práce.

Abstrakt

JANKECH, Martin: *ASP .NET XML webová služba poskytujúca usporiadané informácie o knihách elektronického kníhkupectva. –* Ekonomická univerzita v Bratislave. Fakulta hospodárskej informatiky; Katedra aplikovanej informatiky. – Vedúci záverečnej práce: Ing. Igor Košťál, PhD. - Bratislava: FHI EU, 2023, 111 s.

Hlavným cieľom záverečnej práce je navrhnúť a vytvoriť ASP.NET XML webovú službu poskytujúcu svojmu klientovi usporiadané informácie o rôznych parametroch kníh elektronického kníhkupectva. Samotnému návrhu a vytvoreniu webovej služby predchádza teoretické vymedzenie webových služieb, ako aj analýza a porovnanie možností využitia webových služieb v konkrétnych kníhkupectvách. Webová služba je implementovaná pomocou ASP.NET frameworku a programovacieho jazyka C#. Práca je rozdelená do šiestich kapitol a obsahuje 7 tabuliek a 56 obrázkov. Prvá kapitola je venovaná definícii pojmu webová služba, jej vlastnostiam, histórii a architektúre. Sú tu popísané najdôležitejšie otvorené štandardy a prenosové protokoly využívané pri tvorbe a prevádzke webových služieb. V druhej a tretej kapitole je charakterizovaný cieľ a metodika skúmania, ktorá bola použitá v diplomovej práci. V štvrtej kapitole je vykonaná analýza informačného systému a webových služieb v konkrétnom kníhkupectve.

Záverečné dve kapitoly sa zaoberajú plnením a zhodnotením splnenia hlavného cieľa práce. Je tu popísané vytvorenie XML dátovej základne, webovej služby a webového klienta a v kapitole Diskusia je služba následné zhodnotená a sú popísané možnosti jej nasadenia.

Výsledkom riešenia danej problematiky je vytvorená webová služba, ktorá sa spolu s klientom dá integrovať do reálnych kníhkupectiev rôznych veľkostí.

**Kľúčové slová:** webová služba, XML, ASP .NET, online kníhkupectvo

Abstract

JANKECH, Martin: ASP .NET XML Web service that provides sorted information about e-bookstore books*. –* University of Economics in Bratislava. Faculty of Economic Informatics; Department of Applied Informatics. – Leader of the final thesis: Ing. Igor Košťál PhD. -Bratislava: FHI EU, 2023, 111 s.

The main goal of the thesis is to design and develop an ASP.NET XML web service providing its client with sorted information about various parameters of the e-bookstore books. The actual design and creation of the web service is preceded by a theoretical definition of web services, as well as an analysis and comparison of the possibilities of using web services in specific bookstores. The web service is implemented using the ASP.NET framework and the C# programming language. The thesis is divided into six chapters and contains 7 tables and 56 figures. The first chapter is devoted to the definition of the term web service, its features, history and architecture. The most important open standards and transmission protocols used in the creation and operation of Web services are described. The second and third chapters characterize the aim and methodology of the research used in the thesis. In chapter four, an analysis of the information system and web services in a particular bookstore is conducted.

The final two chapters deal with the fulfilment and evaluation of the main objective of the thesis. The creation of the XML data base, the web service and the web client is described and in the Discussion chapter the service is subsequently evaluated and the possibilities of its deployment are described.

As a result of solving the given problem, a web service has been created that can be integrated together with the client into real bookstores of different sizes.

**Key words**: web service, XML, ASP NET framework, online bookstore

**Obsah**

[Úvod 12](#_Toc132640468)

[1 Webové služby – ich história, architektúra, najdôležitejšie otvorené štandardy a prenosové protokoly 13](#_Toc132640469)

[1.1 Definícia webovej služby 13](#_Toc132640470)

[1.1.1 Rozdiel medzi webovou službou a webovou aplikáciou 14](#_Toc132640471)

[1.1.2 Vlastnosti webových služieb 15](#_Toc132640472)

[1.2 Cesta k webovým službám 15](#_Toc132640473)

[1.2.1 CORBA, RMI, DCOM 15](#_Toc132640474)

[1.2.2 XML-RPC, SOAP, REST, GRAPHQL 16](#_Toc132640475)

[1.3 Architektúra webovej služby 19](#_Toc132640476)

[1.4 Životný cyklus vývoja webovej služby 20](#_Toc132640477)

[1.5 HTTP protokol 22](#_Toc132640478)

[1.5.1 Vlastnosti HTTP 22](#_Toc132640479)

[1.5.2 HTTP správy 23](#_Toc132640480)

[1.5.3 HTTP požiadavka 23](#_Toc132640481)

[1.5.4 HTTP odpoveď 24](#_Toc132640482)

[1.5.5 HTTP metódy 25](#_Toc132640483)

[1.5.6 HTTP stavové kódy 27](#_Toc132640484)

[1.6 XML 28](#_Toc132640485)

[1.6.1 Vlastnosti XML 28](#_Toc132640486)

[1.6.2 Základná syntax a pravidlá 29](#_Toc132640487)

[1.7 JSON 32](#_Toc132640488)

[1.8 JSON základná syntax 33](#_Toc132640489)

[1.8.1 JSON vs XML 34](#_Toc132640490)

[1.9 WSDL 35](#_Toc132640491)

[1.10 SOAP 36](#_Toc132640492)

[1.11 UDDI 39](#_Toc132640493)

[1.12 SOAP vs REST 39](#_Toc132640494)

[2 Ciele práce 42](#_Toc132640495)

[3 Metodika práce a metódy skúmania 44](#_Toc132640496)

[4 Súčasný stav vo vybraných firmách 46](#_Toc132640497)

[5 Výsledky práce 49](#_Toc132640498)

[5.1 Tvorba dátovej XML základne 49](#_Toc132640499)

[5.1.1 Opis dokumentu books.xml 49](#_Toc132640500)

[5.1.2 Opis dokumentu books\_transactions.xml 52](#_Toc132640501)

[5.1.3 Naplnenie XML súborov testovacími dátami 54](#_Toc132640502)

[5.2 Definovanie požiadaviek 56](#_Toc132640503)

[5.2.1 Požiadavky na metódy webovej služby 56](#_Toc132640504)

[5.2.2 Požiadavky na klienta webovej služby 58](#_Toc132640505)

[5.3 Tvorba webovej služby 58](#_Toc132640506)

[5.3.1 Použité technológie pri tvorbe webovej služby 58](#_Toc132640507)

[5.3.2 Popis riešenia vo Visual Studiu 59](#_Toc132640508)

[5.3.3 Diagram tried 61](#_Toc132640509)

[5.3.4 Popis úvodných riadkov webovej služby 61](#_Toc132640510)

[5.3.5 Popis newebových metód 63](#_Toc132640511)

[5.3.6 Popis webových metód 67](#_Toc132640512)

[5.3.7 Testovanie a nasadenie na server 93](#_Toc132640513)

[5.4 Tvorba webového klienta 94](#_Toc132640514)

[5.4.1 Technológie použité pri tvorbe klienta 94](#_Toc132640515)

[5.4.2 Prepojenie vystavených metód webovej služby s klientom 95](#_Toc132640516)

[5.4.3 Oznamy pre používateľa 97](#_Toc132640517)

[5.4.4 Automatické dopĺňanie do vstupov formulára 98](#_Toc132640518)

[5.4.5 Ukážky obrazoviek 99](#_Toc132640519)

[Diskusia 105](#_Toc132640520)

[Záver 107](#_Toc132640521)

[6 Zoznam použitej literatúry 108](#_Toc132640522)

**Zoznam obrázkov**

[Obrázok 1: Príklad klient/server komunikácie dvoch aplikácií pomocou webovej služby zdroj: [48] 11](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014036)

[Obrázok 2: Komunikačné protokoly na časovej osi zdroj: [11] 18](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014037)

[Obrázok 3: Architektúra webovej služby zdroj: [12] 20](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014038)

[Obrázok 4: Životný cyklus vývoja webovej služby zdroj: [13] 22](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014039)

[Obrázok 5: Príklad HTTP požiadavky zdroj: [20] 24](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014040)

[Obrázok 6: Príklad HTTP požiadavky zdroj: [20] 25](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014041)

[Obrázok 7: Príklad stromovej štruktúry XML dokumentu zdroj: [30] 30](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014042)

[Obrázok 8: Príklad XML dokumentu zdroj: vlastné spracovanie 30](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014043)

[Obrázok 9: Príklad použitia XML menných priestorov zdroj: vlastné spracovanie 32](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014044)

[Obrázok 10: Príklad JSON kódu zdroj: vlastné spracovanie 33](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014045)

[Obrázok 11: Štruktúra WSDL dokumentu (verzia 1.1 a verzia 2.0) zdroj: [47] 36](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014046)

[Obrázok 12: Príklad SOAP požiadavky zdroj: vlastné spracovanie 38](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014047)

[Obrázok 13: Príklad SOAP odpovede zdroj: vlastné spracovanie 38](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014048)

[Obrázok 14: Ukážka stromovej štruktúry XML dokumentu books.xml zdroj: vlastné spracovanie 49](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014049)

[Obrázok 15: Príklad stromovej štruktúry XML dokumentu book\_transakcie.xml zdroj: vlastné spracovanie 52](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014050)

[Obrázok 16: Ukážka mapovania XML schémy na tabuľku v Exceli zdroj: vlastné spracovanie 55](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014051)

[Obrázok 17: Solution vo Visual Studiu zdroj: vlastné spracovanie 59](#_Toc132014052)

[Obrázok 18: Diagram tried vygenerovaný vo Visual Studiu zdroj: vlastné spracovanie 61](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014053)

[Obrázok 19: Časť zdrojového kódu definície triedy “book\_services“ zdroj: vlastné spracovanie 61](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014054)

[Obrázok 20: Zdrojový kód obsahujúci triedy BookData a TransactionData zdroj: vlastné spracovanie 62](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014055)

[Obrázok 21: Zdrojový kód členskej newebovej metódy LoadXmlDocument zdroj: vlastné spracovanie 63](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014056)

[Obrázok 22: Časť zdrojového kódu členskej newebovej metódy LoadXElement zdroj: vlastné spracovanie 64](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014057)

[Obrázok 23: Časť zdrojového kódu členskej newebovej metódy LoadXDocument zdroj: vlastné spracovanie 64](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014058)

[Obrázok 24: Zdrojový kód členskej newebovej metódy CreateTimestamp zdroj: vlastné spracovanie 64](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014059)

[Obrázok 25: Zdrojový kód členskej newebovej metódy WriteToTheFileWithTimeStamp zdroj: vlastné spracovanie 64](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014060)

[Obrázok 26: Zdrojový kód členskej newebovej metódy SaveWebMethodResult zdroj: vlastné spracovanie 66](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014061)

[Obrázok 27: Zdrojový kód webovej metódy AddBook zdroj: vlastné spracovanie 67](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014062)

[Obrázok 28: Zdrojový kód webovej metódy UpdateBook zdroj: vlastné spracovanie 69](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014063)

[Obrázok 29: Zdrojový kód webovej metódy DeleteBook zdroj: vlastné spracovanie 71](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014064)

[Obrázok 30: Zdrojový kód webovej metódy SinglebookDataByName zdroj: vlastné spracovanie 72](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014065)

[Obrázok 31: Zdrojový kód webovej metódy GetListAllBooks zdroj: vlastné spracovanie 74](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014066)

[Obrázok 32: Zdrojový kód webovej metódy SortedBookAmoutsByDateAndAtribute zdroj: vlastné spracovanie 74](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014067)

[Obrázok 33: Ukážka vstupov a výstupov pre metódu SortedBookAmoutsByDateAndAtribute (1. výstup v JSON pre klienta 2. výstup do diskového súboru s časovou pečiatkou a parametrami v XML ) zdroj: vlastné spracovanie 80](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014068)

[Obrázok 34: Sekcia zdrojového kódu webovej metódy AgregatedStatiscticsAmount zdroj: vlastné spracovanie 81](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014069)

[Obrázok 35: Ukážka vstupov a výstupov pri webovej metóde AgregatedStatiscticsAmount zdroj: vlastné spracovanie 82](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014070)

[Obrázok 36: Časť zdrojového kódu webovej metódy SortedDrillDownByAtributeDataBetweenTwoDatesSell zdroj: vlastné spracovanie 85](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014071)

[Obrázok 37: Ukážka vstupu a výstupu č.1 pri webovej metóde SortedDrillDownByAtributeDataBetweenTwoDatesSell zdroj: vlastné spracovanie 89](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014072)

[Obrázok 38: Ukážka vstupu a výstupu č.2 pri webovej metóde SortedDrillDownByAtributeDataBetweenTwoDatesSell zdroj: vlastné spracovanie 90](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014073)

[Obrázok 39: Zdrojový kód webovej metódy CalculateFinancialIndicators zdroj: vlastné spracovanie 91](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014074)

[Obrázok 40: Odoslanie testovacej požiadavky v nástroji SOAPUI zdroj: vlastné spracovanie 93](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014075)

[Obrázok 41: Príklad „konzumácie“ webovej služby pomocou Jquery funkcie ajax zdroj: vlastné spracovanie 95](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014076)

[Obrázok 42: spôsob získania čísla portu z aktuálnej url adresy 96](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014077)

[Obrázok 43: Zdrojový kód javascript funkcie showAlert zdroj: vlastné spracovanie 97](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014078)

[Obrázok 44: Ukážka oznámení na stránke zdroj: vlastné spracovanie 97](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014079)

[Obrázok 45: Zdrojový kód javascript funkcie autoCompleteInput zdroj: vlastné spracovanie 98](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014080)

[Obrázok 46: Príklad automatického dopĺňania vstupného poľa formulára zdroj: vlastné spracovanie 98](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014081)

[Obrázok 47: Hlavná obrazovka klienta zdroj: vlastné spracovanie 99](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014082)

[Obrázok 48: Obrazovka na vyhľadanie údajov o jednej knihe zdroj: vlastné spracovanie 99](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014083)

[Obrázok 49: Formulár na pridanie knihy zdroj: vlastné spracovanie 100](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014084)

[Obrázok 50: Údaje o jednej knihe v modálnom okne zdroj: vlastné spracovanie 100](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014085)

[Obrázok 51: Obrazovka s údajmi o všetkých knihách zdroj: vlastné spracovanie 101](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014086)

[Obrázok 52: Obrazovka s údajmi o všetkých transakciách zdroj: vlastné spracovanie 101](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014087)

[Obrázok 53: Obrazovka s údajmi o počtoch kníh v zadanom období a podľa hodnoty vybratého atribútu zdroj: vlastné spracovanie 102](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014088)

[Obrázok 54: Obrazovka s údajmi o predajoch kníh vo vybranom období a podľa vybraného atribútu zdroj: vlastné spracovanie 103](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014089)

[Obrázok 55: Ukážka drill down grafu zdroj: vlastné spracovanie 104](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014090)

[Obrázok 56: Obrazovka zobrazujúca hospodárske výsledky vo vybranom období zdroj: vlastné spracovanie 104](file:///D:\git_repozitare\FHI\diplomovka\moja_praca\diplomovka.docx#_Toc132014091)

**Zoznam tabuliek**

[Tabuľka 1: Popis jednotlivých HTTP metód 25](#_Toc132014109)

[Tabuľka 2: Porovnanie metód GET a POST zdroj: [24] 26](#_Toc132014110)

[Tabuľka 3: Porovnanie vybraných vlastností JSON a XML zdroj: [36] [37] 34](#_Toc132014111)

[Tabuľka 4: Porovnanie SOAP protokolu a REST architektúry zdroj: [44] [45] 40](#_Toc132014112)

[Tabuľka 5: Popis elementov v XML dokumente books.xml zdroj: vlastné spracovanie 50](#_Toc132014113)

[Tabuľka 6: Popis elementov v XML dokumente book\_transaction.xml zdroj: vlastné spracovanie 53](#_Toc132014114)

[Tabuľka 7: Popis jednotlivých komponentov riešenia vo Visual Studiu zdroj: vlastné spracovanie 60](#_Toc132014115)

**Zoznam akronymov a skratiek**

|  |  |
| --- | --- |
| HTTP | *Hypertext Transfer Protocol*/ Hypertextový prenosový protokol |
| XML | *Extensible Markup Language*/ Rozšíriteľný značkovací jazyk |
| SOAP | *Simple Object Access Protocol*/ Protokol na jednoduchý prístup k objektom |
| XSD | *XML Schema Definition/* Definícia schémy XML |
| UDDI | *Universal Description, Discovery and Integration*/ Univerzálny popis, objavovanie a integrácia |
| WSDL | *Web Services Description Language*/ Jazyk popisu webových služieb |
| API | *Application programming interface*/ Rozhranie pre programovanie aplikácií |
| CORBA | *Common Object Request Brok er Architecture* |
| RPC | *Remote procedure call*/Vzdialené volanie procedúry |
| RMI | *Remote Method Invocation*/ Vzdialené vyvolanie metódy |
| DCOM | *Distributed Component Object Model/* Distribuovaný komponentový objektový model |
| REST | *Representational state transfer* |
| IBM | *International Business Machines Corporation* |
| W3C | *World Wide Web Consortium* |
| URL | *Uniform Resource Locator*/ Jednotný vyhľadávač zdrojov |
| ODATA | *Open Data Protocol* |
| MIME | *Multipurpose Internet Mail Extensions*/ Viacúčelové rozšírenie internetovej pošty |
| JSON | *JavaScript Object Notation*/ Zápis objektov JavaScript |
| XSLT | *Extensible Stylesheet Language Transformations*/ Transformácia jazyka rozšíriteľných štýlov |
| UTF | *Unicode Transformation Format*/ Transformačný formát Unicode |
| AJAX | *Asynchronous JavaScript and XML* |
| SMTP | *Simple Mail Transfer Protocol*/ Jednoduchý protokol prenosu pošty |
| IIS | *Internet Information Services*/ Internetové informačné služby |
| LINQ | *Language Integrated Query* |

# Úvod

*„Knihy sú jedinečné prenosné kúzla.“* Tento citát spisovateľa Stephena Kinga dokonale vystihuje podstatu toho, čím sú knihy také výnimočné. Dokážu nás preniesť do iných svetov, naučiť nás nové veci a dokonca zmeniť náš pohľad na život. V dnešnej rýchlo sa rozvíjajúcej digitálnej ére zaznamenalo odvetvie online kníhkupectiev výrazný rast. Vďaka pohodlnému prezeraniu a nákupu kníh bez toho, aby človek opustil pohodlie svojho domova, sa online kníhkupectvá stávajú čoraz obľúbenejšou voľbou pre knižných nadšencov na celom svete. S týmto rastom však prichádza aj výzva a otázka pre mnohých manažérov týchto kníhkupectiev a to, že akým spôsobom efektívne spravovať a analyzovať obrovské množstvo údajov a transakcií, ktoré prostredníctvom ich platforiem vznikajú.

Jedným z viacerých riešení tejto výzvy môže implementácia webových služieb, ktoré môžu poskytnúť štandardizovaný spôsob výmeny údajov medzi rôznymi softvérovými aplikáciami bez ohľadu na použitý programovací jazyk alebo platformu. V kontexte internetového kníhkupectva sa webová služba môže použiť na umožnenie komunikácie medzi webovou stránkou a dátovou základňou kníhkupectva, čo by umožnilo bezproblémovú výmenu a následnú analýzu údajov. Cieľom tejto práce je predstavenie webových služieb a súvisiacich technológií, ich analýza v podnikateľskom prostredí, ako aj vytvorenie vlastnej webovej služby, ktorá bude svojmu klientovi poskytovať rôzne informácie o parametroch kníh.

V prvej kapitole si predstavíme pojem webové služby, prejdeme si ich históriu popíšeme si základnú architektúru a životný cyklus ich vývoja. Predstavíme si aj viacero technológií, jazykov a otvorených štandardov, ktoré s webovými službami úzko súvisia. V druhej a tretej kapitole si popíšeme cieľ a metodiku práce. V štvrtej kapitole bude vykonaná analýza v konkrétnych kníhkupectvách, kde zistíme akým spôsobom využívajú webové služby konkrétne podnikateľské subjekty. V piatej kapitole si popíšeme a zhodnotíme návrh a vybrané metódy našej webovej služby, dátovej základne a webového klienta.

# Webové služby – ich história, architektúra, najdôležitejšie otvorené štandardy a prenosové protokoly

V tejto časti práce si spravíme prehľad rôznych definícií pojmu webová služba. Nazrieme do ich histórie, pokúsime sa nájsť, až kam siahajú ich počiatky a aký bol ich vývoj až do dnešných časov. Ďalej si popíšeme architektúru webových služieb, ako aj životný cyklus vývoja webovej služby. Predstavíme si aj najdôležitejšie protokoly a otvorené štandardy, bez ktorých by webové služby nemohli existovať. Na konci kapitoly si porovnáme dva najpopulárnejšie spôsoby na prístup k webovým službám, a to protokol SOAP a architektúru REST.

## Definícia webovej služby

V prvom rade, by bolo dobré si definovať, čo samotné webové služby predstavujú a aký je ich význam. Odborná literatúra nám poskytuje bohatý výber definícií webových služieb. My sa pokúsime uviesť aspoň tie najvýstižnejšie.

Spoločnosť Gartner definuje webové služby, ako softvérový koncept a infraštruktúru, ktorá je podporovaná poprednými predajcami výpočtovej techniky (najmä firmou Microsoft a IBM) slúžiacu na stroj-stroj komunikáciu medzi aplikáciami naprieč internetovou sieťou. Webová služba predstavuje rozhranie ku konkrétnemu softvérovému komponentu nachádzajúcom sa na serveri, ku ktorému môže pristupovať iná aplikácia (ako je server, klient alebo dokonca iná webová služba), a to všetko za pomoci použitia všeobecne dostupných všadeprítomných internetových protokolov, ako je napríklad prenosový protokol Hypertext Transport Protocol (HTTP) alebo protokol na výmenu správ vo formáte XML (Extensible Markup Language) a SOAP (Simple Object Access Protocol). [1]

Definícia na portály Tutorialspoint popisuje webové služby ako súbor otvorených protokolov a štandardov používaných na výmenu údajov medzi aplikáciami. Pri webových službách ide hlavne o otvorené štandardy, ktoré určujú štruktúru prenášaných správ (XML, SOAP, XSD (XML Schema Definition )), ďalej to môžu byť UDDI (Universal Description, Discovery and Integration) registre, ktoré predstavujú štandard pre popis, publikovanie a vyhľadávanie webových služieb alebo jazyk slúžiaci na popis webových služieb WSDL (Web Services Description Language). Tieto štandardy si v ďalšom priebehu práce lepšie popíšeme. [2]

### Rozdiel medzi webovou službou a webovou aplikáciou

Webové služby bývajú niekedy nesprávne označované ako webové aplikácie. V knihe *Servisne orientovaná architektúra v procesne riadenom* *podniku* od Pavla Juríka nachádzame popísaný jeden z hlavných rozdielov medzi tradičnou webovou službou a webovou aplikáciou, a tým je absencia používateľského rozhrania. Treba si uvedomiť, že tradičné webové služby sú určené hlavne na komunikáciu viacerých aplikácií medzi sebou a nie medzi aplikáciou a používateľom (človekom). Tradičné webové služby používajú na výmenu údajov a komunikáciu s inými aplikáciami tzv. API (Application Program Interface). Toto rozhranie zabezpečuje, aby webová služba mohla prijať správu od inej webovej služby/aplikácie v štandardizovanej forme (napr. XML alebo SOAP). Prax avšak ukázala, že niektoré webové služby sú naprogramované tak, že na svoje fungovanie predsa potrebujú prijať určité vstupné údaje od používateľa. Tie sa môžu prijať zväčša pomocou nejakého formulára a prostredníctvom jedného z najčastejšie používaných typov rozhraní akými sú tenkí klienti, podnikové portály, tuční klienti alebo chytrí klienti. [3]

Webová aplikácia je väčšinou spustená na vzdialenom fyzickom serveri a používateľ k nej pristupuje pomocou klientskej aplikácie, napríklad pomocou webového prehliadača. Dá sa teda povedať, že webové aplikácie sú postavené na komunikácii typu klient/server, kde sa viaceré klientske aplikácie môžu obracať na webovú aplikáciu, ktorá vystupuje ako server. Webová služba môže na rozdiel od webovej aplikácie vystupovať aj ako server aj ako klient. To znamená, že webová služba sa môže obracať na iné webové služby so žiadosťou o poskytnutie údajov (služba je klient) a naopak, iné služby môžu žiadať túto službu o poskytnutie údajov (služba je server). [3]

Obrázok 1: Príklad klient/server komunikácie dvoch aplikácií pomocou webovej služby zdroj: [48]

### Vlastnosti webových služieb

Spoločnosť IBM popisuje vlastnosti webových služieb takto :

**Sebestačné**- na strane klienta nepotrebujeme žiadny doplňujúci softvér. Potrebný je programovací jazyk s podporou XML a HTTP klienta. Na strane servera je potrebný len HTTP a SOAP server.

**Samo-popisujúce-** jazyk WSDL poskytuje všetky informácie, ktoré potrebujeme na implementáciu webovej služby v prípade, že ide o poskytovateľa alebo na zavolanie webovej služby, ak ide o žiadateľa webovej služby.

**Modulárne-** jednoduché webové služby možno agregovať do komplexnejších webových služieb. Príkladom môže byť potreba vykonania nejakej komplikovanejšej funkcie, na ktorej vykonanie sa kompozičným spôsobom podieľa viacero služieb.

**Jazykovo nezávislé-** nebové služby môžu na výmenu údajov používať aplikácie napísané v rôznych programovacích jazykoch a bežiace na rôznych platformách.

**Bez používateľského rozhrania**- tento prístup neposkytuje žiadne grafické používateľské rozhranie. Všetko funguje na úrovni kódu.

**Princíp čiernej skrinky**- žiadateľ o službu potrebuje poznať rozhranie k webovej službe, ale nie podrobnosti o tom, ako bola implementovaná.

**Ľahká integrácia existujúcich aplikácií**- existujúce aplikácie môžu byť veľmi ľahko integrované napr. do servisne orientovanej architektúry a to tak, že sa použije webová služba ako rozhranie k existujúcej aplikácii. [4]

## Cesta k webovým službám

Web bol pred rokom 1998 jednoduchým no zároveň aj chaotickým miestom. Normy boli minimálne a implementácie webových serverov boli z veľkej časti proprietárne. Maximum toho čo, v tej dobe dokázali vykonať prehliadače bolo, že odoslali synchrónnym spôsobom formulár.

### CORBA, RMI, DCOM

Koncom 90-tych rokov sa stalo populárnym množstvo technológií umožňujúcich vzdialené volania procedúr (RPC) z jedného systému do druhého**. CORBA** (Common Object Request Broker Architecture) umožnila komunikáciu medzi softvérom napísaným v rôznych jazykoch a spusteným na rôznych platformách. **RMI** (Java Remote Method Invocation) v podstate ponúkala niečo podobné, až na to, že bola založená na programovacom jazyku Java. Pokladá sa za objektovo orientovanú alternatívu volania vzdialených procedúr. RMI bol o niečo jednoduchší na implementáciu, ale tiež bol obmedzený len na komunikáciu medzi programami napísanými v Jave.

Na strane Microsoftu **DCOM** (Distributed Component Object Model) umožnil interakciu natívnych programov na operačnom systéme Windows. DCOM bol hlavným konkurentom CORBA. Zástancovia oboch technológií si mysleli, že sa jedného dňa stanú modelom pre opätovné použitie kódu a služieb cez internet. Avšak ťažkosti spojené s fungovaním technológií cez internetové brány firewall a na neznámych a nezabezpečených počítačoch znamenali, že bežné požiadavky HTTP v kombinácií s webovými prehliadačmi zvíťazili nad oboma. [5] [6]

Všetky tieto konkurenčné štandardy mali aspoň niekoľko hlavných obmedzení:

1. boli obmedzené, pokiaľ ide o platformy, s ktorými mohli spolupracovať,

2. bola to veľká výzva, aby fungovali bezpečne cez internetové brány firewall.

Revolúciu v tom čase priniesol HTTP protokol, ktorý zabezpečil jednotný spôsob komunikácie medzi servermi nachádzajúcimi sa naprieč svetom. Ľudia začali experimentovať a snažili sa nájsť spôsob zlepšenia komunikácie medzi strojmi, súčasne s rýchlym vývojom technológie prehliadačov a serverov. [7]

### XML-RPC, SOAP, REST, GRAPHQL

V tom čase nabrala na popularite Java, a to hlavne vďaka jej nezávislosti na platforme. To zrejme podnietilo Microsoft, aby prišiel so štandardom nezávislým na platforme, ktorý by ich potenciálne mohol pomôcť dostať sa z kúta, do ktorého boli pomaly tlačení. Novú nádej mal priniesť protokol **XML-RPC**, ktorý vytvoril v roku 1998 Dave Winer z UserLand Software a Microsoft. Ten už v tomto čase pracoval na protokole SOAP, ktorý avšak pre vnútornú politiku vo firme Microsoft vyšiel trochu neskôr. Údaje sú v tomto protokole zapuzdrené pomocou značkovacieho jazyka XML a prenášajú sa vďaka protokolu HTTP. To umožňovalo ľahkú komunikáciu aplikácií, ktoré bežia na rôznych operačných systémoch alebo sú napísané v rôznych programovacích jazykoch. XML-RPC je veľmi jednoduchý protokol, ktorý definuje iba niekoľko dátových typov a príkazov – a vďaka tejto jednoduchosti sa stal veľmi populárnym. V dnešnej dobe je už síce vývoj tohto projektu ukončený, avšak ten predstavoval predlohu pre protokol **SOAP**. [8]

Neskôr sa začal používať protokol **JSON-RPC,** ktorý na prenos údajov používa formát JSON (JavaScript Object Notation). Najnovšia verzia RPC vyvinutá spoločnosťou Google v roku 2016 je gRPC.

SOAP je prirodzené rozšírenie XML-RPC, ktoré poskytuje vyššiu úroveň štruktúry a podpory pre dátové typy a sémantiku operácií. Verzia SOAP 0.9 bola predložená 13. septembra 1999 na verejné posúdenie Komisii pre technickú stránku internetu IETF (Internet Engineering Task Force) ako internetový verejný návrh. Z drobnými úpravami uzrela v decembri toho istého roku svetlo sveta verzia SOAP 1.0. Po nejakom čase sa SOAPu podarilo nájsť silného spojenca vo firme IBM. Microsoft a IBM spoločne presadili špecifikáciu pre SOAP 1.1, ktorá sa 8. mája 2000 stala oficiálnym odporúčaním W3C (World Wide Web Consortium). Podpora IBM bola neočakávanou a osviežujúcou zmenou. Okrem toho SOAP 1.1 špecifikácia bola oveľa modulárnejšia a rozšíriteľnejšia, čím sa eliminovali určité obavy, že podpora SOAP zahŕňa podporu proprietárnej technológie spoločnosti Microsoft. IBM okamžite vydala implementáciu Java SOAP, ktorá bola následne implementovaná do projektu Apache XML, ktorý bol vyvíjaný ako open source. To presvedčilo aj tých najväčších skeptikov, že SOAP je niečo, čomu treba venovať pozornosť. V júni 2003 sa verzia SOAP 1.2 stala druhým odporúčaním W3C. [9]

SOAP mal striktné pravidlá, čo väčšina ľudí považovala za dobrú vec. Vývojári SOAP vedeli, že vďaka tejto štandardizácii boli ťažké problémy s nekonzistenciou dát pre nich vyriešené, takže sa mohli viac sústrediť na implementačné detaily. Nakoniec SOAP pomohlo vývojárom vytvoriť API, programovacie rozhrania, ktoré používateľom umožňuje získavať a aktualizovať údaje z webových serverov. Čoskoro niektoré z najväčších organizácií, ako sú Oracle, HP alebo Sun spustili vďaka SOAPu svoje vlastné API. To umožnilo každému vývojárovi (nielen internému) na planéte pripojiť sa k týmto stránkam programovo a získať prístup k ich údajom.

Našli sa aj ľudia, ako bol napríklad Roy Fielding, ktorým SOAP úplne nevyhovoval. Keď bol SOAP vydaný, Fielding pracoval s Timom Berners-Lee na najnovšej špecifikácii HTTP 1.1. Zároveň vyvinul vlastný súbor princípov pre webové služby s názvom Representational State Transfer alebo REST. Prvú špecifikáciu REST publikoval ako svoju doktorandskú dizertačnú prácu na UC Irvine v roku 2000.

V skutočnosti REST nebol úplný súbor technológií, ale skôr súbor princípov dizajnu, ktoré sa snažili využiť vstavané metódy HTTP (metódy, o ktorých ste možno počuli, ako GET, POST a DELETE). Hlavnou myšlienkou REST bolo, že pre každý údaj zostane adresa URL rovnaká, ale operácia sa zmení v závislosti od použitej metódy. Napríklad dopytom „http://yoursite.com/posts“ na GET sa môže vrátiť jednoduchý zoznam príspevkov, ale žiadosť POST na tú istú adresu URL by namiesto toho vytvorila nový príspevok.

Fielding a jeho priaznivci tvrdili, že REST je jednoduchší a elegantnejší, vytvorený špeciálne pre web. Zdôraznili flexibilitu systémov vzhľadom na prísne štandardy SOAP. Priaznivci SOAPU na druhej strane považovali REST za príliš obrovské zjednodušenie. [10]

V roku 2007 vyšiel OData (Open Data Protocol), ktorý predstavuje normu OASIS schválená ISO/IEC, ktorá definuje súbor najlepších postupov pre vytváranie a používanie REST API.

Obrázok 2: Komunikačné protokoly na časovej osi zdroj: [11]

Dnes niektorí používatelia API označujú REST ako *„Rest in peace“* (odpočívaj v pokoji) a fandia GraphQL (vytvorený firmou Facebook v roku 2012 a vypustený ako open source v roku 2015), zatiaľ čo pred desiatimi rokmi to bol opačný príbeh, keď použitie RESTu začalo dominovať nad SOAPom. Problém s týmito názormi je, že ide o jednostranný výber samotnej technológie namiesto toho, aby zvažovali, aké sú skutočné vlastnosti a charakteristiky a v akej situácii je vhodnejšie, ktorú technológiu použiť. Popísať tieto rozdiely sa pokúsime v poslednej podkapitole tejto časti práce. [11]

## Architektúra webovej služby

Architektúra webových služieb je založená na interakcii medzi troma rolami. Tieto roly predstavuje:

1. Poskytovateľ služby.
2. Register služieb.
3. Žiadateľ služby.

**Poskytovateľ služby**- Z obchodného hľadiska ide o podnik alebo osobu, ktorá je vlastníkom služby. Z architektonického hľadiska je to platforma, ktorá zabezpečuje prístup k službe. [12]

Podľa [3] je *„Webová služba prijíma rolu poskytovateľa služieb (t. j. rolu servera) za týchto podmienok:*

* *je volaná externým zdrojom, akým je napr. žiadateľ služieb (t. j. klient),*
* *poskytuje opis služby, ktorý obsahuje informácie o jej funkciách a správaní.“*

**Register služieb**-ide v podstate o register popisov služieb, ktoré do neho zverejňujú poskytovatelia služieb. Žiadatelia o služby v ňom hľadajú dostupné služby a získavajú informáciu, ako sa na službu napojiť (binding information). [12]

**Žiadateľ služby**- z obchodného hľadiska ide o podnik alebo osobu, ktorá si vyžaduje splnenie určitej funkcie. Z architektonického hľadiska ide o aplikáciu, ktorá hľadá alebo iniciuje interakciu so službou. Rolu žiadateľa služby môže predstavovať prehliadač riadený osobou alebo program bez používateľského rozhrania, napríklad iná webová služba. [12]

Podľa [3] je *„Webová služba prijíma rolu žiadateľa služieb (t.j. klienta) za nasledujúcich podmienok:*

* *hľadá a identifikuje najlepšieho poskytovateľa služieb na základe dostupných opisov služieb,*
* *volá poskytovateľa služieb odoslaním správy,*
* *ak od poskytovateľa služieb požaduje zaslanie spätnej správy s výsledkom, musí sprístupniť svoj opis, aby poskytovateľ služieb vedel, v akom tvare má byť zapísaná táto správa ,a tiež kam a akým spôsobom ju treba poslať.“*

Obrázok 3: Architektúra webovej služby zdroj: [12]

## Životný cyklus vývoja webovej služby

**Fáza požiadaviek**- cieľom fázy požiadaviek je porozumieť obchodným požiadavkám daného podniku, keďže tieto úzko súvisia s požiadavkami na webové služby. Treba jasne definovať pre ktoré entity, procesy a činnosti v podniku bude prebiehať automatizácia pomocou webových služieb. Samotný proces zbierania požiadaviek od zákazníkov, budúcich používateľov a ostatných zainteresovaných strán vykonáva analytik. Analytik by mal tieto požiadavky interpretovať, konsolidovať a oznámiť vývojovému tímu. Požiadavky by mali byť zoskupené v centralizovanom úložisku, kde ich možno prezerať, uprednostňovať a získavať z nich potrebné informácie. [13]

**Fáza analýzy**- účelom fázy analýzy je vytvoriť pre webové služby konceptuálne modely, ktorým môže tím technického vývoja porozumieť. V tejto fáze sa taktiež definujú zmluvy o rozhraní webových služieb (web service interface contracts). Ide v podstate o súbor metadát, ktoré opisujú rôzne aspekty webovej služby, napr. aké funkcie daná služba vykonáva, aké správy musí služba prijať, aby mohla dané funkcie vykonať alebo v akej dátovej štruktúre musí služba danú správu prijať. [13]

**Fáza návrhu-** v tejto fáze sa robí detailný návrh webových služieb. Dizajnéri konkretizujú návrh zmluvy o rozhraní webových služieb, ktorá bola identifikovaná vo fáze analýzy. [13]

**Fáza kódovania-** priebeh fázy kódovania a debugovania je dosť podobný, ako pri vývoji iných softvérových komponentov. Vývoj webových služieb môže prebehnúť tak, že sa vytvorí úplné nová webová služba, transformuje sa existujúca aplikácia na webovú službu alebo sa poskladá nová webová služba z iných webových služieb a aplikácií. [12] [13]

**Testovacia fáza-** v tejto fáze testery vykonávajú testovanie interoperability medzi webovou službou a klientskym programom. Kontrola kvality webovej služby by mala zahŕňať funkčné a nefunkčné testy. Netreba zabudnúť ani na záťažové a bezpečnostné testy, ktoré sú taktiež veľmi dôležité. Pre testerov webových služieb boli vyvinuté testovacie nástroje, ktoré im túto prácu môžu uľahčiť. Ako príklad môžeme uviesť komerčný nastroj SOAPUI, ktorý slúži na testovanie SOAP a RESTových služieb. Pomocou týchto nástrojov je možné vytvoriť simuláciu služby bez ešte vytvoreného komplexného riešenia. Tester si môže simulovať požiadavky (requesty), ktoré chce otestovať a pripraviť na nich množstvo rôznych odpovedí (response). Tieto odpovede môžu obsahovať skripty, vlastné hlavičky HTTP, prílohy a ďalší obsah. [14] [13]

**Fáza nasadenia-** účelom fázy nasadenia je zabezpečiť správne nasadenie webovej služby v distribuovanom systéme. Vykonáva sa po testovacej fáze. Primárnou úlohou nasadenia je zabezpečiť, aby bola webová služba správne nakonfigurovaná. V tejto fáze sa vykonávajú aj ďalšie voliteľné úlohy, ako je špecifikácia a registrácia webovej služby v registri UDDI. [13]



Obrázok 4: Životný cyklus vývoja webovej služby zdroj: [13]

## HTTP protokol

Protokol HTTP je jeden z najčastejšie využívaných prenosových protokolov na internete. V čase písania tejto práce je najnovšia verzia HTTP/3, ktorá vyšla v roku 2022, využívaná 25 percentami všetkých webových stránok [15] a verzia HTTP/2 z roku 2015 je využívaná 39,8 percentami všetkých webových stránok. [16] Kľúčovú rolu hrá aj pri komunikácii webových služieb, a preto si ho v tejto časti lepšie opíšeme.

V  [17] sú rôzne verzie HTTP protokolov definované ako rodina bezstavových, na aplikačnej vrstve pracujúcich protokolov, ktoré fungujú na princípe požiadavka/odpoveď. Tieto protokoly zdieľajú všeobecné rozhranie, majú rozšíriteľnú sémantiku a ich samo-popisné správy umožňujú flexibilnú interakciu s hypertextovými informačnými systémami založenými na sieti.

### Vlastnosti HTTP

**Bezstavovosť**- HTTP je definovaný ako bezstavový protokol, čo znamená, že sémantiku každej správy s požiadavkou možno chápať izolovane, a že vzťah medzi pripojeniami a správami nemá žiadny vplyv na interpretáciu týchto správ. V dôsledku toho server nesmie predpokladať, že dve požiadavky na to isté pripojenie pochádzajú od rovnakého užívateľského agenta, pokiaľ spojenie nie je zabezpečené a špecifické pre daného agenta. [17] Táto vlastnosť je problematická pri e-komerčných webových aplikáciách akými sú napríklad E-shopy, kde používateľ potrebuje s aplikáciou interagovať pomocou viacerých koherentných požiadaviek. Tento problém je v HTTP vyriešený pomocou takzvaných HTTP cookies, ktoré rozširujú hlavičku správy a ktoré môžu na rozdiel od jadra HTTP používať stavové relácie. [18]

**Nezávislosť na médiu**- HTTP je nezávislý od média. To znamená, že akýkoľvek typ údajov môže byť odoslaný prostredníctvom HTTP, pokiaľ klient aj server vedia, ako s obsahom údajov zaobchádzať. Je potrebné, aby klient aj server špecifikovali typ obsahu pomocou vhodného typu MIME (Multipurpose Internet Mail Extensions).

### HTTP správy

Ako už bolo spomínané vyššie, existujú dva typy HTTP správ a to HTTP požiadavka (request ) a HTTP odpoveď (response). [19]

### HTTP požiadavka

Požiadavky HTTP sú správy odoslané klientom na spustenie akcie na serveri. Ich prvý riadok obsahuje tri elementy a to:

1. **Metódu HTTP**- zapísanú ako sloveso (napríklad GET, PUT alebo POST) alebo podstatné meno (napríklad HEAD alebo OPTIONS), ktorá popisuje akciu, ktorá sa má vykonať.
2. **Cieľ požiadavky**- zvyčajne URL alebo absolútna cesta protokolu, portu a domény. Sú zvyčajne charakterizované kontextom požiadavky. Formát tohto cieľa požiadavky je rozličný pri rôznych metódach HTTP.
3. **Verzia http**- ktorá definuje štruktúru zostávajúcej správy a funguje ako indikátor očakávanej verzie, ktorá sa má použiť pre odpoveď. [20]

Ďalej vieme HTTP požiadavku rozdeliť na hlavičky a telo.

V HTTP hlavičkách vieme serveru poskytnúť dodatočné informácie, ako napr. hlavička user agent, ktorá umožňuje identifikovať aplikáciu, operačný systém alebo webový prehliadač používateľského agenta. V hlavičkách vieme definovať aj akým formátom klient rozumie alebo v akých jazykoch alebo v akom kódovaní akceptuje odpoveď. Môže obsahovať aj mnohé ďalšie atribúty, ktorým sa v tejto práci nebudeme venovať.

Telo požiadavky je časť požiadavky HTTP, ktorou je možné poslať dodatočný obsah na server. Napríklad typ súboru JSON alebo XML. Telo HTTP požiadavky sa nemusí nutne využívať pre všetky HTTP metódy. Typickou metódou, kde je telo využité je POST metóda. [20]

Obrázok 5: Príklad HTTP požiadavky zdroj: [20]

### HTTP odpoveď

Po spracovaní požiadavky odošle server klientovi HTTP odpoveď. Cieľom odpovede je poskytnúť klientovi údaje, ktoré požadoval alebo informovať klienta o vykonaní akcie, ktorú požadoval. Klient je taktiež informovaný o tom, že pri spracovaní jeho požiadavky došlo k chybe. Odpoveď HTTP obsahuje:

1. stavový riadok,
2. sériu hlavičiek HTTP alebo polí hlavičiek,
3. telo správy, ktoré je zvyčajne potrebné. [21]

Začiatočný riadok odpovede HTTP, nazývaný stavový riadok, obsahuje nasledujúce informácie:

1. verzia protokolu, zvyčajne HTTP/1.1,
2. stavový kód označujúci úspech alebo zlyhanie požiadavky. Bežné stavovékódy sú 200, 404 alebo 302,
3. stavový text- stručný, čisto informačný, textový popis stavového kódu, ktorý pomôže človeku pochopiť správu HTTP.

Typický stavový riadok môže vyzerať takto: HTTP/1.1 404 Not Found. [20]

Hlavičky HTTP pre odpoveď servera obsahujú informácie, ktoré môže klient použiť na zistenie ďalších informácií o odpovedi a o serveri, ktorý ju odoslal. Tieto informácie môžu pomôcť klientovi so zobrazením odpovede používateľovi, s ukladaním odpovede do trvalého úložiska alebo do vyrovnávacej pamäte pre budúce použitie a pri vytváraní ďalších požiadaviek na server v budúcnosti. [21]

Obrázok 6: Príklad HTTP požiadavky zdroj: [20]

### HTTP metódy

V nasledujúcej tabuľke si v krátkosti predstavíme a popíšeme všetky HTTP metódy využívané pri HTTP požiadavkách a v ďalšej tabuľke si porovnáme dve najčastejšie používané metódy a to GET a POST.

Tabuľka 1: Popis jednotlivých HTTP metód

|  |  |
| --- | --- |
| Metóda | Popis |
| GET | Metóda GET sa používa na získanie informácií zo servera pomocou daného URI. HTTP požiadavky využívajúce GET by mali údaje iba získavať a nemali by mať na nich žiadny iný vplyv. [17] |
| HEAD | Metóda HEAD požaduje odpoveď identickú s požiadavkou GET, ale bez tela odpovede. [20] |
| POST | Požiadavka POST sa používa na odoslanie údajov na server, napr. nahratie informácií o zákazníkovi, nahranie súboru pomocou HTML formulára. [20] |
| PUT | Nahradí všetky aktuálne reprezentácie cieľového zdroja nahraným obsahom. [17] |
| DELETE | Odstráni všetky aktuálne reprezentácie cieľového zdroja pomocou daného URI. [17] |
| CONNECT | Metóda HTTP CONNECT sa používa na vytvorenie tunela HTTP cez proxy server. Odoslaním požiadavky HTTP CONNECT klient požiada proxy server o presmerovanie spojenia TCP na požadované miesto určenia.  Proxy server nadviaže spojenie s požadovaným serverom v mene klienta a po nadviazaní spojenia proxy server pokračuje v sprostredkovaní toku TCP až po klienta. [22] |
| OPTIONS | Popisuje možnosti komunikácie s cieľovým serverom. Pomocou tejto metódy je možné napríklad zistiť, ktoré metódy server, na ktorý klient posiela požiadavku podporuje. [23] |
| TRACE | Požiadavka HTTP tohto typu sa používa na diagnostické účely a nemá telo správy. [17] |

Tabuľka 2: Porovnanie metód GET a POST zdroj: [24]

|  |  |  |
| --- | --- | --- |
|  | GET | POST |
| **Správanie pri znovu- načítaní stránky alebo použití tlačidla spať v prehliadači.** | Nemá účinok. | Údaje budú znova odoslané (prehliadač by mal používateľa upozorniť, že údaje sa budú znova odosielať). |
| **Kešovanie** | Je kešovaná. | Nie je kešovaná. |
| **Parametre v histórii** | Sú uložené. | Nie sú uložené. |
| **Obmedzenia dĺžky údajov** | Je obmedzená (maximálna dĺžka adresy URL je 2 048 znakov). | Bez obmedzení. |
| **Obmedzenia typu údajov** | Povolené sú len znaky ASCII. | Žiadne obmedzenia, povolené sú aj binárne dáta. |
| **Bezpečnosť** | V porovnaní s POST menej bezpečný, pretože odoslané údaje sú súčasťou adresy URL.  GET by sa nemal nikdy používať pri odosielaní hesiel alebo iných citlivých informácií! | POST je o niečo bezpečnejší ako GET, pretože parametre nie sú uložené v histórii prehliadača ani v logoch webového servera. |
| **Viditeľnosť dát** | Údaje sú viditeľné pre každého v adrese URL. | Údaje sa v adrese URL nezobrazujú. |

### HTTP stavové kódy

Všetky stavové kódy odpovedí HTTP delíme do piatich tried alebo kategórií. Prvá číslica stavového kódu definuje triedu odpovede, zatiaľ čo posledné dve predstavujú jeho bližšiu špecifikáciu.

Podľa štandardu rozlišujeme päť tried a to:

**1xx Informačná odpoveď**- webový server odpovedá stavovým kódom 1xx, keď server spracúva prijatú požiadavku. Tieto kódy sú akýmsi potvrdením pre klienta. Odpoveď 1xx sa zvyčajne skladá zo stavového riadku HTTP, hlavičky a je ukončená prázdnym riadkom. Stavové kódy 1xx sú definované len vo verzii HTTP/1.1 a predchádzajúca verzia HTTP/1.0 ich nepodporuje. [25]

**2xx Úspešná**- stavové kódy 2xx znamenajú, že komunikácia prebehla úspešne a webový server mohol spracovať požiadavku prijatú z klientskeho prehliadača. [26]

**3xx** **Presmerovanie-** klient musí vykonať ďalšie kroky na dokončenie žiadosti. Táto skupina stavových kódov označuje, že používateľský agent musí vykonať ďalšiu akciu na dokončenie požiadavky. Požadovanú akciu môže vykonať používateľský agent bez interakcie s klientom len vtedy, ak je metóda použitá v požiadavke GET alebo HEAD. [27]

**4xx** **Chyba klienta**- trieda stavového kódu 4xx (Chyba klienta) označuje, že klient zrejme urobil chybu. S výnimkou odpovede na požiadavku HEAD server musí poslať reprezentáciu obsahujúcu vysvetlenie chybovej situácie a informáciu, či ide o dočasný alebo trvalý stav. Tieto stavové kódy sa vzťahujú na akúkoľvek metódu požiadavky. [17]

**5xx** **Chyba servera**- stavový kód triedy 5xx označuje, že server si je vedomý chyby alebo, že nie je schopný vykonať požadovanú metódu. S výnimkou odpovede na požiadavku HEAD server musí poslať reprezentáciu obsahujúcu vysvetlenie chybovej situácie a informáciu, či ide o dočasný alebo trvalý stav. Tieto stavové kódy sa taktiež vzťahujú na akúkoľvek metódu požiadavky. [17]

## XML

V nasledujúcich podkapitolách tejto časti diplomovej práce si predstavíme a popíšeme otvorené štandardy akými sú XML, WSDL, SOAP alebo UDDI, ktoré tvoria základ konceptu využitia webových služieb.

Prvým otvoreným štandardom, ktorý si predstavíme je jazyk XML (Extensible Markup Language), ktorý prekladáme ako rozšíriteľný značkovací jazyk. Tento jazyk vznikol už koncom 60. rokov 20. storočia, avšak jeho popularita vzrástla až na konci 90. rokov 20. storočia, kedy dochádzalo k rozsiahlemu rozvoju internetu. [3] V [3] je XML definovaný ako „*jazyk slúžiaci na platformovo nezávislé uchovávanie údajov, ktoré môžu byť ľahko prenášané prostredníctvom internetových prenosových protokolov*.“ Pri XML je dôležité si uvedomiť, že nejde o programovací jazyk akým je napríklad C# alebo jazyk na zobrazovanie údajov na obrazovke, akým je napríklad HTML. Ide skôr o jazyk, ktorý zachytáva význam údajov a umožňuje popisovať vzájomné vzťahy medzi údajmi. [3]

### Vlastnosti XML

Základné vlastnosti XML by sme mohli opísať takto:

**Ide o rozšíriteľný a ľudsky čitateľný jazyk**- jazyk XML umožňuje jeho používateľovi vytvoriť vlastné popisné značky. Jazyk je navrhnutý tak, aby bol čitateľný človeku aj stroju. Na zlepšenie prezentácie údajov pre ľudskú čitateľnosť sa používa jazyk XSLT (Extensible Stylesheet Language Transformation), ktorý dokáže XML pretransformovať na HTML stránku. [28]

**Je nezávislý na platforme a jazyku –** to znamená, že ho možno používať v akomkoľvek operačnom systéme alebo aplikáciou naprogramovanou v ľubovoľnom jazyku. [28] Táto vlastnosť XML je veľmi dôležitá, pretože umožňuje jednoduchú a efektívnu výmenu údajov medzi rôznymi systémami. Podľa [3] „*Nezávislosť na platforme podporuje najmä skutočnosť, že obsahom XML dokumentu sú textové informácie (XML je textový dokument) s tým, že informácie o kódovaní znakov sú uvedené v hlavičke dokumentu.“*

**Má pevnú syntax-** jazyk XML má presne zadefinované pravidlá zápisu značiek. To či XML dokument spĺňa všetky pravidlá formátovania sa dá overiť pomocou softvéru, ktorý nazývame validátor. [3]

### Základná syntax a pravidlá

**XML Element –** predstavuje základný stavebný prvok každého XML dokumentu. XML element predstavuje všetko od počiatočnej značky (<>) elementu (vrátane) po konečnú značku (</>) elementu. Každý element môže obsahovať text, atribúty, iné elementy alebo mix týchto prvkov. Príklad elementu môže byť:

<cena>30</cena>

Pravidla pre element:

* názvy elementov rozlišujú malé a veľké písmená,
* názvy elementov musia začínať písmenom alebo podčiarkovníkom,
* názvy elementov nemôžu začínať slovom xml alebo jeho inou variantou napr. Xml a pod.,
* názvy elementov nesmú obsahovať medzery. [29] [3]

**XML Atribút-** atribúty sú súčasťou elementov XML. Element môže mať viacero jedinečných atribútov. Atribút poskytuje viac informácií o elementoch. Presnejšie povedané, definujú vlastnosti elementov. Atribút XML je vždy dvojica názov-hodnota, pričom hodnota sa priraďuje pomocou znaku „=“ a hodnota musí byť zapísaná medzi apostrofmi alebo úvodzovkami. Atribúty sa smú používať, len v začiatočných značkách elementov. Príkladom XML atribútu môže byť atribút mena, ktorý hovorí v akej mene je daná cena zaevidovaná. Príklad XML elementu s atribútom môže vyzerať nasledovne [28] [3]:

  <cena mena="euro">30<cena>

**XML strom-** XML dokumenty tvoria hierarchickú štruktúru, niekedy nazývanú aj ako strom XML dokumentu. Na najvyššom stupni tejto hierarchie sa nachádza koreňový element. Každý XML dokument musí mať práve jeden koreňový element. Všetky elementy, ktoré sa nachádzajú pod koreňovým elementom označujeme ako detské elementy koreňového elementu. Každý element môže obsahovať vnorené elementy. Element, ktorý obsahuje vnorené elementy označujeme ako rodičovský element. Elementy, ktoré sa nachádzajú na rovnakej úrovni označujeme ako súrodenecké elementy. [30]



Obrázok 7: Príklad stromovej štruktúry XML dokumentu zdroj: [30]

Každý dokument XML môže mať voliteľnú XML deklaráciu. Tá sa musí nachádzať v prvom riadku XML dokumentu a zapisuje sa do elementu <xml> [31]

<?xml version="1.0" encoding="UTF-8"?>

V elemente <xml> sa ďalej môže nachádzať atribút *version*, ktorý nám hovorí, akú verziu XML bude dokumentu používať. Väčšinou sa používa verzia 1.0, ale existuje aj verzia 1.1. Pomocou atribútu *encoding* nastavíme podľa akej znakovej sady bude XML dokument kódovaný. [3]

Obrázok 8: Príklad XML dokumentu zdroj: vlastné spracovanie

<?xml version="1.0" encoding="UTF-8"?>

<zoznamStudentov>

 <student id="1">

  <meno>Marek </meno>

  <priezvisko>Berith</priezvisko>

  <certifikat>áno</certifikat>

  <body>

   <programovanie>100</programovanie>

   <matematika>70</matematika>

   <statistika>90</statistika>

  </body>

 </student>

 <student id="2">

  <meno>Matúš </meno>

  <priezvisko>Pešta</priezvisko>

  <certifikat>áno</certifikat>

  <body>

   <programovanie>80</programovanie>

   <matematika>90</matematika>

   <statistika>85</statistika>

  </body>

 </student>

</zoznamStudentov>

Tento úryvok kódu ukazuje validný XML dokument, ktorý obsahuje údaje o dvoch študentoch a výsledkoch ich skúšok.

**XML menné priestory**- menné priestory XML sa používajú na to, aby všetky pomenovania elementov a atribútov v XML dokumente boli jednoznačne rozlíšiteľné. Sú definované v odporúčaní W3C. Inštancia XML môže obsahovať názvy elementov alebo atribútov z viac ako jednej XML schémy definovanej pomocou jazyka XSD. [32] Podľa [3] „*XML schéma zapísaná prostredníctvom jazyka XSD predstavuje určitý predpis, resp. šablónu na vytvorenie XML dokumentu a tento dokument je implementáciou tejto šablóny nesúcou konkrétne údaje podľa štruktúry a dátových typov stanovených v schéme*.“ Ak je v XML dokumente každej XML schéme pridelený menný priestor a prefix, tak možno vyriešiť nejednoznačnosť medzi rovnako pomenovanými elementami alebo atribútmi.

Jednoduchým príkladom môže byť inštancia XML, ktorá obsahuje odkazy na XML schému zákazníka a XML schému objednaný produkt. Prvok zákazník aj prvok výrobok, by mohli mať detský element s názvom ID. Tu by nastal problém, pretože pri vytvorení elementu ID, by nebolo jasné či ide o ID s XML schémy zákazník alebo z XML schémy objednaný produkt. Konfliktom názvov v XML dokumentoch sa dá ľahko vyhnúť použitím prefixu menného priestoru. Menný priestor môže byť definovaný atribútom *xmlns* v počiatočnom tagu elementu. Deklarácia menného priestoru má nasledujúcu syntax: xmlns:prefix="URI". Nasledujúci úryvok XML kódu demonštruje spôsob, akým by sa dal riešiť skorej spomínaný problém s nejasnosťou ID elementu. Pomocou atribútu *xmlns* sme si zadefinovali názvy a prefixy menných priestorov pre *zákaznika(:z)* a *objednavky(:o)* a prefix (*:xsi)* je pre základný menný priestor, ktorý obsahuje definície kľúčových slov jazyka XSD. Atribút *schemalocation* obsahuje adresy XSD schém, podľa ktorých má byť tento dokument validovaný. [32] [33] [3]

<?xml version="1.0" encoding="UTF-8"?>

Obrázok 9: Príklad použitia XML menných priestorov zdroj: vlastné spracovanie

<z:Zakaznik xmlns:z="https://.../Zakaznici"

 xmlns:o="https://....com /Objednavky"

 xmlns:xsi=" "http://www.w3.org/ 2001/XMLSchema"" xsi:schemaLocation=

     "https://.../schemas/Zakaznici.xsd

      https://.../schemas/Objednavky.xsd">

 <z:ID>1</z:ID>

 <z:meno>Alexander</z:meno>

 <z:priezvisko>Kling</z:priezvisko>

 <z:vek>25</z:vek>

 <o:Objednavka>

  <o:ID>25</o:ID>

  <o:tovar>sirup-cola</o:tovar>

  <o:mnoztvo>5</o:mnozstvo>

 </o:Objednavka>

</z:Zakaznik>

## JSON

Po tom, ako sme si definovali značkovací jazyk XML, tak si definujeme ešte jeden formát zápisu dát, s ktorým sa aj v neskoršej časti tejto práce stretneme a to formát JSON (JavaScript Object Notation). JSON je odľahčený formát založený na dátových typoch programovacieho jazyka JavaScript. Formát JSON je syntakticky podobný kódu, ktorým sa vytvárajú objekty v programovacom jazyku JavaScript. Z tohto dôvodu môže program v jazyku JavaScript ľahko konvertovať údaje JSON na objekty JavaScript. Na túto konverziu má Javascript v sebe zabudovanú statickú metódu *JSON.parse()*. Pre opačnú operáciu sa využíva zabudovaná statická metóda *JSON. stringify(),* ktorá prekonvertuje javascriptový objekt na JSON reťazec. Keďže formát JSON je len textový, údaje v ňom zapísané sa dajú ľahko posielať medzi počítačmi a používať v akomkoľvek programovacom jazyku. Vo svojej podstate sú dokumenty JSON slovníky pozostávajúce z dvojíc kľúč-hodnota, kde hodnota môže byť opäť dokument JSON, čo umožňuje ľubovoľnú úroveň vnorenia. Vďaka svojej jednoduchosti a ľahkej čitateľnosti pre ľudí aj stroje, sa JSON rýchlo stáva jedným z najpopulárnejších formátov na výmenu údajov na webe. Je to zrejmé najmä pri webových službách komunikujúcich so svojimi používateľmi prostredníctvom rozhrania API, keďže JSON je v súčasnosti prevládajúcim formátom na odosielanie požiadaviek a odpovedí API, prostredníctvom HTTP. Okrem toho sa často používa formát JSON v databázových systémoch postavených na paradigme NoSQL alebo grafových databázach. [34]

## JSON základná syntax

Syntax JSON je odvodená zo syntaxe zápisu objektu JavaScript:

* údaje sú v pároch názov/hodnota,
* údaje sú oddelené čiarkami,
* objekty sa nachádzajú v kučeravých zátvorkách,
* hranaté zátvorky obsahujú polia.

Nasledujúci úryvok JSON kódu obsahuje údaje o výsledkoch skúšok 2 študentov. Tieto údaje sme vytvorili pretransformovaním skorej vytvoreného XML dokumentu do formátu JSON. [35]

Obrázok 10: Príklad JSON kódu zdroj: vlastné spracovanie

{

  "zoznamStudentov": {

    "student": [

      {

        "meno": "Marek",

        "priezvisko": "Berith",

        "certifikat": "áno",

        "body": {

          "programovanie": 100,

          "matematika": 70,

          "statistika": 90

        }

      },

      {

        "meno": "Matúš",

        "priezvisko": "Pešta",

        "certifikat": "áno",

        "body": {

          "programovanie": 80,

          "matematika": 90,

          "statistika": 85

        }

      }

    ]

  }

}

### JSON vs XML

V nasledujúcej tabuľke si porovnáme oba jazyky.

Tabuľka 3: Porovnanie vybraných vlastností JSON a XML zdroj: [36] [37]

|  |  |  |
| --- | --- | --- |
|  | JSON | XML |
| **Podporované formáty dát** | JSON podporuje len textový a číselný typ údajov. | XML podporuje rôzne typy údajov, ako sú čísla, text, obrázky, tabuľky, grafy atď. |
| **Podporované znakové sady** | Podporuje iba kódovanie UTF8, UTF16, UTF32. | Podporuje rôzne typy kódovania. |
| **Komentáre** | Nepodporuje komentáre. | Podporuje komentáre. |
| **Menné priestory** | Neposkytuje žiadnu podporu pre menné priestory. | Podporuje menné priestory. |
| **Rýchlosť** | JSON je rýchlejší oproti XML, pretože ukladanie rovnakého množstva informácií zaberá menej pamäte. JSON používa iba jednobajtové znaky pre svoje reťazcové dátové typy. | Pomalší pre reťazcové dátové typy. Používa dvojbajtové znaky. |
| **Serializácia a deserializácia v Javascripte** | Deserializácia v Javascripte je veľmi rýchla. | Deserializácia v Javascripte je pomalšia. |
| **Javascript a AJAX** | Väčšina Javascriptových knižníc  a AJAX nástroje majú dobrú  podporu JSON. | Nástroje AJAX nemajú až takú silnú podporu. |

## WSDL

WSDL (Web Services Description Language) je skratka pre jazyk na opis webových služieb. Každý WSDL dokument musí predstavovať validný XML dokument. Je to štandardný formát na opis webovej služby. WSDL spoločne vyvinuli spoločnosti Microsoft a IBM. Jazyk WSDL pomáha webovým službám komunikovať tak, že si navzájom posielajú údaje o svojej funkčnosti a svojich vlastnostiach. WSDL sa často používa v kombinácií s protokolom SOAP a XML schémou na poskytovanie webových služieb cez internet. Klientský program, ktorý sa pripája k webovej službe si môže prečítať WSDL a zistiť, aké funkcie sú na serveri k dispozícii. Všetky použité špeciálne dátové typy sú vložené do súboru WSDL vo forme schémy XML. Klient potom môže použiť SOAP na skutočné volanie jednej z funkcií uvedených v dokumente WSDL. Jazyk je používaný aj UDDI registrami, ktoré si predstavíme v ďalšej kapitole. [38]

WSDL dokument rozdeľujeme na dve časti, a to abstraktný opis služby a konkrétny opis služby :

**Abstraktný opis služby**- V tejto časti sú definované vstupné správy, ktoré služba prijíma od iných služieb a výstupne správy, ktoré služba môže posielať iným službám. Všetky správy sú združené do operácii a operácie sú združené do rozhraní. Abstraktný opis neuvádza žiadne technické detaily o spôsobe prenosu údajov. Tie sú uvedené v konkrétnom opise služby. [3]

* **Správa**- je to abstraktná definícia údajov vo forme správy, ktoré do služby vstupujú alebo vystupujú. Správa môže byť vo forme celého dokumentu alebo vo forme argumentov, ktoré sa majú priradiť k volaniu metódy webovej služby. [38]
* **Operácia-** je to abstraktná definícia operácie pre správu. Operácia obsahuje element *input* a *output*, ktorý definuje mechanizmus výmeny správ. Tento mechanizmus hovorí o tom, či je správa z pohľadu služby vstupnou alebo výstupnou. Pomocou týchto atribútov vieme taktiež zistiť, či ide o jednosmernú alebo obojsmernú komunikáciu (prípade absencie jedného, ide len o jednosmernú). [3] [38]
* **Rozhranie**- je to abstraktná množina operácií mapovaných na jeden alebo viac koncových bodov. Rozhranie taktiež definuje kolekciu operácií, pre väzbu (binding). [38]

**Konkrétny opis služby**- obsahuje technické detaily na komunikáciu so službou. Základné elementy, ktoré obsahuje konkrétny opis služby sú väzba, port a služby.

* **Väzba**- podľa [3] „*väzba vyjadruje vzťah rozhrania služby ku konkrétnym prenosovým protokolom a tiež k protokolom vyjadrujúcim formát prenášaných správ.*“ Ďalej sa tu definuje aj štýl správy. Poznáme štyri štýly správ a to RPC/encoded, RPC literal, Document/encoded a Document/literal. Viac sa o nich dozvieme v kapitole o protokole SOAP. [3]
* **Port**- ide o adresu servera na ktorej sa dá kontaktovať webová služba.
* **Služba**- sú tu zoskupené všetky porty pomocou ktorých sa dá kontaktovať webová služba. [38]



Obrázok 11: Štruktúra WSDL dokumentu (verzia 1.1 a verzia 2.0) zdroj: [47]

## SOAP

SOAP (skratke pre- Simple Object Access Protocol) je špecifikácia protokolu na výmenu štruktúrovaných informácií, pri implementácii webových služieb v počítačových sieťach. Na formátovanie SOAP správ sa používa jazyk XML a pri prenose správ sa spolieha na protokoly aplikačnej vrstvy, najčastejšie na protokol HTTP (Hypertext Transfer Protocol), aj keď niektoré staršie systémy komunikujú prostredníctvom protokolu SMTP (Simple Mail Transfer Protocol). SOAP bol vyvinutý v roku 1998 spoločnosťami Mentor, User Land Company a Microsoft. V roku 1999 bola vydaná jeho prvá verzia. S verziou 1.2 sa SOAP začal používať vo veľkom. Služba SOAP je nezávislá aj od platformy a jazyka. SOAP správy sú zabalené do takzvanej obálky (envelope), ktorá je nosičom celej správy. Obálka sa ďalej delí na povinnú časť telo (body) a nepovinnú časť hlavička (header) [39]

**Obálka**- zapuzdruje všetky údaje v správe a identifikuje dokument XML ako správu SOAP.

**Hlavička**- podľa [3] „*je časť správy, ktorá je vymedzená pre zápis metadát. Metadáta predstavujú doplnkové údaje, ktoré sa viažu k údajom prenášaným v hlavnej časti správy. Niekedy sa preto označujú aj ako údaje o údajoch. Hlavička nie je povinnou súčasťou SOAP správy, ale zvyčajne sa nevynecháva, pretože sa do nej môžu zapisovať údaje dôležité z hľadiska doručenia správy a spracovávania jej obsahu cieľovými aplikáciami.*“

**Telo**- obsahuje podrobnosti skutočnej správy, ktorú je potrebné odoslať z webovej služby do volajúcej aplikácie. Tieto údaje zahŕňajú informácie o požiadavkách a odpovediach. [40]

**Chyba**- tento prvok poskytuje informácie o chybách, ktoré sa vyskytli pri odosielaní správy, ako napr. nesúlad verzie, chyba klienta, keď bola správa nesprávne vytvorená, chyba servera, keď došlo k problémom so serverom. Obsahuje názov chyby jej stav a aj informácie o chybe. [39]

Štýl SOAP správy je definovaný vo WSDL dokumente v časti binding a to pomocou atribútov *style,* ktorý môže mať hodnoty *RPC* a *document* a atribútu use, ktorý môže mať hodnoty encodet a literal. Existujú 4 kombinácie hodnôt týchto atribútov a to:

* RPC/encoded
* RPC/literal
* Document/encoded
* Document/literal [3]

**Štýl RPC**- používame ho vtedy, ak poznáme presný názov funkcie a vstupné dáta funkcie, ktorej funkcionalitu chceme zavolať na vzdialenom serveri.

**Štýl dokument**- pri tomto štýle nemusí byť špecifikovaný názov funkcie a webová služba by mala na základe prijatých dát rozpoznať, ktorú zo svojich funkcií má použiť. [3]

Správy štýlu RPC a dokument môžu byť ďalej dvoch typov a to:

* **Encoded**- SOAP správa musí byť štruktúrovaná podľa konkrétnej XML schémy.
* **Literal**- SOAP správa nemusí byť štruktúrovaná podľa konkrétnej XML schémy. [3] [41]

V nasledujúcich úryvkoch kódu môžeme vidieť príklad dvoch SOAP správ. Prvý predstavuje SOAP požiadavku, pomocou ktorej je volaná metóda na serveri *Získajcenu*(). Údaje, ktoré sú do tejto metódy odoslané sú v elemente *Polozka*. SOAP odpoveď nám vráti cenu v elemente *cena*. [42]

Obrázok 12: Príklad SOAP požiadavky zdroj: vlastné spracovanie

<?<?xml version="1.0"?>

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope/" soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

 <soap:Body>

  <!--Názov metódy, ktorá sa má spustiť na servery-->

  <o:ZiskajCenu xmlns:o="https://nasObchod/ceny">

   <!--Názov položky je vstupná hodnota-->

   <o:Polozka>pohár-sklenený </o:Polozka>

  </o:ZiskajCenu>

 </soap:Body>

</soap:Envelope>

Obrázok 13: Príklad SOAP odpovede zdroj: vlastné spracovanie

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope/" soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

 <soap:Body>

  <o:ZiskajCenuOdpoved xmlns:o="https://nasObchod/ceny">

   <o:Cena>5.2</o:Cena>

  </o:ZiskajCenuOdpoved>

 </soap:Body>

</soap:Envelope>

## UDDI

UDDI (Universal Description, Discovery and Integration) je štandard založený na jazyku XML na opis, publikovanie a vyhľadávanie informácií o webových službách. Myšlienka UDDI registrov spočíva vo vytvorení centrálneho adresára služieb, v ktorom si autorizovaný používatelia alebo iné služby vedia vyhľadať služby podľa ich požiadaviek. UDDI bol vytvorený spoločnosťami Microsoft, IBM a Ariba v septembri 2000. Dnes sú tieto registre spravované firmou OASIS. UDDI registre delíme na typy, a to na verejné a súkromné registre.

Myšlienka verejných registrov spočíva v tom, že k službám v nej zverejnených by mal mať prístup každý, kto o nich prejavil záujem. Hlavnou snahou pri týchto registroch bolo podporiť elektronický obchod medzi podnikmi. V roku 2006 najväčší prevádzkovatelia týchto registrov zrušili ich prevádzku pre nedostatočný záujem. Verejné registre sa delili na biele, žlté a zelené stránky. Biele stránky obsahovali kontaktné údaje na poskytovateľa služby. Žlté stránky obsahovali klasifikačné údaje o činnosti firmy, ktorá poskytovala webové služby. Zelené stránky obsahovali informácie o dostupných službách od daného poskytovateľa. Taktiež sa tu nachádzali WSDL dokumenty daných služieb, ktoré hovorili o tom, ako sa dá s týmito službami po technickej stránke spojiť. [3]

Na rozdiel od verejných sú súkromné registre používané na zverejňovanie webových služieb, iba v rámci jedného podnikateľského subjektu. Predpokladom je, že firma má svoj informačný systém vybudovaný na princípoch SOA (Servisne orientovaná architektúra), čo znamená, že sa daný systém skladá z viacerých menších webových služieb, ktoré v prípade potreby spolupracujú a vedia si vymieňať údaje. Súkromné registre sa na rozdiel od verejných používajú dodnes. [3]

## SOAP vs REST

V súčasnosti existuje niekoľko modelov dizajnu webových služieb, ale dva najdominantnejšie sú SOAP a REST. REST predstavuje mladšiu alternatívu k SOAP webovým službám. Na rozdiel od SOAP je REST architektonický štýl pre vývoj softvéru. Keďže SOAP je protokol a REST je architektúra, ich porovnávanie sa môže zdať komplikované a mätúce. Dôležité je uvedomiť si, že napriek rozdielom sú oba populárne spôsoby prenosu údajov prostredníctvom rozhraní API. Architektúra štýlu REST je klient server, v ktorej klient posiela požiadavku na server a potom server spracúva požiadavku a vráti odpovede. Tieto žiadosti a odpovede sú založené na prenose reprezentácií zdrojov. Zdroj je niečo, čo je identifikované pomocou URI. REST webové služby využívajú HTTP metódy GET, PUT, POST a DELETE HTTP metódy na načítanie, vytváranie, aktualizovanie a mazanie zdrojov. V nasledujúcej tabuľke si oba prístupy porovnáme. [43]

Tabuľka 4: Porovnanie SOAP protokolu a REST architektúry zdroj: [44] [45]

|  |  |  |
| --- | --- | --- |
|  | SOAP | REST |
| **Dizajn** | Štandardizovaný protokol s prísnymi pravidlami, ktoré treba dodržiavať. | Architektúra s voľnými odporúčaniami a usmerneniami. |
| **Prístup** | Funkčne riadený, SOAP používa rozhrania služieb na odhalenie obchodnej logiky. | Dátami riadený. REST používa URI na odhalenie obchodnej logiky. |
| **Kešovanie** | Volania API sa neukladajú do vyrovnávacej pamäte. | Volania API sa ukladajú do vyrovnávacej pamäte. |
| **Bezpečnosť** | SOAP definuje štandardy, ktoré treba prísne dodržiavať. | Nedostatok podpory štandardov pre bezpečnostnú politiku. |
| **Výkon** | Spotrebuje väčšiu šírku pásma, pretože odpoveď SOAP môže vyžadovať viac ako 10-krát toľko bajtov v porovnaní s REST. | Spotrebuje menšiu šírku pásma, pretože jeho odozva je ľahká. |
| **Formát správ** | Len XML. | XML, JSON,YAML,HTML a ďalšie. |
| **Prenosový protokol** | SMTP, HTTP UDP a ďalšie. | Iba HTTP. |
| **Odporúčaný v** | Finančné služby, podnikové aplikácie, aplikácie ktoré vyžadujú vysokú mieru ochrany, telekomunikačné služby . | Verejné API, pre web služby aplikácii ako sú sociálne siete. Pre CRUD webové služby. |
| **Výhody** | Štandardizácia, bezpečnosť, rozšíriteľnosť. | Vysoký výkon, škálovateľnosť, flexibilita. |
| **Nevýhody** | Ich komplexnosť a zložitosť nemusí byť vždy vhodná. Slabší výkon, menšia flexibilita. | Slabá štandardizácia. Žiadne vstavané bezpečnostné mechanizmy. |

# Ciele práce

Rýchly technologický pokrok výrazne zmenil spôsob, akým dnes podnikáme. Internet spôsobil revolúciu v spôsobe nakupovania, komunikácie a interakcie s ostatnými. Elektronický obchod sa stal prosperujúcim odvetvím, pričom nakupovanie na internete je čoraz populárnejšie a nadobúda čoraz väčší význam. Online kníhkupectvá nezostali pozadu a dopyt po nich výrazne vzrástol. V reakcii na tento dopyt internetové kníhkupectvá neustále zlepšujú svoje služby, aby zabezpečili spokojnosť zákazníkov a zaistili si konkurenčnú výhodu.

Cieľom tejto práce je prispieť k rozvoju online kníhkupectiev, skúmaním možností využívania XML webových služieb v ich informačnom systéme. Hlavným cieľom tejto práce je navrhnúť a vyvinúť webovú ASP .NET XML službu, ktorú možno integrovať do online kníhkupectva. Táto webová služba by mala poskytovať rôzne usporiadané informácie o knihách a transakciách v danom online kníhkupectve, ako aj zabezpečiť základnú manipuláciu s týmito informáciami. Cieľ práce súvisí aj s tvorbou ukážkového dynamického webového klienta, ktorý môže dané vystavené metódy webovej služby použiť a výsledky vo vhodnej forme zobraziť používateľovi. Služba má rozšíriť funkčnosť internetového kníhkupectva a svojou funkcionalitou poskytnúť svojím používateľom lepší zážitok.

Pre účely splnenia hlavného cieľa práce sme si zvolili aj čiastkové ciele, ktoré toto splnenie podmieňujú. Medzi tieto čiastkové ciele patria :

1. Štúdium materiálov ohľadom webových služieb, ako aj oboznámenie sa s technológiami pomocou, ktorých sa dajú webové služby navrhnúť vytvoriť, testovať, nasadiť a v konečnom dôsledku aj použiť v prostredí webového klienta.
2. Analýza a výber vhodných technológií, nástrojov a knižníc na tvorbu webových služieb a klienta.
3. Pomocou prieskumu a pomocou konzultácii zistiť, akým spôsobom sledujú parametre kníh vybrané kníhkupectvá, aké technológie pri tom využívajú, a v ktorých ďalších oblastiach ich majú nasadené.
4. Formulácia jasných a merateľných požiadaviek na nami vytvorenú webovú službu tak, aby bolo od začiatku jasné akú funkcionalitu od týchto služieb očakávame.
5. Vytvorenie vhodnej dátovej štruktúry v značkovacom jazyku XML a naplnenie tejto štruktúry dostatočným množstvom testovacích údajov.
6. Pri kódovaní služieb a klienta sa snažiť písať čo najčistejší, dobre čitateľný a okomentovaný kód. Taktiež si dobre premyslieť, aké rôzne chybové stavy by mohli pri každej metóde nastať a snažiť sa tieto chyby zachytiť. Metódy taktiež vhodným spôsobom otestovať.
7. Po otestovaní nasadiť na IIS server.

# Metodika práce a metódy skúmania

V tejto časti si predstavíme, akým spôsobom a za pomoci akých metód sme túto diplomovú prácu postupne vypracovávali.

**Literárna rešerš**- ešte pred tým, ako sme začali písať samotnú prácu sme určitý čas strávili výberom a študovaním vhodnej literatúry ohľadom webových služieb a všetkých potrebných technológií, ktoré sú potrebné na ich tvorbu, ako aj tvorbu webového klienta. V tejto fáze tvorby diplomovej práce sme si rozšírili poznatky o danej téme a získali pomerne ucelený pohľad na túto problematiku. Tieto poznatky sme potom ďalej využili pri písaní prvej teoretickej časti, ale hlavne aj pri tvorbe webových služieb v praktickej časti tejto práce. Pri citovaní a parafrázovaní sme využívali knižné zdroje, ale aj početné množstvo internetových zdrojov. Dôvod, pre tento väčší počet internetových zdrojov je ten, že k niektorým technológiám prinášali novšie poznatky, ako knižné zdroje, ktoré boli napísané o niečo skôr a existovalo riziko, že by informácie v nich nemuseli byť už relevantné. Pri výbere internetových zdrojov sme si taktiež zvolili kombináciu menej prehľadných a rozsiahlych dokumentácií a  ľahšie stráviteľných a prehľadnejších článkov a tutoriálov, v ktorých bola daná problematika výstižné spracovaná. Využívali sme aj  dostupné zdroje známych veľkých firiem, ako je napríklad IBM, ktoré stáli pri zrode webových služieb a majú k tejto téme vypracované bohaté množstvo rôzneho materiálu.

**Metóda komparácie**- túto metódu sme využili v praktickej aj v teoretickej časti. V teoretickej, napríklad pri porovnávaní jazykov na prenos údajov JSON a XML, ale aj pri porovnaní modelov dizajnu webových služieb podľa REST a SOAP architektúry. V praktickej časti pri porovnaní nami vytvoreného riešenia a riešení, ktoré sú nasadené v reálnych kníhkupectvách.

**Metóda analýzy**- túto metódu sme využili pri analýze situácie v reálnych kníhkupectvách, ale aj pri rozhodovaní o tom, ktoré technológie je najvhodnejšie použiť pri tvorbe webových služieb, webového klienta a XML dátovej základne tak, aby bol vývoj čo najefektívnejší. Analýzou sme taktiež vybrali vhodné parametre, ktoré môžeme sledovať pri knihách a transakciách internetového kníhkupectva, ako aj najlepší spôsob ich zobrazenia pre konečného používateľa.

**Metóda prieskumu**- túto metódu sme použili, keď sme sa snažili získať informácie o dostupných riešeniach v iných internetových kníhkupectvách. Metódu sme robili emailovým a osobným oslovením kompetentných osôb z vybraných kníhkupectiev, ako aj prieskumom voľne dostupných materiálov.

**Metóda syntézy**- túto metódu sme využili, keď sme nami vytvorený súbor webových služieb a funkcií na strane klienta prepojili do jedného celistvého fungujúceho celku.

# Súčasný stav vo vybraných firmách

Za účelom zistenia aktuálnej situácie na trhu sme oslovili viacero kníhkupectiev. Zisťovali sme, že akým spôsobom a za pomoci akých technológií sledujú napr. rôzne parametre kníh, aké spôsoby používajú na ich evidenciu, vyhľadávanie, a v ktorých iných oblastiach im dané technológie uľahčujú prevádzku online kníhkupectva. Zistené informácie boli inšpiráciou pre nami vytvorené webové služby v praktickej časti. Tieto informácie, ale môžu čitateľovi taktiež priniesť zaujímavý prehľad o tom, ako je daná problematika riešená v prostredí reálnych podnikateľských subjektov. Keďže v malých kníhkupectvách si väčšinou vystačili s nástrojom Excel, tieto kníhkupectvá nebudeme opisovať. V tejto časti si predstavíme prípadovú štúdiu vo väčšej firme, o ktorej sa nám podarilo získať najväčšej množstvo informácií. Názov firmy samozrejme z konkurenčných dôvodov anonymizujeme a budeme ju nazývať ako kníhkupectvo Omega.

**Kníhkupectvo Omega** je popredným online predajcom kníh, časopisov a elektronických kníh v slovenskej republike. Spoločnosť ponúka široký výber titulov v rôznych kategóriách, od beletrie a literatúry faktu, až po akademické a odborné knihy. Aby spoločnosť splnila požiadavky rastúcej zákazníckej základne, investovala do moderných technológií, ktoré podporujú jej prevádzku, od riadenia zásob až po realizáciu objednávok. Jednou z kľúčových technológií, ktoré kníhkupectvo používa, je PHP programovací jazyk, ktorý umožňuje vývoj dynamických webových aplikácií. V spolupráci s jazykom Javascript (bez použitia frameworkov) na frontende je v PHP napísaný celý internetový obchod, ktorý umožňuje zákazníkom vyhľadávať a prezerať ukážky rôznych knižných titulov, ako aj spravovať svoje účty a  vytvárať nové objednávky a rezervácie. V tomto jazyku sú napísané aj ostatné podsystémy daného kníhkupectva, ako je napr. skladový systém a fakturačný systém. Vo firme je zamestnaných celkovo 10 developerov, z čoho 9 je takzvaných „fullstack developerov“ a jeden je čisto zameraný na frontend. V tíme taktiež pracuje UX dizajnér a 2 produktové manažérky, ktoré napríklad pripravujú jasne prehľadné zadania pre programátorov. Webová lokalita je postavená na frameworku CakePHP, ktorý poskytuje pevný základ na vývoj komplexných aplikácií, ktoré zvládajú veľké objemy návštevnosti. Na správu svojej rozsiahlej dátovej základne kníhkupectvo používa MySQL relačný databázový systém. MySQL sa používa na uloženie a organizáciu informácií o parametroch knižných produktov, akými sú tituly, autori, obsahy, ceny, dostupnosť a mnoho ďalších. Používa sa aj na perzistenciu informácií o objednávkach a na vytváranie reportov, ktoré pomáhajú spoločnosti monitorovať jej výkonnosť. Informačný systém spoločnosti Omega čelil v minulosti hneď niekoľkým výzvam. Medzi tieto výzvy patrilo hlavne:

**Pomalé vyhľadávanie**- kníhkupectvo malo rozsiahlu databázu kníh a existujúci vyhľadávač nebol dostatočne rýchly na spracovanie požiadaviek na vyhľadávanie, čo spôsobovalo oneskorenie výsledkov vyhľadávania.

**Nepresné výsledky vyhľadávania**- existujúci vyhľadávač poskytoval nepresné výsledky vyhľadávania, čo viedlo k zlému zážitku zákazníkov s webovou aplikáciou kníhkupectva.

**Slabé možnosti filtrovania**- kníhkupectvo malo obmedzené možnosti filtrovania, ktoré zákazníkom sťažovali nájdenie správnej knihy.

Na zlepšenie funkcií vyhľadávania na svojej webovej lokalite spoločnosť implementovala distribuovaný vyhľadávací a analytický engine Elasticsearch. Elasticsearch umožňuje zákazníkom vyhľadávať knihy vo viacerých kategóriách a filtrovať výsledky podľa autora, vydavateľa, cenového rozpätia a mnohých ďalších kritérií. Elasticsearch poskytuje presnejšie výsledky vyhľadávania, a to za pomoci vyhľadávania v prirodzenom jazyku, ktoré rozumie zámeru používateľa a priraďuje najrelevantnejšie výsledky. Poskytuje tiež návrhy a opravy nesprávne napísaných slov v reálnom čase, čo zákazníkom uľahčuje nájsť to, čo hľadajú. Elasticsearch taktiež poskytol rýchlejšie vyhľadávanie s výsledkami vrátenými do niekoľkých sekúnd.

Okrem toho kníhkupectvo Omega využíva webové služby na automatizáciu rôznych úloh a integráciu s externými systémami. Spoločnosť napríklad vytvorila vlastnú aplikáciu, ktorá umožňuje zákazníkom rezerváciu knihy, ktorú si môžu vyzdvihnúť v nimi určenom kamennom obchode. Táto aplikácia využíva webové služby na prístup k informáciám o zásobách kníh na sklade a na každej jednotlivej predajni v reálnom čase. To umožňuje rýchly presun knihy na zákazníkom požadované miesto. Kníhkupectvo používa webové služby aj na automatizáciu procesu importu informácií o produktoch od svojich dodávateľov. Spoločnosť dostáva prostredníctvom API rozhraní podrobné informácie o nových a existujúcich tituloch od desiatok vydavateľov a distribútorov. Kníhkupectvo používa vlastný skript na analýzu a spracovanie údajov o knihách, ktoré sa potom načítajú do MySQL databázy .

Kníhkupectvo Omega každý deň prijíma stovky až tisíce objednávok od zákazníkov, ktorí navštívia ich predajne, objednajú si tovar online a požiadajú o doručenie domov alebo vyzdvihnutie na predajni. Keďže dnešný zákazník je náročný a svoju objednávku by chcel mať doma už najradšej na druhý deň potrebovala firma vymyslieť nejaké riešenie, ako optimalizovať celý proces naskladňovania, vyskladňovania, expedície a samotného doručenia knihy.

Na optimalizáciu trás a predpoveď dopytu spoločnosť využíva externého partnera Algopine. Riešenie od tejto firmy používa algoritmy strojového učenia, ako aj grafové algoritmy na optimalizáciu trás, predpoveď dopytu, optimalizáciu cenovej politiky, optimalizáciu odporúčania produktu alebo optimalizáciu trasy produktu v sklade.

Spoločnosti Algopine sú poskytnuté nasledujúce údaje:

* počet objednávok prijatých na každom mieste každý deň,
* umiestnenie každej predajne a adresy doručenia ich zákazníkov,
* kapacitu každého vozidla používaného na doručovanie,
* časové okná doručenia určené zákazníkmi,
* vzdialenosť medzi jednotlivými predajňami a adresami doručenia.

Spoločnosť Algopine spracúva tieto údaje pomocou svojho algoritmu optimalizácie trasy. Tie určia, aké sú najefektívnejšie trasy na doručovanie objednávok. Spracované údaje sú posielané späť do systému spoločnosti Omega.

Algopine tiež pomáha kníhkupectvu zlepšiť plánovanie dopytu tým, že analyzuje údaje o objednávkach a poskytuje prehľad o štruktúre dopytu. To pomáha kníhkupectvu presnejšie predpovedať svoj dopyt a optimalizovať úroveň zásob.

Využívanie vyššie spomínaných technológií v spoločnosti jej celkovo umožnilo zefektívniť svoju činnosť, zlepšiť služby zákazníkom a zvýšiť ziskovosť. Využitím pokročilých programovacích jazykov, grafových algoritmov a algoritmov strojového učenia, ako aj systémov riadenia databáz, vyhľadávačov a webových služieb, spoločnosť vytvorila vysoko efektívnu a škálovateľnú platformu, ktorá dokáže zvládnuť požiadavky jej rastúceho podnikania.

# Výsledky práce

V tejto časti práce si predstavíme postup, akým sme vytvárali webové služby na sledovanie vybraných parametrov  kníh a transakcií elektronického kníhkupectva. V samostatných podkapitolách si postupne predstavíme všetky technológie, frameworky triedy a knižnice, ako aj detailne popísané zdrojové kódy jednotlivých metód vytvorených na strane ASP .NET webových služieb, ako aj webového klienta.

## Tvorba dátovej XML základne

Základom každej dobre fungujúcej aplikácie alebo webovej služby je vhodne štruktúrovaná databáza. V tejto práci sú všetky údaje uložené v XML dokumentoch. Všetky údaje, z ktorých naša webové služba vychádza sú rozdelené do dvoch XML súborov, a to books.xml, ktorý obsahuje základné informácie o knihách a transaction.xml, ktorý obsahuje základné informácie o obchodných transakciách s týmito knihami.

### Opis dokumentu books.xml

V nasledujúcej ukážke môžete vidieť stromovú štruktúru dokumentu *books.xml* spolu s údajmi o jednej knihe.

Obrázok 14: Ukážka stromovej štruktúry XML dokumentu books.xml zdroj: vlastné spracovanie

<?xml version="1.0" encoding="UTF-8"?>

<Bookstore xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

 <books>

  <book>

   <id>10</id>

   <nazov>Hacking bez tajemství - Webové aplikace</nazov>

   <autori>

    <autor1>Mike Shema</autor1>

    <autor2>Joel Scambray</autor2>

   </autori>

   <kategoria>Odborné a náučné</kategoria>

   <isbn>8072267698</isbn>

   <jazyk>cz</jazyk>

   <pocet\_stran>360</pocet\_stran>

   <vazba>brožovaná väzba</vazba>

   <rok\_vydania>2003</rok\_vydania>

   <vydavatelstvo>Computer Press</vydavatelstvo>

   <predajna\_cena>18.5</predajna\_cena>

   <nakupna\_cena>11.89</nakupna\_cena>

   <marza>0.555929352396972</marza>

   <zisk\_kus>6.61</zisk\_kus>

   <obsah>Správcovia počítačových systémov zistili, že zákerní a ambiciózni jedinci môžu ľahko preniknúť do počítačových sietí a systémov. Kniha s názvom Hacking bez tajomstiev poukázala na túto skutočnosť a ukázala správcom, že pre zabezpečenie systémov je nutné dodržiavať pravidlá ako sú firewall, bezpečné konfigurácie operačných systémov, používať aktualizácie od dodávateľov a iné pravidlá, ktoré boli v minulosti opomínané. Hoci stále existuje mnoho správcov, ktorí nemajú s touto krutou skúsenosťou skúsenosti, väčšina už pochopila dôležitosť týchto zásad zabezpečenia informačných systémov.</obsah>

   <priemerne\_hodnotenie>5</priemerne\_hodnotenie>

   <obrazok>../img/10.jpg</obrazok>

  </book>

 </books>

</Bookstore>

Koreňovým elementom dokumentu books.xml je *<Bookstore>*, ktorý má atribút xmlns:xsi na určenie menného priestoru XML. Element Bookstore má jeden detský element s názvom *<books>*, ktorý obsahuje zoznam elementov *<book>.* Každý element *<book>* predstavuje informácie o jednej knihe a obsahuje nasledujúce detské elementy:

Tabuľka 5: Popis elementov v XML dokumente books.xml zdroj: vlastné spracovanie

|  |  |  |
| --- | --- | --- |
| Meno elementu | Popis | Dátový typ |
| **<id>**: | Jedinečný  identifikátor knihy. | Celé číslo. |
| **<nazov>**: | Názov knihy. | Znakový reťazec. |
| **<autori>:** | Rodičovský element pre autorov knihy, ktorý obsahuje dva detské elementy *<autor1>* a *<autor2>,* z ktorých každý predstavuje meno autora. V prípade že kniha nemá druhého autora tak element <autor2> má hodnotu nastavenú na „-“. | Znakový reťazec. |
| **<kategoria>**: | Kategória  Knihy. Hodnota tohto elementu môže obsahovať napr. žáner knihy, ale aj iné marketingové označenia pre účely lepšej vyhľadateľnosti knihy. | Znakový reťazec. |
| **<isbn>**: | Medzinárodné štandardné číslo knihy. | Znakový reťazec. |
| **<jazyk>**: | Jazyk, v ktorom je daná kniha vydaná. Je reprezentovaný ako znakový reťazec. Jazyk je vždy len jeden. V prípade, že má kníhkupectvo viac kníh s rovnakým titulom, ale iným jazykom, tak sú tieto knihy uložené samostatne s jedinečným identifikátorom. | Znakový reťazec. |
| **<pocet\_stran>**: | Počet strán v knihe. | Celé číslo. |
| **<väzba>:** | Väzba knihy- napr. pevná brožovaná a pod. | Znakový reťazec. |
| **<rok\_vydania>**: | Rok vydania knihy. | Znakový reťazec. |
| **<vydavatelstvo>**: | Názov vydavateľa knihy. | Znakový reťazec. |
| **<predajna\_cena>**: | Predajná cena knihy. Ide o cenu, za ktorú je kniha predávaná zákazníkom. | Desiatkové číslo. |
| **<nakupna\_cena>**: | Nákupná cena knihy. Ide o cenu za akú je kniha kupovaná od dodávateľa. | Desiatkové číslo. |
| **<marza>:** | Zisková marža knihy. Vzorec pre výpočet marže je: (predajná cena – nákupná cena) / predajná cena x 100. | Desiatkové číslo. |
| **<zisk\_kus>**: | Zisk na jeden výtlačok knihy. Vzorec (predajná cena- nákupná cena). | Desiatkové číslo. |
| **<obsah>**: | Obsah knihy opisujúci základnú tému a účel knihy. | Znakový reťazec. |
| **<priemerne\_hodnotenie>**: | Priemerné hodnotenie knihy. Škála hodnotenie je od 1-5. | Znakový reťazec. |
| **<obrazok>**: | Obsahuje relatívnu cestu k súboru, ktorá obsahuje obrázok obalu knihy. | Znakový reťazec. |

### Opis dokumentu books\_transactions.xml

V nasledujúcej ukážke je stromová štruktúra dokumentu book\_transakcie.xml spolu s údajmi  o dvoch typoch transakcií. Jedna je typu nákup a druhá je typu predaj.

Obrázok 15: Príklad stromovej štruktúry XML dokumentu book\_transakcie.xml zdroj: vlastné spracovanie

<?xml version="1.0" encoding="utf-8"?>

<knihy\_transakcie xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<transakcia>

 <id\_transakcie>815</id\_transakcie>

 <id\_knihy>9</id\_knihy>

 <datum>2020-03-27</datum>

 <typ\_transakcie>nákup</typ\_transakcie>

 <mnozstvo>5</mnozstvo>

 <cena\_za\_jednotku>25.99</cena\_za\_jednotku>

 <celkovo\_cena>129.95</celkovo\_cena>

 <aktualne\_mnozstvo\_na\_sklade>7</aktualne\_mnozstvo\_na\_sklade>

</transakcia>

<transakcia>

 <id\_transakcie>816</id\_transakcie>

 <id\_knihy>1</id\_knihy>

 <datum>2020-03-28</datum>

 <typ\_transakcie>predaj</typ\_transakcie>

 <mnozstvo>-4</mnozstvo>

 <cena\_za\_jednotku>11.9</cena\_za\_jednotku>

 <celkovo\_cena>47.6</celkovo\_cena>

 <aktualne\_mnozstvo\_na\_sklade>121</aktualne\_mnozstvo\_na\_sklade>

</transakcia>

</knihy\_transakcie>

Koreňovým elementom dokumentu XML je element *knihy\_transakcie*, ktorý má atribút *xmlns:xsi* na určenie menného priestoru XML. Každá transakcia je reprezentovaná elementom *<transakcia>*. Element *<transakcia>* obsahuje podradené elementy predstavujúce špecifické informácie o transakcii:

Tabuľka 6: Popis elementov v XML dokumente book\_transaction.xml zdroj: vlastné spracovanie

|  |  |  |
| --- | --- | --- |
| Názov elementu | popis | Dátový typ |
| **<id\_transakcie>** | Jedinečný identifikátor transakcie. | Celé číslo. |
| **<id\_knihy>** | Jedinečný identifikátor knihy zapojenej do transakcie. | Celé číslo. |
| **<datum>** | Dátum kedy bola transakcia realizovaná. Formát dátumu je „yyyy-mm-dd“. | Dátum. |
| **<typ\_transakcie>** | Typ transakcie. Môže nadobúdať hodnoty „nákup“ alebo „predaj“. | Znakový reťazec. |
| **<mnozstvo>** | Množstvo kníh zahrnutých do transakcie (kladné pre „nákup“ a „záporné“ pre predaj). | Celé číslo. |
| **<cena\_za\_jednotku>** | Cena za jednotku knihy v eurách. | Desiatkové číslo. |
| **<celkovo\_cena>** | Celková cena transakcie v eurách. | Desiatkové číslo. |
| **aktualne\_mnozstvo\_na\_sklade>** | Aktuálny stav zásob danej knihy na sklade po realizácii transakcie. | Celé číslo. |

### Naplnenie XML súborov testovacími dátami

Po tom, ako sme vytvorili štruktúru XML dokumentov, tak prišiel čas ich naplniť dostatočným množstvom testovacích dát. Testovacie dáta sme vytvárali pomocou excelovského nástroja vývojár.

Nasledujúce kroky popisujú postup, ktorý sme použili na vytvorenie XML dokumentu z našej tabuľky v Exceli:

**Krok 1: Vytvorenie tabuľky**

V prvom kroku sme si vytvorili tabuľky s rovnakými názvami stĺpcov, aké majú naše dva XML dokumenty a naplnili sme ich testovacími dátami. Uistili sme sa, že tabuľka zahŕňa všetky potrebné polia a hlavičky a taktiež sme skontrolovali, či dáta boli dobre štruktúrované a naformátované.

**Krok 2: Aktivácia karty Developer**

Aby sme sa dostali k nástroju vývojár, museli sme aktivovať kartu vývojár v Exceli. Táto karta obsahuje všetky potrebné nástroje na vytváranie a úpravu XML súborov. Na to, aby sme túto kartu pridali do hlavného menu, tak sme klikli na položku *Súbor*, kde sme zvolili možnosti, potom možnosť prispôsobiť pás s nástrojmi a nakoniec sme zaškrtli políčko vedľa možnosti *Vývojár.*

**Krok 3: Vytvorenie XML schémy**

V nasledujúcom kroku sme si v samostatnom XML súbore vytvorili XML schému, ktorá ma rovnakú štruktúru ako očakávaný výstupný XML dokument. Excelu ako schéma stačia dve vzorové XML stromy s údajmi. Táto schéma je v podstate náčrtom pre XML súbor, ktorý chceme vytvoriť. Na vloženie schémy do Excelu sme prešli na kartu *Vývojár* a zvolili sme *zdroje,* a potom ďalej možnosť XML mapy, kde sme v ďalšom okne zvolili pridať mapu XML, a tým sme pridali nami vytvorenú schému XML.

**Krok 4: Mapovanie tabuľky na schému**

Po tom, ako sme pridali XML schému do Excelu, tak sme museli prepojiť dáta z našej tabuľky so schémou. To zahŕňalo prepojenie názvov stĺpcov v našej tabuľke Excelu s elementami v XML schéme. Na to, aby sme to dosiahli, tak sme na karte *Vývojár* zvolili *zdroje*, kde už môžeme vidieť nami vytvorenú schému z jej jednotlivými elementami. V ďalšom kroku sme jednoduchým „drag and droppom“ namapovali elementy z XML schémy s našou tabuľkou v Exceli.

Obrázok 16: Ukážka mapovania XML schémy na tabuľku v Exceli zdroj: vlastné spracovanie

**Krok 5: Exportovanie dát do XML**

Nakoniec sme údaje z tabuľky v Exceli exportovali do XML súboru. Na to, aby sme to dosiahli, tak sme v karte *vývojár* zvolili možnosť Exportovať a potom sme si vybrali možnosť exportovať dáta do XML.

Celkovo sa takto v našej XML dátovej základni nachádzajú údaje o 30 knihách a 6873 transakciách medzi rokmi 2020 a 2022.

## Definovanie požiadaviek

Ešte pred tým, než sme sa pustili do samotného programovania, tak sme si jasne zadefinovali požiadavky na funkcionalitu, ktorú by naša webová služba ako aj klient mali poskytovať.

### Požiadavky na metódy webovej služby

Vystavené metódy webovej služby, by mali byť schopné zvládať nasledujúce úlohy:

* Vystavené metódy webovej služby majú byť schopné načítať údaje z XML súboru.
* Pri metódach s komplexnejším výstupom tento výstup zapísať do jedného samostatného novo-vytvoreného XML súboru. Názov XML súboru bude pozostávať z časovej pečiatky, názvu volanej metódy a z aktuálnych hodnôt parametrov volanej metódy.
* Pri metódach s menším a opakujúcim sa výstupom zapísať tento výstup s časovou pečiatkou do jedného toho istého XML súboru s tým, že výsledok aktuálneho volania služby sa zapíše na koniec za posledné dieťa koreňového elementu XML dokumentu.
* Vystavené metódy webovej služby majú byť schopné ten istý výstup, ktorý bol zapísaný do XML súboru serializovať do JSON formátu a v tejto štruktúre ho poslať klientovi na ďalšie spracovanie.
* V prípade chyby služba odošle klientovi status-kód chyby, ako aj popis chyby.
* Vystavené metódy webovej služby by mali byť schopné vrátiť údaj o jednej knihe alebo transakcii na základe identifikátora, zoznam kníh a transakcií.
* Vystavené metódy webovej služby majú byť schopné vrátiť zoznam všetkých kníh a transakcií.
* Vystavené metódy webovej služby majú byť schopné pridať, aktualizovať alebo vymazať údaj o jednej knihe alebo transakcii.
* Vystavené metódy majú byť schopné vrátiť začiatočný a konečný počet kníh na sklade kníhkupectva v zadanom období. Používateľ si môže zvoliť, či chce vrátiť údaje len pre knihy patriace do hodnoty zadaného atribútu (napríklad autor, kategória, vydavateľstvo) alebo údaje o všetkých knihách. Výsledky s počtami a názvami kníh služba poskytne zoradené vzostupne alebo zostupne podľa hodnôt rôznych atribútov knihy (počet strán, rok vydania, cena, priemerné hodnotenie, názov). Metódy majú byť taktiež schopné vrátiť celkový, maximálny, minimálny a priemerný počet kníh na začiatku a na konci sledovaného obdobia.
* Vystavené metódy majú byť schopné poskytnúť agregované a zoskupené údaje o počte kusov predaných kníh, ako aj peňažného príjmu z predaja kníh v  zadanom období. Počty a príjmy z predaja kníh môžu byť agregované podľa rôzneho atribútu (kategória, jazyk, autor, väzba, rok vydania), ale aj podľa samotných titulov kníh. Cieľom je klientovi poskytnúť výstup v takom formáte, aby bol schopný vytvoriť drill down graf (teda graf, v ktorom sú dáta agregované a zoskupené podľa rôznej úrovne detailu a granularity), kde na najvyššej hierarchii budú údaje agregované a zoskupené podľa zvoleného atribútu a na nižšej hierarchii budú údaje o počtoch a príjmoch agregované a zoskupené podľa jednotlivých kníh.

Tieto údaje budú zoradené vzostupne alebo zostupne, a to buď podľa hodnoty (číselná hodnota/počty, príjmy) alebo názvu (abecedné zoradenie podľa názvu atribútu/knihy).

Vystavené metódy, by mali taktiež vrátiť celkový počet kusov a celkový príjem z predaných kníh v danom období, priemerný denný príjem a priemerný  počet predaných kusov v zadanom období, maximálny a minimálny príjem a počet predaných kusov pre podkategóriu atribútu, podľa ktorého sa zoskupovalo (Napr. pri zvolenom atribúte *kategória* je podkategória *beletria*, *klasika.* a pri zvolenom atribúte *väzba* je to napr. *pevná väzba*, *brožovaná väzba* ), a taktiež maximálny a minimálny príjem a počet predaných kusov pre tituly jednotlivých kníh.

* Vystavené metódy, by mali byť schopné urobiť podobnú funkcionalitu spomínanú v predošlom bode aj pre počty naskladnených kníh od dodávateľa ako aj pre náklady ktoré z toho vyplývajú.
* Vystavené metódy, by mali taktiež na základe údajov o predajoch a nákupoch v danom období poskytnúť pre manažment zaujímavé metriky, ktoré im prezradia niečo viacej o výkonnosti podniku v danom období.

### Požiadavky na klienta webovej služby

* Na domovskej stránke klienta, by sa mal nachádzať zoznam, ktorý bude obsahovať linky na stránky, kde sa bude demonštrovať funkcionalita vystavených metód webovej služby.
* Klient má byť single page dynamická webová aplikácia, ktorá dokáže s používateľom interagovať dynamicky bez nutnosti znovu načítania stránky.
* O každom úspechu alebo chybe, by mal byť používateľ klienta informovaný vhodným farebne označeným oznámením alebo upozornením.
* Vstupy do formulárov by mali obsahovať funkcionalitu auto dopĺňania textu, ktorá bude fungovať na princípe pred-načítaných údajov z XML dátovej základne. To poslúži jednak na urýchlenie zadávania vstupov, ale aj zníži počet chybových vstupov, keďže používateľ tak bude vedieť v akom formáte má vstup zadať.
* Klient by mal byť vytvorený ako responzívny, tak aby bol vhodný na použitie na rôznych zariadeniach.
* Výstupy webových služieb by mal klient prekonvertovať z JSON formátu do používateľsky prívetivého výstupu, akým je tabuľka alebo graf.
* Použitím vhodného nástroja, by malo byť možné výstup z klienta stiahnuť napríklad ako obrázok alebo vo formáte pdf, xsl, csv.

## Tvorba webovej služby

V nasledujúcich kapitolách sa budeme venovať tvorbe webových služieb. V krátkosti si predstavíme technológie, ktoré sme pri nich použili. Popíšeme si riešenie vo Visual studiu. Následne si popíšeme vybrané webové a newebové služby.

### Použité technológie pri tvorbe webovej služby

Pri tvorbe webových služieb sme použili framework ASP .NET, ktorý poskytuje vývojárom širokú škálu nástrojov a knižníc, ktorá im umožňuje vytvárať dynamické webové aplikácie, webové stránky alebo  webové služby pomocou rôznych programovacích jazykov, ako sú C#, Visual Basic alebo F#. Zo širokej ponuky programovacích jazykov sme si zvolili objektovo orientovaný jazyk C#, s ktorým máme najväčšie skúsenosti.

Celá naša webová služba sa bude nachádzať v ASMX súbore. Skratka ASMX (Active Server Method Extension) je to typ súboru, ktorý sa používa na definovanie a implementáciu webových SOAP služieb v ASP .NET frameworku. Ako integrované vývojové prostredie pri tvorbe webovej služby budeme používať produkt od firmy Microsoft a to Visual studio. Webové služby budú nasadené na IIS server. Služby budeme testovať pomocou testovacieho klienta bežiaceho na ľahšej verzii IIS Express a ukážeme si aj testy v nástroji SOAP UI. Na dopytovanie z XML dokumentov sme využili jazyky XPath a LINQ. XPath sme využili hlavne pri jednoduchších dopytoch, kde nám stačilo prechádzať štruktúrou XML dokumentu a vybrať požadované údaje. Na druhej strane LINQ sme využili pri sofistikovanejších dopytoch, kde sme potrebovali technológie, ktoré nám umožnia funkcionalitu, ako je projekcia údajov do novej podoby, agregácia údajov, filtrovanie údajov, triedenie a pod.

### Popis riešenia vo Visual Studiu

Obrázok 17: Solution vo Visual Studiu zdroj: vlastné spracovanie

Celý kód ohľadom webovej služby klienta a XML súborov sa nachádza v riešení knihy\_jankech.sln.

Tabuľka 7: Popis jednotlivých komponentov riešenia vo Visual Studiu zdroj: vlastné spracovanie

|  |  |
| --- | --- |
| Názov komponenty | Popis |
| Solutions | Kontajner pre viacero projektov v rámci jedného riešenia. Zabezpečuje organizáciu a správu projektových súborov. |
| Project | Kontajner pre súbory zdrojového kódu, konfiguračné súbory a iné zdroje, ktoré tvoria aplikáciu. |
| Properties | Obsahuje súbory, ktoré definujú rôzne vlastnosti projektu, ako napríklad *AssemblyInfo.cs*, ktorý obsahuje metadáta o projekte. |
| References | Priečinok, ktorý obsahuje odkazy na iné zostavy a projekty, od ktorých projekt závisí. Môžu to byť knižnice .NET frameworky alebo balíky tretích strán. |
| Packages.config | Súbor, ktorý obsahuje zoznam všetkých balíkov NuGet, od ktorých projekt závisí. |
| Web.config | Konfiguračný súbor, ktorý obsahuje nastavenia pre webovú aplikáciu. |
| book\_services.asmx | Súbor, ktorý definuje našu webovú službu v aplikácii. |
| book\_services.asmx.cs | Súbor, ktorý obsahuje kód našej webovej služby definovanej v súbore book\_services.asmx. |
| dataClass.cs | Obsahuje 2 nami definované triedy. Prvá trieda je *"BookData"* a druhá trieda je *"TransactionData".* |
| priečinok bin | Tento priečinok obsahuje binárny výstup procesu zostavovania vrátane spustiteľných súborov (.exe) a knižníc dynamického prepojenia (.dll). |
| priečinok obj | Tento priečinok obsahuje priebežné súbory vytvorené počas procesu zostavovania, ako sú objektové súbory (.obj), dočasné súbory a iné artefakty. |
| priečinok packages | Priečinok packages (balíky): Tento priečinok obsahuje balíky NuGet, od ktorých aplikácia závisí, spolu s ich závislosťami. |
| priečinok xml | Obsahuje všetky vstupné a výstupne súbory v XML. |
| priečinok static | Obsahuje html css a javascriptové súbory klienta. |

### Diagram tried

Obrázok 18: Diagram tried vygenerovaný vo Visual Studiu zdroj: vlastné spracovanie

### Popis úvodných riadkov webovej služby

Obrázok 19: Časť zdrojového kódu definície triedy “book\_services“ zdroj: vlastné spracovanie

using System;

using System.IO;

using System.Linq;

using System.Web;

using System.Web.Services;

using System.Web.UI.WebControls;

using System.Xml;

using System.Xml.Linq;

using Newtonsoft.Json;

using Formatting = Newtonsoft.Json.Formatting;

namespace knihy\_jankech

{

[WebService(Namespace = "http://tempuri.org/")]

[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1\_1)]

[System.ComponentModel.ToolboxItem(false)]

// To allow this Web Service to be called from script, using ASP.NET AJAX, uncomment the

//following line.

[System.Web.Script.Services.ScriptService]

public class book\_services : System.Web.Services.WebService

{// tieto cesty treba nastaviť

private String fileBookInfo = private String fileBookInfo = HttpContext.Current.Server.MapPath("~/xml/books.xml");

private String fileBookTransactionInfo = HttpContext.Current.Server.MapPath("~/xml/books\_transactions.xml");

private String fileOutputSingleSearch = HttpContext.Current.Server.MapPath("~/xml/output.xml");

private String fileAmountFilterPath = HttpContext.Current.Server.MapPath("~/xml/outputfiles")...};

Toto sú prvé riadky našej webovej služby *book\_services.asmx*. Kód začína niekoľkými príkazmi *using*, ktoré importujú menné priestory potrebné pre funkčnosť webovej služby. Ďalej je trieda ozdobená niekoľkými atribútmi, ktoré poskytujú informácie o webovej službe vrátane jej menného priestoru a súladu so špecifikáciou *BasicProfile1\_1*. Atribút *ToolboxItem* je nastavený na hodnotu *false*, čo znamená, že webová služba sa nedá použiť zo súboru nástrojov vo Visual Studiu. Trieda obsahuje štyri reťazcové premenné, ktoré predstavujú cesty k dvom vstupným a jednému výstupnému XML súboru a jednému výstupnému priečinku, kde sa budú generovať výstupné XML súbory. Tieto cesty k XML súborom a priečinku sú získané pomocou funkcie *Server.MapPath* *triedy HttpContext.Current.* Táto funkcia mapuje virtuálnu cestu k súboru alebo priečinku na fyzickú cestu na serveri.Všetok kód pre webové a newebové služby sa nachádza v triede s názvom book\_services, ktorá dedí vlastnosti a metódy z triedy *"System.Web.Services.WebService".* Táto trieda obsahuje funkcie, ako je podpora metód HTTP GET a POST, automatické generovanie dokumentov WSDL alebo podpora formátov správ SOAP.

V dataClass.cs sa nachádzajú nami vytvorené dve triedy, ktoré slúžia na reprezentáciu a manipuláciu s údajmi súvisiacimi s knihami a transakciami. Použitím {get; set;} namiesto explicitného definovania metód get a set jazyk C# automaticky vygeneruje potrebný kód pre prístup k členským premenným objektu triedy.

Obrázok 20: Zdrojový kód deklarácií tried BookData a TransactionData zdroj: vlastné spracovanie

public class BookData{

public string Id { get; set; }

public string Nazov { get; set; }

public string Autor1 { get; set; }

public string Autor2 { get; set; }

public string Kategoria { get; set; }

public string Isbn { get; set; }

public string Jazyk { get; set; }

public string Pocet\_stran { get; set; }

public string Vazba { get; set; }

public string Rok\_vydania { get; set; }

public string Vydavatelstvo { get; set; }

public decimal Predajna\_cena { get; set; }

public decimal Nakupna\_cena { get; set; }

public double Marza { get; set; }

public double Zisk\_kus { get; set; }

public string Obsah { get; set; }

public string Priemerne\_hodnotenie { get; set; }

public string ImageName { get; set; }

public byte[] ImageBytes { get; set; }

}

public class TransactionData

{

public string Id\_transakcie { get; set; }

public string Id\_knihy { get; set; }

public string Datum { get; set; }

public string Typ\_transakcie { get; set; }

public int Mnozstvo { get; set; }

public double Cena\_za\_jednotku { get; set; }

public double Celkovo\_cena { get; set; }

public int Aktualne\_mnozstvo\_na\_sklade { get; set; }}

### Popis newebových metód

Začneme popisom newebových metód, ktoré zabezpečujú pomocnú funkcionalitu pre webové metódy.

**Metódy na načítanie XML**

Tieto tri metódy sa používajú na načítanie súboru XML. Rozdiel medzi nimi je v type vráteného objektu:

**Metóda LoadXmlDocument*-*** vracia objekt typu *XmlDocument* z menného priestoru *System.XML.*

**Metóda LoadXElement*-*** vracia objekt typu *XElement* z menného priestoru *System.Xml.Linq*.

**Metóda LoadXDocument*-*** vracia objekt typu *XDocument* z menného priestoru *System.Xml.Linq.*

Dôvodom, prečo máme až tri metódy na načítanie XML je ten, že ďalej prezentované webové služby pracujú s rôznymi typmi objektov XML, a taktiež odlišnými dopytovacími jazykmi. Pri jazyku LINQ načítavame XML buď ako XElement alebo XDocument a pri Xpathe ako XmlDocument.

Všetky tri metódy používajú podobné spracovanie výnimiek na zachytenie chýb. Ak sa súbor nenájde, zachytí sa výnimka *FileNotFoundException* a vráti sa chybová správa so stavovým kódom 404. Ak dôjde k chybe pri parsovaní XML, zachytí sa výnimka *XmlException* a vráti sa chybová správa so stavovým kódom 400. V prípade akýchkoľvek iných chýb sa zachytí výnimka a vráti sa chybová správa so stavovým kódom 500.

Obrázok 21: Zdrojový kód členskej newebovej metódy LoadXmlDocument zdroj: vlastné spracovanie

public XmlDocument LoadXmlDocument(string filePath)

{

// Vytvorenie nového objektu XmlDocument

XmlDocument doc = new XmlDocument();

try

{

// Načítanie XML dokumentu zo zadaného súboru

doc.Load(filePath);

return doc;

}

catch (FileNotFoundException ex)

{

// Ak sa nevie nájsť zadaný súbor, nastaví sa status kód na 404 a vráti sa popisná chybová správa

Context.Response.StatusCode = 404;

Context.Response.StatusDescription = "Zadaný súbor sa nenašiel.";

Context.Response.Write("Zadaný súbor sa nenašiel: " + ex.Message);

return null;

}

catch (XmlException ex)

{

// Ak sa vyskytne chyba pri parsovaní XML dokumentu, nastaví sa status kód na 400 a

//vráti sa popisná chybová správa

Context.Response.StatusCode = 400;

Context.Response.StatusDescription = "Chyba pri parsovaní XML dokumentu.";

Context.Response.Write("Chyba pri parsovaní XML dokumentu: " + ex.Message);

return null;

}

catch (Exception ex)

{

// Ak sa vyskytne akákoľvek iná chyba pri načítaní dokumentu, nastaví sa status kód

//na 500 a vráti sa popisná chybová správa

Context.Response.StatusCode = 500;

Context.Response.StatusDescription = "Chyba pri načítaní dokumentu.";

Context.Response.Write("Chyba pri načítaní dokumentu: " + ex.Message);

return null;

}

}

...

Obrázok 22: Časť zdrojového kódu členskej newebovej metódy LoadXElement zdroj: vlastné spracovanie

// Načítanie XML elementu zo zadaného súboru

XElement element = XElement.Load(filePath);

return element;

...

Obrázok 23: Časť zdrojového kódu členskej newebovej metódy LoadXDocument zdroj: vlastné spracovanie

...

// Načítanie XML dokumentu zo zadaného súboru

XDocument document = XDocument.Load(filePath);

return document;

...

Obrázok 24: Zdrojový kód členskej newebovej metódy CreateTimestamp zdroj: vlastné spracovanie

public string CreateTimestamp()

{

// Formátovať aktuálny dátum a čas pomocou vzoru "yyyy-MM-dd HH:mm:ss"

return DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss");

}

Táto metóda vráti aktuálny dátum a čas ako reťazec vo formáte "rrrr-mm-dd hh:mm:ss"

Obrázok 25: Zdrojový kód členskej newebovej metódy WriteToTheFileWithTimeStamp zdroj: vlastné spracovanie

public void WriteToTheFileWithTimeStamp(string path, XmlNodeList data)

{

try

{

// Vytvorenie objektu XmlDocument

XmlDocument doc = new XmlDocument();

XmlElement root;

XmlElement output;

XmlElement timestamp;

XmlDeclaration xmlDeclaration = doc.CreateXmlDeclaration("1.0", "UTF-8", null);

// Kontrola, či súbor existuje

bool fileExists = System.IO.File.Exists(path);

// Ak súbor neexistuje, vytvorí sa nový súbor

if (!fileExists)

{

doc.AppendChild(xmlDeclaration);

root = doc.CreateElement("root");

doc.AppendChild(root);

}

else

{

// Načítanie existujúceho súboru

doc = LoadXmlDocument(path);

root = doc.DocumentElement;

}

// Prechod cez všetky uzly v XmlNodeList

foreach (XmlNode node in data)

{

// Importovanie uzla do XmlDocument

XmlNode importedNode = doc.ImportNode(node, true);

output = doc.CreateElement("output");

timestamp = doc.CreateElement("timestamp");

// Vytvorenie časovej pečiatky

timestamp.InnerText = CreateTimestamp();

// Pridanie importovaného uzla a časovej pečiatky k elementu output

output.AppendChild(importedNode);

output.AppendChild(timestamp);

// Pridanie elementu output ku koreňovému elementu

root.AppendChild(output);

}

// Uloženie zmeneného súboru

doc.Save(path);

}

catch (FileNotFoundException ex)

{

// Výpis chyby, ak súbor sa nenašiel

Console.WriteLine("Súbor sa nenašiel: " + ex.Message);

}

catch (XmlException ex)

{

// Výpis chyby, ak nastala chyba v XML súbore

Console.WriteLine("Chyba XML: " + ex.Message);

}

catch (IOException ex)

{

// Výpis chyby, ak nastala chyba pri vstupe/výstupe

Console.WriteLine("Chyba I/O: " + ex.Message);

}

catch (UnauthorizedAccessException ex)

{

// Výpis chyby, ak nemáte oprávnenie k prístupu k súboru

Console.WriteLine("Nedovolený prístup: " + ex.Message);

}

}

Táto metóda zapisuje údaje vo formáte XML s časovou pečiatkou do súboru určeného parametrom *path*. Metóda má dva parametre a to reťazec *path*, ktorý predstavuje cestu k súboru, do ktorého sa má zapisovať a parameter *data* typu XmlNodeList, ktorý predstavuje zoznam XML uzlov, ktoré sa majú zapísať do súboru. Na začiatku metódy je vytvorený objekt triedy XmlDocument *doc* a sú deklarované premenné typu XmlElement *root, output a timestamp* a premenná *xmlDeclaration* typuXmlDeclaration, do ktorej je pomocou metódy *CreateXmlDeclaration()* vložená XML deklarácia. Metóda potom kontroluje, či súbor uvedený v parametri *path* existuje alebo nie. Ak súbor neexistuje, vytvorí sa nový súbor, do ktorého sa pridá XML deklarácia a koreňový element *root*. Ak súbor existuje, existujúci súbor sa načíta do objektu *doc* a do premennej *root* je vložený koreňový element existujúceho XML dokumentu. Metóda potom prechádza cez každý uzol v *XmlNodeListe* *data* a importuje každý uzol, ktorý bol metóde odovzdaný ako argument do *XmlDocument* objektu *importedNode*. Druhým argumentom metódy ImportNode() je logická hodnota, ktorá určuje, či sa majú rekurzívne importovať detské uzly. Ak ju nastavíme na hodnotu true, všetky detské uzly importovaného uzla sa tiež importujú a pridajú do objektu *importedNode*. Metóda ďalej vytvorí elementy s názvami *output* a  *timestamp* a pripojí ich na koniec premennej *output*. Vnútorný text premennej *timestamp* je naplnený pomocou metódy *CreateTimestamp()*, ktorú sme si popísali nad popisom tejto metódy. Nakoniec je premenná *output*  pripojená k premennej *root,* ktorá predstavuje koreňový element. Po spracovaní všetkých uzlov metóda uloží upravený objekt *doc* do súboru určeného parametrom *path*. Metóda obsahuje aj kód na spracovanie chýb v prípade, že sa počas vykonávania metódy vyskytnú nejaké výnimky, ako napríklad FileNotFoundException, XmlException, IOException alebo UnauthorizedAccess Exception. Ak sa vyskytne niektorá z týchto výnimiek, metóda zapíše do objektu Response aktuálneho kontextu HTTP chybovú správu.

Obrázok 26: Zdrojový kód členskej newebovej metódy SaveWebMethodResult zdroj: vlastné spracovanie

public void SaveWebMethodResult(string xmlResult, string methodName, string[] parameters, string path)

{

// Skontroluje, či adresár existuje, ak nie, vytvorí ho

if (!Directory.Exists(path))

{

Directory.CreateDirectory(path);

}

// Vytvorí názov súboru pre XML súbor pomocou aktuálneho dátumu a času, názvu metódy

//a parametrov

string fileName = DateTime.Now.ToString("yyyyMMddHHmmss") + "\_" + methodName + "\_" +

string.Join("\_", parameters) + ".xml";

// Uloží XML výsledok do súboru

string fullPath = Path.Combine(path, fileName);

System.IO.File.WriteAllText(fullPath, xmlResult);

}

Táto metóda na rozdiel od členskej metódy *WriteToTheFileWithTimeStamp* vytvorí po každom volaní nový XML súbor, v mene ktorého sa nachádza názov metódy, ktorý ju volal, hodnoty parametrov aktuálneho volania metódy a taktiež časová pečiatka.

Metóda prijíma štyri parametre:

*xmlResult:* reťazec obsahujúci XML výsledok,

*methodName*: reťazec obsahujúci názov webovej metódy,

*parameters*: pole reťazcov obsahujúce parametre odovzdané webovej metóde,

*path*: reťazec obsahujúci cestu k adresáru, do ktorého sa má uložiť súbor XML.

Metóda najprv skontroluje, či adresár špecifikovaný parametrom *path* existuje. Ak neexistuje, metóda vytvorí adresár pomocou metódy *CreateDirectory* triedy *Directory*. Metóda potom vytvorí názov XML súboru pomocou aktuálneho dátumu a času, názvu webovej metódy a parametrov odovzdaných webovej metóde. Nakoniec metóda uloží výsledok XML do súboru s názvom súboru a adresárovou cestou zadanou pomocou metódy *WriteAllText* triedy *File*.

### Popis webových metód

Ďalej si popíšeme vystavené metódy webovej služby. Tieto metódy na rozdiel od newebových metód označujeme pomocou atribútu [WebMethod].

Naše webové služby môžeme rozdeliť na 2 typy a to:

**Entitné metódy-** slúžia na vybratie jedného alebo viacerých údajov o knihe alebo transakcii v nezmenenej podobe presne tak, ako sú zapísané v XML súboroch. Patria sem aj metódy na pridanie, aktualizáciu a vymazanie knihy alebo transakcie.

**Spracovateľské metódy**- sú metódy, v ktorých sa okrem výberu údajov vykonáva aj nejaká spracovateľská logika. Pri týchto metódach sme naplno využili silu jazyka LINQ, ktorý je zabudovaný v  C#. Ten nám umožnil vykonávať s údajmi funkcie ako je projekcia podmnožiny údajov, zoskupovanie, spájanie na základe spoločného elementu, rôzne agregačné funkcie, filtrácia, ale aj triedenie údajov a mnoho ďalších.

Obrázok 27: Zdrojový kód webovej metódy AddBook zdroj: vlastné spracovanie

[WebMethod (Description = "prida do xml súboru záznam o novej knihe")]

public void AddBook()

{

try

{

var request = HttpContext.Current.Request;

// vytvorí sa nová inštancia triedy BookData

var bookData = new BookData();

// Kontrola parametrov z requestu a priradenie hodnôt.

// Ak niektorý parameter chýba, funkcia vráti chybový kód a príslušnú hlášku

if (string.IsNullOrEmpty(request["nazov"]))

{

Context.Response.StatusCode = 500;

Context.Response.Write("Zadajte parameter nazov");

return;

}

else

{

bookData.Nazov = request["nazov"];

}

//...

// !! pozor skrátene!! podobná kontrola sa vykoná pre všetky polia (plná verzia

//v prílohe práce )

if (string.IsNullOrEmpty(request["predajna\_cena"]))

{

Context.Response.StatusCode = 500;

Context.Response.Write("Zadajte parameter predajna\_cena");

return;

}

else

{

bookData.Predajna\_cena = decimal.Parse(request["predajna\_cena"]);

}

if (request.Files.Count == 0)

{

Context.Response.StatusCode = 500;

Context.Response.Write("Vlozte obrazok knihy pozadovana velkost formatu je

430\*600px");

return; }

// získanie obrázka zo súboru

var postedFile = request.Files[0];

// Tento riadok nastaví členskú premennú ImageName objektu bookData na názov súboru

nahraného obrázka.

// Metóda Path.GetFileName získa názov súboru z úplnej cesty k súboru, ktorá je

//uložená vo vlastnosti FileName objektu postedFile.

bookData.ImageName = Path.GetFileName(postedFile.FileName);

// Tento riadok deklaruje v objekte bookData nové pole bajtov s názvom ImageBytes,

ktorého dĺžka sa rovná vlastnosti ContentLength objektu postedFile.

//Toto pole bajtov sa použije na uloženie binárnych údajov nahraného obrázka.

bookData.ImageBytes = new byte[postedFile.ContentLength];

//Tento riadok načíta binárne údaje nahraného obrázka z člennskej premennej

// InputStream objektu postedFile a uloží ich do členneskej premmenej ImageBytes objektu bookData.

//Metóda Read objektu InputStream prečíta zo streamu zadaný počet bajtov a uloží

//ich do zadaného poľa bajtov, pričom začína na zadanom offsete (ktorý je v tomto prípade 0).

//Počet bajtov, ktoré sa majú prečítať, je určený vlastnosťou ContentLength objektu

//postedFile.

postedFile.InputStream.Read(bookData.ImageBytes, 0, postedFile.ContentLength);

// načítanie xml súboru s knihami a získanie hodnoty najväčšieho id

string xmlFilePath = fileBookInfo;

XElement xmlDoc = LoadXElement(xmlFilePath);

var lastBookId = xmlDoc.Element("books").Elements("book").Max(x => (int?)x.Element("id")) ?? 0;

bookData.Id = (lastBookId + 1).ToString();

// pridanie novej knihy do xml súboru

XElement bookElement = new XElement("book",

new XElement("id", bookData.Id),

new XElement("nazov", bookData.Nazov),

new XElement("autori",

new XElement("autor1", bookData.Autor1),

new XElement("autor2", bookData.Autor2)),

new XElement("kategoria", bookData.Kategoria),

new XElement("isbn", bookData.Isbn),

new XElement("jazyk", bookData.Jazyk),

new XElement("pocet\_stran", bookData.Pocet\_stran),

new XElement("vazba", bookData.Vazba),

new XElement("rok\_vydania", bookData.Rok\_vydania),

new XElement("vydavatelstvo", bookData.Vydavatelstvo),

new XElement("predajna\_cena", bookData.Predajna\_cena),

new XElement("nakupna\_cena", bookData.Nakupna\_cena),

new XElement("marza", (bookData.Predajna\_cena - bookData.Nakupna\_cena) /

bookData.Predajna\_cena ),

new XElement("zisk\_kus", bookData.Predajna\_cena - bookData.Nakupna\_cena),

new XElement("obsah", bookData.Obsah),

new XElement("priemerne\_hodnotenie", bookData.Priemerne\_hodnotenie),

new XElement("obrazok", "../img/" + bookData.Id + ".jpg"));

xmlDoc.Element("books").Add(bookElement);

xmlDoc.Save(xmlFilePath);

// uloženie obrázka do súboru

string imageFilePath = Path.Combine(Server.MapPath("~/img"), "../img/" +

bookData.Id + ".jpg");

System.IO.File.WriteAllBytes(imageFilePath, bookData.ImageBytes);

Context.Response.Write("kniha uspesne pridana");

}

catch (Exception ex)

{

Context.Response.StatusCode = 500;

Context.Response.Write("Error: " + ex.Message); } }

Táto metóda slúži na pridanie nového záznamu o knihe do XML dátovej základne. Metóda prijíma požiadavku POST s parametrami v tele požiadavky a súbor s obrázkom ako binárnu prílohu.

Metóda sa začína získaním aktuálneho HttpContext objektu získaného z aktuálnej HTTP požiadavky. Ďalej je vytvorená inštancia triedy *BookData,* ktorá sa použije na uloženie informácií o novej knihe. Metóda skontroluje, či parametre požiadavky nie sú prázdne hodnoty alebo nemajú hodnotu null a priradí ich hodnoty členským premenným objektu bookData. Ak niektorý parameter chýba alebo je neplatný, metóda vráti chybový kód 500 a príslušnú chybovú správu.

Metóda tiež kontroluje, či je k požiadavke pripojený súbor s obrázkom obalu knihy. Ak nie je pripojený žiadny obrázok, metóda vráti chybovú správu. Ak je obrázok pripojený, tak metóda prečíta jeho obsah do poľa bajtov a priradí toto pole do členskej premennej *ImageBytes* objektu *bookData*. Metóda tiež nastaví názov súboru s obrázkom ako členskú premennú objektu *bookData.Nazov*. Ďalej metóda načíta súbor XML, ktorý obsahuje záznamy o knihách a vyberie z neho najvyššie ID knihy. Metóda potom zvýši toto ID o jedna a priradí ho k členskej premennej *Id* objektu *bookData*. Týmto sa zabezpečí, že nová kniha bude mať vždy unikátne ID a bude väčšie o jedna ako predošlý záznam. Po uložení údajov o parametroch knihy a obrázka do členských premenných objektu *bookData* metóda vytvorí nové XML elementy pre všetky parametre knihy a spolu s relatívnou cestou k obrázku ich pridá do súboru XML. Hodnoty elementov *zisk* a *marza* sú vypočítané priamo pri vytváraní elementov, a to na základe hodnôt v členských premenných *Nakupna\_cena* a *Predajna\_cena.* Pri konci je obrázok uložený aj do priečinka na serveri/disku, pričom ako názov súboru použije ID knihy. Ak sa počas tohto procesu vyskytnú nejaké chyby, metóda vráti klientovi chybovú správu. Ak je všetko úspešné, metóda vráti správu, že kniha bola úspešne pridaná.

Obrázok 28: Zdrojový kód webovej metódy UpdateBook zdroj: vlastné spracovanie

[WebMethod(Description = "aktualizuje záznam o jednej knihe na základe id")]

public void UpdateBook()

{

try

{

var request = HttpContext.Current.Request;

// vytvorí sa nová inštancia triedy BookData

var bookData = new BookData();

// získanie údajov o knihe z premennej request a ich priradenie do členských

//premenných book.Data

bookData.Id = request["id"];

bookData.Nazov = request["nazov"];

bookData.Autor1 = request["autor1"];

bookData.Autor2 = request["autor2"];

bookData.Kategoria = request["kategoria"];

bookData.Isbn = request["isbn"];

bookData.Jazyk = request["jazyk"];

bookData.Pocet\_stran = request["pocet\_stran"];

bookData.Vazba = request["vazba"];

bookData.Rok\_vydania = request["rok\_vydania"];

bookData.Vydavatelstvo = request["vydavatelstvo"];

bookData.Predajna\_cena = decimal.Parse(request["predajna\_cena"]);

bookData.Nakupna\_cena = decimal.Parse(request["nakupna\_cena"]);

bookData.Obsah = request["obsah"];

bookData.Priemerne\_hodnotenie = request["priemerne\_hodnotenie"];

if (request.Files.Count > 0)

{

var postedFile = request.Files[0];

bookData.ImageName = Path.GetFileName(postedFile.FileName);

bookData.ImageBytes = new byte[postedFile.ContentLength];

postedFile.InputStream.Read(bookData.ImageBytes, 0, postedFile.ContentLength);

}

string xmlFilePath = fileBookInfo;

XElement xmlDoc = LoadXElement(xmlFilePath);

// Nájdenie prvku v XML súbore s id knihy rovnakým ako členskej premennej

//bookData.Id

var bookElement = xmlDoc.Element("books").Elements("book").FirstOrDefault(x => x.Element("id").Value == bookData.Id);

if (bookElement == null)

{

Context.Response.StatusCode = 500;

Context.Response.Write("kniha zo zadanym id sa nenasla");

}

// Aktualizovanie hodnôt elementov prvku bookElement s hodnotami z premennej

//bookData

bookElement.SetElementValue("nazov", bookData.Nazov);

bookElement.Element("autori").SetElementValue("autor1", bookData.Autor1);

bookElement.Element("autori").SetElementValue("autor2", bookData.Autor2);

bookElement.SetElementValue("kategoria", bookData.Kategoria);

bookElement.SetElementValue("isbn", bookData.Isbn);

bookElement.SetElementValue("jazyk", bookData.Jazyk);

bookElement.SetElementValue("pocet\_stran", bookData.Pocet\_stran);

bookElement.SetElementValue("vazba", bookData.Vazba);

bookElement.SetElementValue("rok\_vydania", bookData.Rok\_vydania);

bookElement.SetElementValue("vydavatelstvo", bookData.Vydavatelstvo);

bookElement.SetElementValue("predajna\_cena", bookData.Predajna\_cena);

bookElement.SetElementValue("nakupna\_cena", bookData.Nakupna\_cena);

bookElement.SetElementValue("obsah", bookData.Obsah);

bookElement.SetElementValue("priemerne\_hodnotenie", bookData.Priemerne\_hodnotenie);

bookElement.SetElementValue("marza", (bookData.Predajna\_cena - bookData.Nakupna\_cena) / bookData.Predajna\_cena ); bookElement.SetElementValue("zisk\_kus", bookData.Predajna\_cena - bookData.Nakupna\_cena);

// Overí, či boli nahrané dáta o obale knihy, a či nie sú prázdne

if (bookData.ImageBytes != null && bookData.ImageBytes.Length > 0)

{// Vytvorí cestu k súboru s obrázkom obalu knihy a uloží dáta o obale knihy na

//túto cestu

string imageFilePath = Path.Combine(HttpContext.Current.Server.MapPath("~/img"), "../img/" + bookData.Id + ".jpg");

System.IO.File.WriteAllBytes(imageFilePath, bookData.ImageBytes);

}

// Uloží zmeny v XML súbore s informáciami o knihách

xmlDoc.Save(xmlFilePath);

// Vypíše správu o úspešnom vykonaní aktualizácie knihy

Context.Response.Write("kniha bola uspesne aktualizovana");

}

// V prípade výskytu chyby, nastaví HTTP status code na 500 a vypíše chybovú správu

catch (Exception ex)

{

Context.Response.StatusCode = 500;

Context.Response.Write("Error: " + ex.Message);

}

}

Táto metóda aktualizuje záznam o jednej knihe na základe jej ID.  Aktualizovaný záznam s novými údajmi o knihe zapisuje do XML súboru a aktualizovaný obrázok je nahraný do diskového súboru s obrázkami obalov kníh. Metóda sa podobne ako *AddBook* začína získaním objektu požiadavky HTTP, ktorá obsahuje nové údaje o knihe získané napríklad pomocou HTML formulára. Vytvorí sa nová inštancia triedy *BookData*. Metóda potom načíta hodnoty jednotlivých polí z objektu *request* a priradí ich príslušným členským premenným inštancie triedy *BookData*. Ak je v požiadavke nahraný obrázok, metóda načíta jeho údaje do poľa bajtov a priradí ich členskej premennej *ImageBytes* inštancie *BookData.* Zároveň nastaví členskú premennú *ImageName* na názov súboru. Metóda potom načíta XML súbor, ktorý obsahuje informácie o všetkých knihách. Metóda ďalej nájde v tomto súbore element knihy, ktorý má rovnaké ID ako aktualizovaná kniha. Ak sa nájde zodpovedajúci element knihy, priradí sa do premennej *bookElement*. Ak je premenná *bookElement* nulová, znamená to, že v súbore XML nebola nájdená žiadna zodpovedajúca kniha, takže metóda nastaví stavový kód na 500 a vráti klientovi chybovú správu. Ak sa nájde zodpovedajúci element knihy, metóda aktualizuje jeho prvky novými hodnotami z inštancie *BookData*. To sa vykoná pomocou metódy *SetElementValue* triedy *XElement*. Metóda vypočíta nové hodnoty pre elementy *marza* a *zisk\_kus* na základe nových hodnôt členských premenných *Predajna\_cena* a *Nakupna\_cena* inštancie BookData. Ak členská premenná *ImageBytes* inštancie *BookData* nie je nulová a má dĺžku väčšiu ako 0, metóda uloží pole bajtov ako súbor s obrázkom do adresára *img* na serveri/disku. Nakoniec metóda uloží zmeny do súboru XML a vráti správu o úspechu klientovi. Ak sa počas niektorého z krokov vyskytne výnimka, metóda nastaví stavový kód HTTP na 500 a klientovi vráti chybovú správu.

Obrázok 29: Zdrojový kód webovej metódy DeleteBook zdroj: vlastné spracovanie

[WebMethod(Description = "zmaze knihu podla zazadného id ")]

public void DeleteBook(string id)

{

try

{

// Cesta k súboru s informáciami o knihách

string xmlFilePath = fileBookInfo;

// Načíta XML dokument z daného súboru

XElement xmlDoc = LoadXElement(xmlFilePath);

// Nájde knihu s daným ID

XElement bookToDelete = xmlDoc.Element("books").Elements("book").FirstOrDefault(x

=> x.Element("id").Value == id);

if (bookToDelete != null)

{

// Odstráni prvok "book" z XML súboru

bookToDelete.Remove();

xmlDoc.Save(xmlFilePath);

// Vymaže obrázok z filesystému

string imageFilePath = Path.Combine(Server.MapPath("~/img"), "../img/" + id +

".jpg");

System.IO.File.Delete(imageFilePath);

// Vypíše správu o úspešnom vymazaní knihy

Context.Response.Write("kniha uspesne zmazana");

}

else

{// Ak sa kniha s daným ID nenašla, vráti chybový kód a príslušnú správu

Context.Response.StatusCode = 500;

Context.Response.Write("kniha so zadanym id nebola najdena");

}

}

catch (Exception ex)

{// Ak nastala chyba, vypíše chybovú správu

Context.Response.Write("Error: " + ex.Message);

}

}

Táto metóda slúži na odstránenie knihy na základe zadaného ID knihy. Metóda má jeden reťazcový parameter *id*, ktorý predstavuje ID knihy, ktorá sa má vymazať. Metóda najprv načíta XML súbor obsahujúci informácie o knihách a potom v dokumente vyhľadá pomocou jazyka LINQ knihu so zadaným ID. Ak sa kniha nájde, tak je vymazaná a  XML dokument je nanovo uložený.

Okrem toho metóda odstráni zo súborového systému aj obrázok obalu danej knihy. Predpokladá sa, že obrázok obalu je uložený v priečinku s názvom *img* a jeho názov je rovnaký ako ID knihy s príponou .jpg. Ak sa kniha so zadaným ID v dokumente XML nenachádza, metóda nastaví stavový kód odpovede HTTP na 500 a do odpovede zapíše chybovú správu.

**Webové metódy SinglebookDataById, SinglebookDataByName a SinglebookDataByIsbn**

Všetky tri metódy slúžia na získanie informácie o jednej knihe na základe rôzneho parametru. Telá týchto webových služieb sú veľmi podobné, a preto si uvedieme len jednu ako príklad a ostatné môžete nájsť v elektronickej prílohe tejto práce. Všetky tri metódy preberajú príslušný parameter *id/názov/isbn* a pomocou jazyka XPath nájdu knihu s príslušnou hodnotou parametra v XML dokumente. Ak sa kniha nájde, serializuje sa do reťazca JSON a vráti sa v odpovedi. Ak sa kniha nenájde alebo ak sa vyskytne chyba, vráti sa príslušný stavový kód a chybová správa. Všetky metódy sa zdajú byť podobné z hľadiska funkčnosti a spracovania chýb. Metóda "SinglebookDataByName" je však flexibilnejšia, pokiaľ ide o vyhľadávanie kníh na základe ich názvu, pretože dokáže vyhľadávať bez rozlišovania veľkých a malých písmen.

Obrázok 30: Zdrojový kód webovej metódy SinglebookDataByName zdroj: vlastné spracovanie

[WebMethod(Description = "odošle udaje o jednej knihe na zaklade nazvu v json formate ")]

public void SinglebookDataByName(string name)

{

try

{

// Načítanie XML dokumentu

XmlDocument doc = LoadXmlDocument(fileBookInfo)

// Ak sa dokument nedá načítať, nastaví sa stavový kód na 500 a opíše sa chyba

if (doc == null)

{

Context.Response.StatusCode = 500;

Context.Response.StatusDescription = "Dokument nemohol byť načítaný";

return;

}

// Ak nie je zadaný názov knihy, nastaví sa stavový kód na 400 a vypíše sa chybová

//hláška

if (string.IsNullOrEmpty(name))

{

Context.Response.StatusCode = 400;

Context.Response.StatusDescription = "Nezadali ste hodnotu mena knihy";

Context.Response.Write("Nezadali ste nazov knihy");

return;

}

// Prevod názvu knihy na malé písmená

name = name.ToLower();

// Vyhľadanie záznamu o knihe s daným názvom Xpathu ukážka, kde je vhodný pre svoju

//jednoduchosť

XmlNodeList nodeListBook = doc.SelectNodes("Bookstore/books/book[translate(nazov,'ABCDEFGHIJKLMNOPQRSTUVWXYZÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖØÙÚÛÜÝÞŸŽŠŒ','abcdefghijklmnopqrstuvwxyzàáâãäåæçèéêëìíîïðñòóôõöøùúûüýþÿžšœ') = \"" + name + "\"]");

// Ak sa záznam nenašiel, vyhodí sa výnimka a nastaví sa stavový kód na 500

if (nodeListBook == null || nodeListBook.Count == 0)

{

Context.Response.StatusCode = 500;

Context.Response.StatusDescription = "Chyba pri spracovávaní požiadavky";

throw new Exception("Záznam pre daný názov nebol nájdený");

}

// Zapísanie výsledku do súboru s časovou pečiatkou

WriteToTheFileWithTimeStamp(fileOutputSingleSearch, nodeListBook);

// Výpis výsledku v podobe JSONu

WriteToTheFileWithTimeStamp(fileOutputSingleSearch, nodeListBook);

Context.Response.BinaryWrite(System.Text.Encoding.UTF8.GetPreamble());

Context.Response.Write(JsonConvert.SerializeXmlNode(nodeListBook.Item(0), Formatting.Indented));

}

//Ak došlo k výnimke, spracuj chybu

catch (Exception ex)

{

Context.Response.StatusCode = 500;

Context.Response.StatusDescription = "Chyba pri spracovavani poziadavky";

Context.Response.Write("Error: " + ex.Message);

}

}

**Webové metódy GetListAllBooks a GetListAllTransactions**

Tieto webové metódy uvádzame tiež spolu, lebo ich telá sú veľmi podobné. Funkciou týchto webových metód je načítať zoznam kníh/transakcií z XML dokumentu a vrátiť vo formáte JSON ako HTTP odpoveď. Metóda najprv načíta dokument XML, a potom vyberie všetky uzly knihy/transakcie pomocou výrazu XPath. Ak v zozname nie sú žiadne knihy/transakcie alebo je prázdny, metóda odošle klientovi stavový kód a chybovú správu. Ako príklad si uvedieme metódu GetListAllBooks.

Obrázok 31: Zdrojový kód webovej metódy GetListAllBooks zdroj: vlastné spracovanie

[WebMethod(Description = "získa zoznam všetkých kníh v json formáte")]

public void GetListAllBooks()

{ // vytvorenie a načítanie XML dokumentu

XmlDocument doc = LoadXmlDocument(fileBookInfo);

// získanie zoznamu všetkých kníh pomocou Xpathu ukážka kde je vhodný pre svoju

//jednoduchosť

XmlNodeList AllBook = doc.SelectNodes("Bookstore/books");

// v prípade, že zoznam je prázdny alebo neexistuje

if (AllBook == null || AllBook.Item(0) == null)

{// nastavenie chybového kódu na 500

Context.Response.StatusCode = 500;

// popis chyby

Context.Response.StatusDescription = "Chyba pri spracovávaní dát v dokumente";

return;

}

// nastavenie UTF-8 sady pre http response

Context.Response.BinaryWrite(System.Text.Encoding.UTF8.GetPreamble());

Context.Response.Write(JsonConvert.SerializeXmlNode(AllBook.Item(0),

Formatting.Indented));

// serializácia XML uzla ako JSON a odoslanie ako http odpoveď s formátovaním

}

**Metódy Addtransaction UpdateTransaction a DeleteTransaction**

Tieto tri metódy slúžia na pridanie, aktualizáciu a odstránenie transakcie. Ich telá sú veľmi podobné ako pri knihách, jedine s tým rozdielom, že tieto metódy nezískavajú údaje pomocou aktuálneho HttpContext objektu získaného z aktuálnej HTTP požiadavky, ale priamo pomocou prevzatia hodnoty z argumentov metódy. Pri knihách sme využili triedu Httpcontext a jej členskú premennú, hlavne z dôvodu ľahkého prístupu k nahratému obrázku z formulára klienta. Keďže transakcie neobsahujú žiadne obrázky ani súbory, tak tento prístup nebolo potrebné použiť a vystačili sme si s obyčajnými parametrami metódy. Ukážky kódov týchto metód si nebudeme v tejto časti uvádzať, ale môžete ich nájsť v elektronickej prílohe práce.

[WebMethod (Description = "poskytne zoradené údaje o počtoch kníh na základe atribútu v zadanom období ")]

Obrázok 32: Zdrojový kód webovej metódy SortedBookAmoutsByDateAndAtribute zdroj: vlastné spracovanie

public void SortedBookAmoutsByDateAndAtribute(string selectedAtribute, string selectedValueAtribute, string startDate, string endDate, string sortField, string sortOrder)

{

string[] parameters = { selectedAtribute, selectedValueAtribute, startDate, endDate,

sortField, sortOrder, };

if (selectedAtribute != "vsetky")

{

// Skontroluje, či sú vložene všetky parametre

if (string.IsNullOrEmpty(selectedAtribute) ||

string.IsNullOrEmpty(selectedValueAtribute) ||

string.IsNullOrEmpty(startDate) ||

string.IsNullOrEmpty(endDate) ||

string.IsNullOrEmpty(sortField) ||

string.IsNullOrEmpty(sortOrder))

{

Context.Response.StatusCode = 400;

Context.Response.Write("Prosím zadajte všetky vstupné parametre");

return;

}

}

else

{

if (string.IsNullOrEmpty(selectedAtribute) ||

string.IsNullOrEmpty(startDate) || string.IsNullOrEmpty(endDate) || string.IsNullOrEmpty(sortField) || string.IsNullOrEmpty(sortOrder))

{

Context.Response.StatusCode = 400;

Context.Response.Write("Prosím zadajte všetky vstupné parametre");

return;

}

}

// skontroluje či sú hodnoty s sortorder "Ascending" alebo "Descending "

if (!sortOrder.Equals("ascending") && !sortOrder.Equals("descending"))

{

Context.Response.StatusCode = 400;

Context.Response.Write("Input Error: sortOrder must be either 'Ascending' or

'Descending'");

return;

}

// skontroluje či sú dostal správnu z vybraných hodnôt, podľa ktorých ma zoradiť údaje

if (!sortField.Equals("nazov") && !sortField.Equals("pocet\_stran") && !sortField.Equals("rok\_vydania") && !sortField.Equals("predajna\_cena") && !sortField.Equals("nakupna\_cena") && !sortField.Equals("priemerne\_hodnotenie"))

{

Context.Response.StatusCode = 400;

Context.Response.Write("Input Error: zoradovat možete podla tých atribútov 'nazov' 'pocet\_stran''rok\_vydania','predajna\_cena''nakupna\_cena','priemerne\_hodnotenie'");

return;

}

// Definujeme premenné start a end typu DateTime

DateTime start, end;

// Overíme, či vstupné dátumy sú v správnom formáte. Ak nie, nastavíme chybový kód a

//vypíšeme chybovú správu.

if (!DateTime.TryParse(startDate, out start) || !DateTime.TryParse(endDate, out end))

{

Context.Response.StatusCode = 400;

Context.Response.Write("Zadajte validný formát dútumu ( napr. 'yyyy-MM-dd')");

return;

}

// Overíme, či začiatočný dátum nie je neskôr ako koncový dátum. Ak áno, nastavíme

//chybový kód a vypíšeme chybovú správu

if (start > end)

{

Context.Response.StatusCode = 400;

Context.Response.Write("Začiatočný dátum nemože byť neskorej ako konečný dátum");

return;

}

// Načítame XML súbory obsahujúce informácie o knihách a transakciách s knihami

var booksXml = LoadXElement(fileBookInfo);

var transactionXml = LoadXElement(fileBookTransactionInfo);

// Načítame XML súbory obsahujúce informácie o knihách a transakciách s knihami

var booksXml = LoadXElement(fileBookInfo);

var transactionXml = LoadXElement(fileBookTransactionInfo);

// Použijeme LINQ na spojenie informácií z oboch XML súborov a filtrovanie výsledkov na

//základe vybranej kategórie a hodnoty

if (selectedAtribute != "vsetky")// Kontrola, či bola vybraná konkrétna kategória a nie

// hodnota vsetky

{// Vyberie všetky elementy "book" zo súboru booksXml

var result = from b in booksXml.Descendants("book")

// Spojí elementy zo súboru transactionXml na základe zhody atribútu "id\_knihy" s

//atribútom "id" v súbore booksXml a vytvorí z toho kolekciu

join w1 in transactionXml.Descendants("transakcia") on (string)b.Element("id")

equals (string)w1.Element("id\_knihy") into g

// Vyberie prvý element z kolekcie, ktorý má najbližší dátum ku dátumu "start"

from w1 in g.OrderBy(x => Math.Abs((DateTime.Parse((string)x.Element("datum"))

- start).Ticks)).Take(1)

// Uloží aktuálny počet kusov na sklade z vybraného elementu do premennej startAmount

let startAmount = (int)w1.Element("aktualne\_mnozstvo\_na\_sklade")

// Vyberie prvý element z kolekcie, ktorý má najbližší dátum ku dátumu "end"

from w2 in g.OrderBy(x => Math.Abs((DateTime.Parse((string)x.Element("datum"))

- end).Ticks)).Take(1)

// Filter pre vybranú kategóriu a hodnotu

where (string)b.Element(selectedAtribute) == selectedValueAtribute ||

(selectedAtribute == "autor" && (string)b.Element("autori").Element("autor1") == selectedValueAtribute) || (selectedAtribute == "autor" && (string)b.Element("autori").Element("autor2") == selectedValueAtribute)

select new //Vytvorí nový anonymný typ so zvolenými atribútmi a

//hodnotami

{

BookID = (string)b.Element("id"),

BookName = (string)b.Element("nazov"),

BookNumPages = (int)b.Element("pocet\_stran"),

BookYear = (int)b.Element("rok\_vydania"),

BookSellPrice = (float)b.Element("predajna\_cena"),

BookBuyPrice = (float)b.Element("nakupna\_cena"),

BookRating = (float)b.Element("priemerne\_hodnotenie"),

StartAmount = startAmount,

EndAmount = (int)w2.Element("aktualne\_mnozstvo\_na\_sklade"),

StartDate = start.ToString("yyyy-MM-dd"),

EndDate = end.ToString("yyyy-MM-dd")

};

if (sortOrder == "ascending")// Zoradenie výsledkov podľa vybranej hodnoty

{

switch (sortField)

{

case "nazov":

result = result.OrderBy(r => r.BookName);

break;

case "pocet\_stran":

result = result.OrderBy(r => r.BookNumPages);

break;

case "rok\_vydania":

result = result.OrderBy(r => r.BookYear);

break;

case "predajna\_cena":

result = result.OrderBy(r => r.BookSellPrice);

break;

case "nakupna\_cena":

result = result.OrderBy(r => r.BookBuyPrice);

break;

case "priemerne\_hodnotenie":

result = result.OrderBy(r => r.BookRating);

break;

default:

break;

}

}

else if (sortOrder == "descending")// Zoradenie výsledkov zostupne podľa vybranej hodnoty

{

switch (sortField)

{

case "nazov":

result = result.OrderByDescending(r => r.BookName);

break;

case "pocet\_stran":

result = result.OrderByDescending(r => r.BookNumPages);

break;

case "rok\_vydania":

result = result.OrderByDescending(r => r.BookYear);

break;

case "predajna\_cena":

result = result.OrderByDescending(r => r.BookSellPrice);

break;

case "nakupna\_cena":

result = result.OrderByDescending(r => r.BookBuyPrice);

break;

case "priemerne\_hodnotenie":

result = result.OrderByDescending(r => r.BookRating);

break;

default:

break;

}

}

if (result.Count() == 0)

{

Context.Response.ContentType = "application/json";

Context.Response.Write("Žiadny záznam nespĺňa zadané kritériá");

};

// Vytvoriť objekt s výsledkami a serializovať ho do JSON formátu

var anoresult = new { result, };

var json = JsonConvert.SerializeObject(anoresult, Formatting.Indented);

// Konvertovať JSON reťazec do XML a uložiť ho do premennej doc

XmlDocument doc = JsonConvert.DeserializeXmlNode(json, "Root");

// Serializovať XmlDocument do XML reťazca a uložiť ho pomocou metódy

//SaveWebMethodResult

string xmlString = doc.OuterXml;

SaveWebMethodResult(xmlString, "SortedBookAmoutsByDateAndCategory", parameters, fileAmountFilterPath);

// Odošli odpoveď v JSON formáte

Context.Response.BinaryWrite(System.Text.Encoding.UTF8.GetPreamble());

Context.Response.Write(json);

}//ak selectedAtribute=="všetky" tak sa vykoná táto vetva, ktorá má skoro identickú

//logiku. jediný rozdiel môžeme nájsť v absencii where v linq dopyte

else{// ...táto vetva len v elektronickej prílohe tejto prace}

Táto metóda ako prvá patrí medzi spracovateľské metódy. Úlohou tejto metódy je vrátiť usporiadané informácie o začiatočných a konečných počtoch kníh na sklade v užívateľsky zadanom časovom rozmedzí. Metoda vracia tieto počty buď pre všetky knihy alebo len pre užívateľsky vyfiltrované knihy na základe zvoleného atribútu (napr. “*jazyk“*) a hodnoty tohoto atribútu ( napr.*“sk“* ).

Metóda prijíma šesť vstupných parametrov: *selectedAtribute, selectedValueAtribute, startDate, endDate, sortField a sortOrder.*

Parameter **selectedAtribute** je reťazec, ktorý určuje atribút, podľa ktorého sa majú údaje o knihách filtrovať. Môže v podstate nadobúdať hociktorý platný atribút o knihe (v klientovi to ale obmedzíme len na autora, kategóriu, vydavateľstvo, jazyk a väzbu) alebo hodnotu „všetky“ čo znamená, že nechceme filtrovať, ale vrátiť údaje pre všetky knihy.

Parameter **selectedValueAtribute** je reťazec, ktorý určuje konkrétnu hodnotu vybraného atribútu, podľa ktorého sa má filtrovať. Pri jazyku môže byť táto hodnota napr. *„sk* alebo *„en“*, pri väzbe *„pevná“* alebo *„brožovaná“* a pod.

Parametre **startDate endDate** sú reťazce, ktoré určujú dátum začiatku a konca obdobia, na základe ktorého sa majú údaje filtrovať. Očakáva sa, že tieto dátumy budú vo formáte "rrrr-MM-dd".

Parameter **sortField** je reťazec, ktorý určuje atribút, podľa ktorého sa majú výsledky triediť. Môže nadobúdať jednu z niekoľkých hodnôt vrátane *„nazov“, “pocet\_strán“, “rok\_vydania“, „predajna\_cena“ a „nakupna\_cena“.*

Parameter **sortOrder** je reťazec, ktorý určuje, či sa majú výsledky zoradiť vzostupne alebo zostupne.

V prvej časti metódy sú ošetrené rôzne možné chybové stavy. Metóda kontroluje, či sú zadané všetky požadované parametre a či je zadaná platná hodnota pre spôsob zoradenia. Metóda taktiež kontroluje, či je zadaný platný atribút, podľa ktorého sa bude zoraďovať. Pri dátumoch skontroluje, či majú validný formát a taktiež, či je začiatočný dátum skorej ako konečný dátum. Ak niektorá podmienka nie je splnená, tak sa klientovi odošle chybová správa s príslušným kódom chyby. Ďalšia časť kódu používa na spracovanie údajov jazyk LINQ. Prvý riadok bloku kódu inicializuje premennú *result* a nastaví ju na výsledok dotazu LINQ. Vo všeobecnosti sa v tomto dotaze vykoná nasledujúce:

Načítajú sa obidva XML súbory, z ktorých jeden obsahuje informácie o knihách a druhý informácie o transakciách s knihami.

Operácia *join* používa ID knihy z *booksXml* a spája ho s ID knihy z *transactionXml*. Výsledok tejto operácie spojenia sa uloží do premennej *g*, ktorá obsahuje skupinu transakcií pre každú knihu.

Pred popisom ďalšieho kroku si pripomenieme, že náš XML súbor o transakciách nesie v každom zázname informáciu o aktuálnom množstve danej knihy, v čase po realizácii transakcie. Ak však používateľ vyberie rozsah dátumov, v ktorom nie sú k dispozícii informácie o počte nejakej knihy na sklade, (transakcia s touto knihou nebola v danom časovom rozmedzí realizovaná), tak metóda musí nájsť najbližšie informácie (najbližšie dátumy) o transakciách s danou knihou, aby získala informácie o počte danej knihy na začiatku a na konci daného obdobia. Toto realizujeme pomocou dvoch pomocných LINQ príkazov.

Prvý LINQ príkaz (začínajúci príkazom "from w1") zoradí údaje o transakciách vzostupne podľa absolútneho rozdielu medzi dátumom transakcie a začiatočným dátumom, ktorého hodnota je získaná z argumentu metódy. Potom vyberie prvý prvok z usporiadaného zoznamu, ktorý zodpovedá transakcii najbližšej k dátumu začiatku obdobia.

Druhý LINQ príkaz (začínajúci príkazom "from w2") zoradí vzostupne údaje o transakciách podľa absolútneho rozdielu medzi dátumom transakcie a dátumom konca zadaného obdobia, ktorého hodnota je získaná z argumentu metódy. Potom z usporiadaného zoznamu vyberie prvý prvok, ktorý zodpovedá transakcii najbližšej ku konečnému dátumu.

Pri oboch výpočtoch je použitá vlastnosť objektu *TimeSpan Ticks*, vracajúca počet tikov, ktoré predstavujú najmenšiu jednotku merania času v rámci .NET Frameworku. Jeden Tik predstavuje 100 nanosekúnd. Následne metóda v ďalších krokoch filtruje výsledky na základe vybraného atribútu a hodnoty, napríklad názvu knihy alebo autora a pod.

Metóda ďalej vyberie špecifické informácie o každej knihe vrátane jej ID, názvu, počtu strán, roku vydania, predajnej, nákupnej ceny a priemerného hodnotenia, ako aj počiatočného a konečného množstva kníh na sklade a počiatočného a konečného dátumu a uloží ich ako nový anonymný objekt. Ďalej zoradí výsledky na základe vybraného atribútu a zvoleného poradia.

Ak nie sú k dispozícii žiadne výsledky, metóda vráti chybovú správu. V opačnom prípade metóda serializuje výsledky do formátu JSON a zapíše ich do odpovede. Výsledky sú taktiež serializované do formátu XML a následne v tomto formáte uložené do súboru pomocou nami definovanej newebovej metódy *SaveWebMethodResult*, ktorej opis nájdete na strane 66 tejto práce.

V ukážke si približíme len časť tejto metódy, ktorá filtruje knihy na základe zvoleného atribútu. Druhá nezobrazená časť vykonáva tú istú logiku, ale pre všetky knihy. Celá metóda je dostupná v elektronickej prílohe tejto práce.



Obrázok 33: Ukážka vstupov a výstupov pre metódu SortedBookAmoutsByDateAndAtribute (1. výstup v JSON pre klienta 2. výstup do diskového súboru s časovou pečiatkou a parametrami v XML ) zdroj: vlastné spracovanie

Obrázok 34: Sekcia zdrojového kódu webovej metódy AgregatedStatiscticsAmount zdroj: vlastné spracovanie

...

if (selectedAtribute != "vsetky")

{ // Výber dát zo súborov booksXml a transactionsXml a zlúčenie pomocou JOIN

var result = from b in booksXml.Descendants("book")

join w1 in transactionsXml.Descendants("transakcia") on

(string)b.Element("id") equals (string)w1.Element("id\_knihy") into g

// Výber prvku s minimálnou absolútnou hodnotou rozdielu medzi dátumom

//a počiatkom intervalu

from w1 in g.OrderBy(x =>

Math.Abs((DateTime.Parse((string)x.Element("datum")) - start).Ticks)).Take(1)

let startAmount = (int)w1.Element("aktualne\_mnozstvo\_na\_sklade")

// Výber prvku s minimálnou absolútnou hodnotou rozdielu medzi dátumom

//a koncom intervalu

from w2 in g.OrderBy(x =>

Math.Abs((DateTime.Parse((string)x.Element("datum")) - end).Ticks)).Take(1)

// Výber prvkov, ktoré spĺňajú podmienku vybranej hodnoty atribútu

//alebo autora

where (string)b.Element(selectedAtribute) == selectedValueAtribute ||

(selectedAtribute == "autor" &&

(string)b.Element("autori").Element("autor1") == selectedValueAtribute) ||

(selectedAtribute == "autor" &&

(string)b.Element("autori").Element("autor2") == selectedValueAtribute)

// Vytvorenie nového objektu so zvolenými hodnotami

select new

{

StartAmount = startAmount,

EndAmount = (int)w2.Element("aktualne\_mnozstvo\_na\_sklade"),

StartDate = start.ToString("yyyy-MM-dd"),

EndDate = end.ToString("yyyy-MM-dd")

};

// Výpočet celkového počtu kníh na začiatku a na konci intervalu

double TotalStartAmount = result.Sum(x => x.StartAmount);

double TotalEndAmount = result.Sum(x => x.EndAmount);

// Výpočet maximálneho a minimálneho počtu kníh na začiatku a na konci intervalu

double MaxStartAmount = result.Max(x => x.StartAmount);

double MinStartAmont = result.Min(x => x.StartAmount);

double MaxEndAmount = result.Max(x => x.EndAmount);

double MinEndAmont = result.Min(x => x.EndAmount);

// Výpočet priemeru počtu kníh na začiatku a na konci intervalu

double AvgStartAmont = result.Average(x => x.StartAmount);

double AvgEndAmont = result.Average(x => x.EndAmount);

var serializedResult = new

{

TotalStartAmount,

TotalEndAmount,

MaxStartAmount,

MinStartAmont,

MaxEndAmount,

MinEndAmont,

AvgStartAmont,

AvgEndAmont,

}

...

Táto metóda je takmer identická s metódou *SortedBookAmoutsByDateAndAtribute* avšak namiesto údajov o začiatočných počtoch jednotlivých kníh vracia súhrnné agregované údaje. Táto metóda má len 4 parametre, keďže výstupne dáta v nej netreba triediť. V ukážke je len časť metódy, ktorá je rozdielna oproti metóde *SortedBookAmountsByDateAndAtribtue*. Metóda vracia celkový, maximálny a minimálny počet kníh na začiatku a na konci sledovaného obdobia.



Obrázok 35: Ukážka vstupov a výstupov pri webovej metóde AgregatedStatiscticsAmount zdroj: vlastné spracovanie

Druhý vstup je totožný so vstupom, ktorý bol prezentovaný v ukážkach vstupov a výstupov pri metóde *SortedBokkAmountsByDateAndAtribtue,* avšak výstupné údaje sú už agregované.

**Webové metódy *SortedDrillDownByAtributeDataBetweenTwoDatesSell* a *SortedDrillDownByAtributeDataBetweenTwoDatesCost***

Tieto metódy slúžia na vytvorenie reportu, ktorý poskytuje zoradené informácie o počte a príjmoch alebo o počte a nákladoch z predaných/nakúpených kníh v zadanom období a podľa zadaného atribútu, podľa ktorého sa údaje zoskupujú. Údaje sú poskytnuté v dvoch úrovniach granulity. Metódy sú takmer totožné, rozdiel je len, že jedna rieši počty a príjmy z predaných kníh a druhá počty a náklady z nakúpených kníh od dodávateľov. Celú komplexnú funkcionalitu, týchto metód si demonštrujeme na prvej menovanej webovej metóde s tým, že kód druhej webovej metódy sa nachádza ako elektronická príloha k tejto práci.

Metóda má šesť parametrov, z ktorých každý je reťazec:

**atribute**- predstavuje parameter knihy, podľa ktorého budú transakcie pre jednotlivé knihy zoskupené. V podstate, by tu mohol byť ľubovoľný parameter, ale v klientovi to bude obmedzené len na parametre autori, kategória, vydavateľstvo, jazyk, väzba a rok vydania, pre ktoré zoskupovanie dáva zmysel.

**startDate a endDate-** predstavujú začiatočný a konečný dátum pre obdobie, v ktorom sa údaje o predajoch zoskupujú. Mal by byť vo formáte "*rrrr-MM-dd".*

**sortingField**- predstavuje parameter podľa hodnoty, ktorého sa budú výsledky triediť. Môže mať hodnotu *"nazov"* (abecedné zoradenie) alebo *"hodnota"* (zoradenie podľa číselnej hodnoty).

**sortingOrder**- predstavuje poradie, v ktorom budú výsledky zoradené. Môže mať hodnotu *"ascending"* alebo *"descending".*

**optionalParameter**- predstavuje voliteľný parameter, ktorý sa používa ako dodatočné kritérium triedenia. Môže mať hodnotu *"quantity"* alebo *"revenue"*. Tento parameter sa používa len vtedy, keď je parameter *sortingField* nastavený na hodnotu *"hodnota"*. Parameter hovorí o tom, či jednotlivé podkategórie pre atribút a knihy patriace do tejto podkategórie majú byt zoradené podľa hodnoty počet alebo hodnoty príjem. (zmysel to má vtedy, keď v klientovi vytvárame rôzne grafy pre jednotlivé hodnoty a potrebujeme triediť len podľa príjmov alebo podľa počtu).

Metóda najprv podobne, ako predošlé metódy skontroluje platnosť vstupných parametrov. Ak je niektorý zo vstupných parametrov neplatný, vráti chybovú správu s príslušným stavovým kódom.‘

Ďalej metóda zistí, či zadaný atribút nie je “autori” alebo “autor”. Dôvodom tejto podmienky je, že pri tomto atribúte sa musela logika zoskupovania mierne upraviť, keďže kniha môže mať buď jedného alebo dvoch autorov a my chceme každého autora ako samostatnú podkategóriu, pre ktorú sa budú zoskupovať počty a príjmy. Ak sú všetky vstupné parametre platné, metóda načíta dva XML dokumenty obsahujúce údaje o knihách, resp. údaje o transakciách. Potom spojí tieto dva dokumenty na základe ID knihy, aby získala všetky informácie týkajúce sa každej transakcie. Metóda filtruje transakcie len na tie, ktoré sú realizované v zadanom období a majú hodnotu atribútu typ\_transackie *"predaj"*. Ďalej metóda vypočíta celkové množstvo predaných kníh a celkové príjmy pre každú podkategóriu atribútu (napr. pre atribút jazyk sú podkategórie *“sk“*, *“cz“*, *“en“*). Pre každú podkategóriu metóda vypočíta celkové množstvo predaných kusov a celkové príjmy aj pre jednotlivé tituly, ktoré patria do danej podkategórie. Tieto tituly patriace do danej podkategórie, taktiež zoskupí podľa ich ID, aby získala súhrnné údaje o príjmoch a predajoch podľa jednotlivých titulov kníh. Metóda potom zoradí agregované údaje na základe hodnôt parametrov *sortingField* a *sortingOrder*. Ak je hodnota paremetru sortingField zadaná používateľom *"nazov",* tak sa agregované údaje zoradia na základe názvu podkategórie a potom podľa názvov jednotlivých kníh pre danú podkategóriu. Ak je hodnota parametra *sortingOrder* *"ascending"*, agregované údaje sú zoradené vzostupne, v opačnom prípade sú zoradené zostupne. Ak je hodnota parametra *sortingField* zadaná ako *"hodnota"* a *optionalParameter* je *"quantity",* agregované údaje sú zoradené na základe celkového množstva predaných kníh. Ak je hodnota nepovinného parametra “*revenue*”, tak sú agregované údaje jednotlivých kategórii a kníh v týchto podkategóriách zoradené podľa celkového príjmu. Pri konci metóda vytvorí ešte súhrn štatistík pre dané hľadanie. Metóda najprv vypočíta maximálne a minimálne príjmy a predané množstvo pre jednotlivé podkategórie a jednotlivé knihy v podkategóriách. Urobí to pomocou metód Max() a Min() jazyka LINQ. Ďalej metóda nájde názvy podkategórií a jednotlivých kníh, ktoré zodpovedajú maximálnym a minimálnym hodnotám tržieb a predaného množstva.

Metóda potom vypočíta niektoré ďalšie štatistiky na základe údajov v premennej *agregatedData* vrátane celkového predaného množstva, celkových tržieb, priemerného denného predaného množstva a priemerných denných tržieb.

Metóda spojí všetky tieto údaje do jedného objektu a serializuje ich do formátu JSON pomocou knižnice Newtonsoft.Json. Potom konvertuje JSON na XML pomocou tej istej knižnice a uloží XML do diskového/serverového súboru. Nakoniec odošle JSON ako odpoveď na webovú požiadavku, ktorá spustila program.

V druhej polovici metódy je aplikovaná rovnaká logika, avšak údaje sa zoskupujú podľa autorov (teda mená autorov sú brané ako podkategória). Ak má kniha dvoch autorov, tak sa údaje zoskupujú pre každého autora samostatne s tým, že do celkových počtov a celkových príjmov sa započítavajú len údaje prvého autora. V ukážke kódu si uvedieme len polovicu metódy, ktorá nerieši špeciálny prípad, ktorý nastal pri autoroch. Celú ju môžete nájsť v elektronickej prílohe tejto práce.

Obrázok 36: Časť zdrojového kódu webovej metódy SortedDrillDownByAtributeDataBetweenTwoDatesSell zdroj: vlastné spracovanie

[WebMethod(Description = "vytvori drill down report pre udaje o poctoch a prijmoch z predaja kníh")]

public void SortedDrillDownByAtributeDataBetweenTwoDatesSell(string atribute, string startDate, string endDate, string sortingField = "", string sortingOrder = "", string optionalParameter = "")

{ // Vytvorenie poľa parametrov

string[] parameters = { atribute,startDate, endDate,

sortingField,sortingOrder,optionalParameter };

// Načítanie údajov o knihách zo súboru XML

XDocument booksData = LoadXDocument(fileBookInfo);

// Načítanie údajov o transakciách zo súboru XML

XDocument transactionsData = LoadXDocument(fileBookTransactionInfo);

DateTime start, end;

// Kontrola či dátumy sú v správnom formáte

// Kľúčové slovo "out" sa v metóde TryParse() používa na odovzdanie premenných "start"

//a "end" ako argumentov prostredníctvom referencie,

// to znamená že ich možno upraviť vnútri metódy a ich aktualizované hodnoty možno

//vrátiť volajúcemu kódu.

// Ak konverzia nie je úspešná, metóda vráti false a premenné "start" a "end" zostanú

//neinicializované.

if (!DateTime.TryParse(startDate, out start) || !DateTime.TryParse(endDate, out end))

{

Context.Response.StatusCode = 400;

Context.Response.Write("Zadajte validný formát dútumu ( napr. 'yyyy-MM-dd')");

return;

}

// Kontrola, či boli zadané všetky vstupy

if (string.IsNullOrEmpty(atribute) || startDate == null || endDate == null ||

sortingOrder == null)

{

Context.Response.StatusCode = 500;

Context.Response.Write("Jedna alebo viacero vstupov nebolo vyplnených");

return;

}

// Kontrola či dátum 'start' je pred dátumom 'end'

if (start > end)

{

Context.Response.StatusCode = 500;

Context.Response.Write("Zaciatočný dátum nesmie byť vačší ako konečný");

return;

}

// Kontrola, či bola zadaná správna hodnota 'sortingOrder'

if (sortingOrder != "ascending" && sortingOrder != "descending")

{

Context.Response.StatusCode = 500;

Context.Response.Write("Sortovanie može byť iba zostupne alebo vzostupne");

}

// Kontrola, či bola zadaná správna hodnota 'sortingField'

if (sortingField != "hodnota" && sortingField != "nazov")

{

Context.Response.StatusCode = 500;

Context.Response.Write("zoradovat sa može iba podľa názvu alebo hodnoty");

}

// Výpočet počtu dní medzi dátumami 'start' a 'end'

double days = (end - start).Days + 1;

// Ak nie je zadaný atribút 'autor' alebo 'autori'

if (atribute != "autor" && atribute != "autori")

{

// Spojenie údajov o knihách a transakciách podľa id knihy pre získanie všetkých

//informácií o každej transakcii podľa jednotlivých kníh

var aggregatedData = from book in booksData.Descendants("book")

join transaction in transactionsData.Descendants("transakcia")

on (int)book.Element("id") equals

(int)transaction.Element("id\_knihy")

// Filtrujte transakcie tak, aby sa zobrazovali len tie s

//dátumom v zadanom rozmedzí a typom "predaj"

where (DateTime)transaction.Element("datum") >= start

&& (DateTime)transaction.Element("datum") <= end

&& transaction.Element("typ\_transakcie").Value == "predaj"

// Zoskupenie transakcií podľa zvoleného atributu

group new { Book = book, Transaction = transaction } by

book.Element(atribute).Value into g

select new

{

// Uloženie názov podkategorie ako Podkategoria

Podkategoria = g.Key,

// Výpočet celkového/priemerného množstva predaných kníh a

//celkového príjmu pre každú podkategóriu

TotalQuantity = g.Sum(x =>

Math.Abs((int)x.Transaction.Element("mnozstvo"))),

TotalRevenue = g.Sum(x =>

(double)x.Transaction.Element("celkovo\_cena")),

AverageTotalQuantity = g.Sum(x =>

Math.Abs((int)x.Transaction.Element("mnozstvo"))) / days,

AverageTotalRevenue = g.Sum(x =>

(double)x.Transaction.Element("celkovo\_cena")) / days,

// pre každú podkategóriu aj zoskupenie kníh patriacich do tejto podkategórie

// Zoskupenie kníh podľa ich id, aby sme získali//agregované údaje pre knihy s rovnakým id

Books = g.GroupBy(x => x.Book.Element("id").Value)

.Select(x => new

{

Id = x.Key,

Name = x.First().Book.Element("nazov").Value,

// Výpočet celkového/priemerného množstva predaných kníh a celkový príjem pre každú knihu

//patriacu do podkategorie

TotalQuantity = x.Sum(y =>

Math.Abs((int)y.Transaction.Element("mnozstvo"))),

TotalRevenue = x.Sum(y =>

(double)y.Transaction.Element("celkovo\_cena")),

AverageTotalQuantity = x.Sum(y =>

Math.Abs((int)y.Transaction.Element("mnozstvo"))) / days,

AverageTotalRevenue = x.Sum(y =>

(double)y.Transaction.Element("celkovo\_cena")) / days

}).ToList()

};

if (!aggregatedData.Any())

{

Context.Response.StatusCode = 500;

Context.Response.Write("Neboli nájdené žiadne záznamy pre zadané kritériá");

return;

}

// Zoradenie agregovaných údajov podľa parametrov sortingField a sortingOrder

if (sortingField == "nazov")

if (sortingField == "nazov")

{// triedenie podľa názvu vzostupne

aggregatedData = sortingOrder == "ascending"

? aggregatedData.OrderBy(x => x.Podkategoria)

.ThenBy(x => x.Books.OrderBy(y => y.Name))

.Select(x => new

{

Podkategoria = x.Podkategoria,

TotalQuantity = x.TotalQuantity,

TotalRevenue = x.TotalRevenue,

AverageTotalQuantity = x.AverageTotalQuantity,

AverageTotalRevenue = x.AverageTotalRevenue,

Books = x.Books.OrderBy(y => y.Name).ToList()

})

// triedenie podľa názvu zostupne

: aggregatedData.OrderByDescending(x => x.Podkategoria)

.ThenBy(x => x.Books.OrderByDescending(y => y.Name))

.Select(x => new

{

Podkategoria = x.Podkategoria,

TotalQuantity = x.TotalQuantity,

TotalRevenue = x.TotalRevenue,

AverageTotalQuantity = x.AverageTotalQuantity,

AverageTotalRevenue = x.AverageTotalRevenue,

Books = x.Books.OrderByDescending(y => y.Name).ToList()

});

}// triedenie podľa množstva a hodnoty vzostupne alebo zostupne

else if (sortingField == "hodnota" && optionalParameter == "quantity")

{

aggregatedData = aggregatedData.OrderBy(x => x.TotalQuantity \* (sortingOrder ==

"ascending" ? 1 : -1))

.ThenBy(x => x.Podkategoria)

.Select(x => new

{

Podkategoria = x.Podkategoria,

TotalQuantity = x.TotalQuantity,

TotalRevenue = x.TotalRevenue,

AverageTotalQuantity = x.AverageTotalQuantity,

AverageTotalRevenue = x.AverageTotalRevenue,

Books = x.Books.OrderBy(y => y.TotalQuantity \* (sortingOrder == "ascending"

? 1 : -1))

.ThenBy(y => y.Name)

.ToList()

});

}// triedenie podla vynosu vzostupne alebo zostupne

else if (sortingField == "hodnota" && optionalParameter == "revenue")

{

aggregatedData = aggregatedData.OrderBy(x => x.TotalRevenue \* (sortingOrder ==

"ascending" ? 1 : -1))

.ThenBy(x => x.Podkategoria)

.Select(x => new

{

Podkategoria = x.Podkategoria,

TotalQuantity = x.TotalQuantity,

TotalRevenue = x.TotalRevenue,

AverageTotalQuantity = x.AverageTotalQuantity,

AverageTotalRevenue = x.AverageTotalRevenue,

Books = x.Books.OrderBy(y => y.TotalRevenue \* (sortingOrder == "ascending"

? 1 : -1))

.ThenBy(y => y.Name)

.ToList()

});

}// Najvyšší a najnižší celkový výnos a množstvo pre podkategórie a pre knihy

var maxRevenuePodkategoria = aggregatedData.Max(x => x.TotalRevenue);

var minRevenuePodkategoria = aggregatedData.Min(x => x.TotalRevenue);

var maxQuantityPodkategoria = aggregatedData.Max(x => x.TotalQuantity);

var minQuantityPodkategoria = aggregatedData.Min(x => x.TotalQuantity);

var maxRevenueBook = aggregatedData.SelectMany(x => x.Books).Max(x =>

x.TotalRevenue);

var minRevenueBook = aggregatedData.SelectMany(x => x.Books).Min(x =>

x.TotalRevenue);

var maxQuantityBook = aggregatedData.SelectMany(x => x.Books).Max(x =>

x.TotalQuantity);

var minQuantityBook = aggregatedData.SelectMany(x => x.Books).Min(x => x.TotalQuantity);

// premennej ktoré obsahujú mená maximálne a minimálne hodnoty pre kategórie a knihy

var namePodkategoriaMaxRevenue = aggregatedData.Where(x => x.TotalRevenue ==

maxRevenuePodkategoria).First().Podkategoria;

var namePodkategoriaMinRevenue = aggregatedData.Where(x => x.TotalRevenue ==

minRevenuePodkategoria).First().Podkategoria;

var namePodkategoriaMaxQuantity = aggregatedData.Where(x => x.TotalQuantity ==

maxQuantityPodkategoria).First().Podkategoria;

var namePodkategoriaMinQuantity = aggregatedData.Where(x => x.TotalQuantity ==

minQuantityPodkategoria).First().Podkategoria;

var nameBookMaxRevenue = aggregatedData.SelectMany(x => x.Books).Where(x =>

x.TotalRevenue == maxRevenueBook).First().Name;

var nameBookMinRevenue = aggregatedData.SelectMany(x => x.Books).Where(x =>

x.TotalRevenue == minRevenueBook).First().Name;

var nameBookMaxQuantity = aggregatedData.SelectMany(x => x.Books).Where(x =>

x.TotalQuantity == maxQuantityBook).First().Name;

var nameBookMinQuantity = aggregatedData.SelectMany(x => x.Books).Where(x =>

x.TotalQuantity == minQuantityBook).First().Name;

// Definujeme premennú, ktorá obsahuje celkové súhrnné informácie celkove/celkove priemerne min max pre kategoriu a knihu plus nazvy

var totalAggregatedData = new

{// tu su ešte dopočitane ešte celkové a celkové priemerne udaje (prve 4)

totalQuantity = aggregatedData.Sum(x => x.TotalQuantity),

totalRevenue = aggregatedData.Sum(x => x.TotalRevenue),

averageTotalDailyQuantity = aggregatedData.Sum(x => x.TotalQuantity) / days,

averageTotalDailyRevenue = aggregatedData.Sum(x => x.TotalRevenue) / days,

namePodkategoriaMaxRevenue = namePodkategoriaMaxRevenue,

maxRevenuePodkategoria = maxRevenuePodkategoria,

namePodkategoriaMinRevenue = namePodkategoriaMinRevenue,

minRevenuePodkategoria = minRevenuePodkategoria,

namePodkategoriaMaxQuantity = namePodkategoriaMaxQuantity,

maxQuantityPodkategoria = maxQuantityPodkategoria,

namePodkategoriaMinQuantity = namePodkategoriaMinQuantity,

minQuantityPodkategoria = minQuantityPodkategoria,

nameBookMaxRevenue = nameBookMaxRevenue,

maxRevenueBook = maxRevenueBook,

nameBookMinRevenue = nameBookMinRevenue,

minRevenueBook = minRevenueBook,

nameBookMaxQuantity = nameBookMaxQuantity,

maxQuantityBook = maxQuantityBook,

nameBookMinQuantity = nameBookMinQuantity,

minQuantityBook = minQuantityBook,

};

// skombinujeme objekt aggregateddata a totalaggregateddata do jedného výsledného objektu

var result = new

{

AggregatedData = aggregatedData,

TotalAggregatedData = totalAggregatedData

};

// serializacia objektu na json

var json = JsonConvert.SerializeObject(result, Formatting.Indented);

// deserializacia jsonu na xml

XmlDocument doc = JsonConvert.DeserializeXmlNode(json, "Root");

// serializacia the XmlDocument na reťazec

string xmlString = doc.OuterXml;

// zápis xml vysledku do diskového súboru

SaveWebMethodResult(xmlString, "SortedDrillDownByAtributeDataBetweenTwoDatesSell", parameters, fileAmountFilterPath);

// odoslanie jsonu v utf8 klientovi

Context.Response.BinaryWrite(System.Text.Encoding.UTF8.GetPreamble());

Context.Response.Write(json);

}

else{// ošetrenie pre prípad autorov v elektronickej prílohe tejto práce...}}

**Príklad vstupu a výstupu**

Predstavíme si ukážky dvoch vstupov a výstupov. Obidva budú pre rovnaké časové obdobie. V prvom výstupe sú údaje zoskupené podľa jazyka, v ktorom je kniha napísaná, v druhom výstupe sú údaje zoskupené podľa autorov kníh. Vidíme, že výsledné agregované hodnoty pre údaje sú rovnaké, avšak vždy z iného pohľadu, čo dáva manažmentu kníhkupectva silný analyticky nástroj, v ktorom si môžu vytvárať rôzne dynamické reporty z rôznych pohľadov.

Obrázok 37: Ukážka vstupu a výstupu č.1 pri webovej metóde SortedDrillDownByAtributeDataBetweenTwoDatesSell zdroj: vlastné spracovanie



Obrázok 38: Ukážka vstupu a výstupu č.2 pri webovej metóde SortedDrillDownByAtributeDataBetweenTwoDatesSell zdroj: vlastné spracovanie



Obrázok 39: Zdrojový kód webovej metódy CalculateFinancialIndicators zdroj: vlastné spracovanie

[WebMethod(Description = "slúži na výpočet celkového príjmu, nákladov, zisku a iných

ukazovateľov")]

public void CalculateFinancialIndicators(int year, int quarter, int month)

{

string[] parameters = { year.ToString(), quarter.ToString(), month.ToString() };

// Načítanie xml súboru do objektu XDocument

XDocument xDoc = LoadXDocument(fileBookTransactionInfo);

// Vytvorenie zoznamu transakcií na základe xml súboru

var transactions = from transaction in xDoc.Descendants("transakcia")

select new

{

Date = DateTime.Parse(transaction.Element("datum").Value),

Type = transaction.Element("typ\_transakcie").Value,

Amount = (int)transaction.Element("mnozstvo"),

Price = (double)(transaction.Element("cena\_za\_jednotku"))

};

// Ak nie je zvolený rok, nastaví sa chybový kód 500 a vráti sa chybová správa

if (year == 0)

{

Context.Response.StatusCode = 500;

Context.Response.Write("Prosím zvoľte si rok ");

return;

}

// Ak sú zvolené štvrťrok aj mesiac súčasne, nastaví sa chybový kód 500 a vráti sa

// chybová správa

if (quarter != 0 && month != 0)

{

Context.Response.StatusCode = 500;

Context.Response.Write("štvrťrok a mesiac nemožu byť zvolené súčasne ");

return;

}

// Ak je zvolený iba štvrťrok, filtrovanie sa vykoná pre transakcie z daného štvrťroku

else if (quarter != 0)

{

transactions = transactions.Where(t => t.Date.Year == year && (t.Date.Month - 1) / 3 + 1 == quarter);

}

// Ak je zvolený iba mesiac, filtrovanie sa vykoná pre transakcie z daného mesiaca

else if (month != 0)

{

transactions = transactions.Where(t => t.Date.Year == year && t.Date.Month == month);

}

// Ak nie je zvolený ani štvrťrok, ani mesiac, filtrovania sa vykoná pre transakcie z

// daného roku

else

{

transactions = transactions.Where(t => t.Date.Year == year);

}

// Výpočet celkového príjmu na základe transakcií so záporným množstvom a typom

//"predaj"

double totalRevenue = transactions.Where(t => t.Type == "predaj" && (t.Amount < 0))

.Sum(t => Math.Abs(t.Amount) \* t.Price);

// Výpočet celkových nákladov na základe transakcií s kladným množstvom a typom "nakup"

double totalCost = transactions.Where(t => t.Type == "nákup" && t.Amount > 0)

.Sum(t => t.Amount \* t.Price);

// vypočet zisku

double profit = totalRevenue - totalCost;

// celkový počet objednávok a celkový počet kusov v objednávkach

int numSellOrders = transactions.Count(t => t.Type == "predaj" && (t.Amount < 0));

int numBuyOrders = transactions.Count(t => t.Type == "nákup" && t.Amount > 0);

int totalQuantityOfBooksNakup = transactions.Where(t => t.Type == "nákup" && t.Amount >

0).Sum(t => t.Amount);

int totalQuantityOfBooksPredaj = transactions.Where(t => t.Type == "predaj" &&

(t.Amount < 0)).Sum(t => Math.Abs(t.Amount));

// vypočet netprofitmargin a ROI

double netProfitMargin = profit / totalRevenue;

double returnOnInvestment = profit / totalCost;

// zápis a odoslanie odpovede podobné ako v predošlých metódach

var result = new { totalRevenue, totalCost, profit, numSellOrders, numBuyOrders,totalQuantityOfBooksNakup,totalQuantityOfBooksPredaj, netProfitMargin, returnOnInvestment, };

var json = JsonConvert.SerializeObject(result, Formatting.Indented);

XmlDocument doc = JsonConvert.DeserializeXmlNode(json, "Root");

string xmlString = doc.OuterXml;

SaveWebMethodResult(xmlString, " CalculateRevenueCostProfit", parameters,

fileAmountFilterPath);

Context.Response.BinaryWrite(System.Text.Encoding.UTF8.GetPreamble());

Context.Response.Write(json);

}

Táto webová metóda počíta viaceré finančné ukazovatele, ako sú celkové príjmy, celkové náklady, zisk, čistá zisková marža, návratnosť investícií, počet objednávok na predaj a nákup a celkové množstvo kníh nakúpených a predaných v danom roku, štvrťroku alebo mesiaci.

Metóda prijíma tri vstupné parametre: *year, quarter, month,* na základe, ktorých metóda filtruje údaje o transakciách. Metóda najprv načíta transakcie z XML súboru do objektu *XDocument.* Potom vytvorí zoznam transakcií pomocou jazyka LINQ. Z každej transakcie si pomocou metódy *select* vyberieme štyri elementy a to *dátum*, *typ\_transakcie*, *množstvo*  a *cena*. Metóda potom na zoznam transakcií aplikuje rôzne filtre na základe vstupných parametrov.

Metóda vracia finančné údaje buď v konkrétnom roku alebo v konkrétnom štvrťroku daného roka alebo v konkrétnom mesiaci daného roka. Vždy musí byť zvolený nejaký rok a voliteľne môže byť zvolený štvrťrok alebo mesiac. Nikdy nemôžu byt zvolené všetky tri parametre súčasne.

Ďalej metóda vypočíta celkové výnosy, celkové náklady a zisk na základe filtrovaných transakcií. Celkové výnosy sa vypočítajú ako súčet absolútnych hodnôt množstva predaných kníh vynásobených ich príslušnými cenami. Celkové náklady sa vypočítajú ako súčet množstva nakúpených kníh vynásobený ich príslušnými cenami. Zisk sa vypočíta ako rozdiel medzi celkovými príjmami a celkovými nákladmi. Metóda potom na základe vyfiltrovaných transakcií vypočíta počet zákazníckych a počet dodávateľských objednávok a celkové množstvo kúpených a predaných kníh.

Nakoniec metóda vypočíta čistú ziskovú maržu a návratnosť investície. Čistá zisková marža sa vypočíta ako pomer zisku k celkovým výnosom. Návratnosť investícií sa vypočíta ako pomer zisku k celkovým nákladom.

Výsledky výpočtov sa potom serializujú do formátu JSON a vrátia sa ako odpoveď na volanie webovej metódy. Výsledky sa tiež uložia do XML súboru spolu s názvom metódy a vstupnými parametrami.

### Testovanie a nasadenie na server

Služby sme po vytvorení manuálne otestovali pomocou predpripraveného testovacieho klienta bežiaceho na IIS Express serveri, ktorého sme mohli vidieť pri ukážkach vstupov a výstupov niektorých webových metód. Taktiež sme použili komerčný produkt SOAPUI, ktorý poskytuje viac funkcií ako testovací klient vo Visual Studiu, pretože ponúka funkcie, ako je napríklad testovanie záťaže a automatizované testovanie. Okrem toho SoapUI umožňuje používateľom prispôsobiť svoje testy pomocou skriptov a podporuje integráciu s inými nástrojmi a frameworkmi, ako sú Jenkins a Selenium, čo z neho robí univerzálnejší a výkonnejší nástroj na testovanie webových služieb.

Po dostatočnom otestovaní služieb sme ich nasadili na IIS server nachádzajúci sa na našom lokálnom počítači. Pre tento účel sme museli v IIS managerovi vytvoriť novú webová stránku, ktorej sme priradili port 8050.

Obrázok 40: Odoslanie testovacej požiadavky v nástroji SOAPUI zdroj: vlastné spracovanie

## Tvorba webového klienta

Jedným z čiastkových cieľov práce bola tvorba dynamického responzívneho webového klienta, pomocou ktorého vieme údaje vrátené webovými službami zobraziť v prostredí webového prehliadača. Tvorba klienta neprebiehala až po ukončení tvorby webových služieb, ale v podstate súbežne s ich tvorbou. Aj keď tvorba klienta zabrala porovnateľné množstvo času a dĺžka zdrojového kódu je porovnateľná s tou, akú má webová služba, nebudeme si ho opisovať až do takej hĺbky ako pri webových službách. Spoľahneme sa skôr na predstavenie základných princípov, ako aj základných technológií, ktoré boli pri ňom použité a pomocou ukážok obrazoviek si priblížime jeho funkcionalitu.

### Technológie použité pri tvorbe klienta

Na tvorbu základnej štruktúry webovej stránky a jej štýlov sme využili Hypertextový značkovací jazyk (HTML) a kaskádové štýly (CSS). Na zabezpečenie dynamiky sme použili štandardne jazyk Javascript. Pri tvorbe klienta sme využili viacero frameworkov, ktoré jeho tvorbu uľahčili. Medzi tieto patria:

**Bootstrap**- populárny frontendový framework, ktorý umožňuje vývojárom rýchlo a jednoducho vytvárať responzívne webové stránky a webové aplikácie pre mobilné zariadenia. Medzi kľúčové vlastnosti Bootstrapu patrí grid systém (systém mriežky), ktorý umožňuje flexibilné a responzívne rozvrhnutie stránky, rozsiahly súbor tried CSS na štylizáciu bežných komponentov používateľského rozhrania, ako sú tlačidlá, formuláre tabuľky a rôzne pomocné moduly JavaScriptu, ktoré poskytujú ďalšie funkcie, napríklad modálne okná.

* **jQuery**- javascriptová knižnica, ktorá zjednodušuje skriptovanie na strane klienta a poskytuje funkcie AJAX, ktoré boli použité na tzv. „konzumáciu“ webových metód tak, aby sa výsledky na stránke zobrazili bez nutnosti znovu načítania stránky.
* **jQuery UI**- túto knižnicu založenú na klasickom Jquery sme využili hlavne pre zabezpečenia automatického dopĺňania vstupov formulára.
* **DataTables**- Doplnok pre jQuery sa používa na pridávanie pokročilých ovládacích prvkov interakcie do tabuliek HTML.
* **Highcharts a Chart.js-** tieto javascritové knižnice sme používali na vytváranie interaktívnych a responzívnych grafov.
* Ako verzionovací systém sme použili **Git** a cloudovú platformu **Github.**
* **Visual studio code**- editor zdrojového kódu, ktorý vďaka mnohým doplnkom uľahčuje písanie HTML, CSS a Javascriptoveho kódu.

### Prepojenie vystavených metód webovej služby s klientom

Na prepojenie vystavených webových služieb s klientom sme využili Jquery funkciu ajax(). Vo funkcii jQuery sú *url, type* a ďalšie podobné parametre vlastnosťami objektu settings, ktorý sa odovzdáva ako argument funkcii. Túto funkciu sme využili pri tzv. „konzumácii“ všetkých vystavených metód webovej služby book\_services. Jej zápis a fungovanie si demonštrujeme na jednom príklade Jquery scriptu.

Obrázok 41: Príklad „konzumácie“ webovej služby pomocou Jquery funkcie ajax zdroj: vlastné spracovanie

<script>

$(document).on('submit', "#formAg", function (event) {

event.preventDefault()

var year = $("#yearSelect").val();

var quarter = $("#quarterSelect").val();

var month = $("#monthSelect").val();

if (year == 0) {

showAlert("Prosím zvoľte rok", "alerts", "danger")

event.preventDefault()

}

else if (quarter != 0 & month != 0) {

showAlert("štvrťrok a mesiac nemožu byť zvolené súčasne",

"alerts", "danger")

event.preventDefault()

}

$.ajax({

type: "POST",

url: "http://localhost: " + appPort + "/book\_services.asmx/CalculateFinancialIndicators",

data: { year: year, quarter: quarter, month: month },

dataType: "json",

encode: true,

}).done(function (response) {

var result = response

$("#totalRevenue").text(Math.round(result.totalRevenue \* 100) / 100 + "€");

$("#totalCost").text(Math.round(result.totalCost \* 100) / 100 + "€");

$("#totalProfit").text(Math.round(result.profit \* 100) / 100 + "€");

$("#numSellOrders").text(result.numSellOrders)

$("#numBuyOrders").text(result.numBuyOrders);

$("#totalQuantityOfBooksNakup").text(result.totalQuantityOfBooksNakup + "ks")

$("#totalQuantityOfBooksPredaj").text(result.totalQuantityOfBooksPredaj + "ks");

$("#netProfitMargin").text(Math.round(result.netProfitMargin \* 10000) / 100 + "%");

$("#returnOnInvestment").text(Math.round(result.returnOnInvestment \* 10000) / 100 + "%");

})

.fail(function (jqXHR, textStatus, errorThrown) { showAlert(jqXHR.responseText, "alerts", "danger"); })

event.preventDefault()

});

</script>

Tento skript sa vykoná, keď používateľ odošle formulár s ID *"formAg"*. Funkcia najprv zabráni predvolenému správaniu pri odoslaní formulára volaním metódy event.preventDefault(). Bez použitia tejto metódy, by sa odpoveď načítala na novej stránke, čo by v našom prípade spôsobilo chybu. Potom pomocou metódy val() získame hodnoty z formulára obsiahnuté v select zoznamoch a priradíme ich do premenných *year, quarter a month*. Ak používateľ nenastaví rok alebo nastaví súčasne hodnoty pre štvrťrok a mesiac, tak sa zobrazí chybové hlásenie pomocou funkcie showAlert() a zabráni ďalšiemu odoslaniu formulára. Funkcia potom odošle ajax POST požiadavku na adresu URL, kde sa nachádza vystavená metóda webovej služby *CalculateFinancialInfikators,* pričom ako údaje odovzdá hodnoty premenných *year, quarter a month.* Očakáva sa, že odpoveď bude vo formáte JSON, ako je uvedené v parametri dataType. Ak je požiadavka úspešná, funkcia aktualizuje rôzne prvky HTML na stránke s údajmi o finančných ukazovateľoch vrátených v odpovedi. Ak požiadavka zlyhá, zobrazí chybové hlásenie pomocou funkcie *showAlert*. Nakoniec funkcia opäť zabráni normálnemu odoslaniu formulára volaním funkcie *event.preventDefault*.

Obrázok 42: spôsob získania čísla portu z aktuálnej URL adresy

var appPort = window.location.port;...

Tento príkaz získava číslo portu z aktuálnej URL adresy, na ktorej sa nachádza webová stránka.

### Oznamy pre používateľa

Jedným z čiastkových cieľov bolo, že klient má byť vždy informovaný o každom úspechu alebo chybe, a to vhodným farebným označením. Pre tieto účely sme vytvorili funkciu showAlert.

Obrázok 43: Zdrojový kód javascript funkcie showAlert zdroj: vlastné spracovanie

function showAlert(message, Id, type) {

var alertContainer = "#" + Id + " #alert\_container";

var alertText = "#" + Id + " #alert-text";

var alertClass = ".alert alert-" + type;

$(alertContainer).show();

$(alertText).html(message);

$(alertContainer).addClass(alertClass);

$(alertContainer).fadeTo(2000, 500).slideUp(500, function () {

$(alertContainer).removeClass(alertClass);

$(alertContainer).slideUp(500);

});

}

Parameter *message* je text, ktorý sa zobrazí v upozornení, zatiaľ čo parameter *Id* predstavuje ID HTML elementu, v ktorom sa upozornenie zobrazí. Parameter *type* je reťazec, ktorý určuje typ oznamu, napríklad *"success"* (úspech), *"warning"* (varovanie) alebo *"error"* (chyba). Funkcia najprv pomocou parametra *Id* skonštruuje reťazec predstavujúci selektor pre kontajner alertu a textový element alertu (premenná alertContainer a alertText). Premenná *alertClass* ukladá referenciu na CSS triedu, ktorá určuje typ oznámenia (napr. *alert alert-success*). Po zavolaní funkcie sa zobrazí kontajner s oznámením a pridá sa mu trieda s typom oznámenia. Správa sa vloží do elementu a kontajner s oznámením sa zjemní a zosunie. Po uplynutí 2 sekúnd sa trieda oznámenia odstráni a kontajner sa opäť zasunie spať hore.

Obrázok 44: Ukážka oznámení na stránke zdroj: vlastné spracovanie

### Automatické dopĺňanie do vstupov formulára

Jednou z požiadaviek na klienta bolo zabezpečenie funkcie automatického dopĺňania textu vo vybraných vstupných poliach formulára. Pre tieto účely sme vytvorili funkciu *autoCompleteInput*.

Obrázok 45: Zdrojový kód javascript funkcie autoCompleteInput zdroj: vlastné spracovanie

function autoCompleteInput(inputSelector, maxItem, sourceArray) {

var selector = inputSelector;

$(document).on("click", selector, function () {

$(this).autocomplete({

maxShowItems: maxItem,

source: sourceArray

});

});

}

Funkcia preberá tri parametre:

**inputSelektor**- reťazec predstavujúci CSS selektor pre element vstupu, na ktorý sa použije funkcia automatického dopĺňania.

**maxItem**- celé číslo predstavujúce maximálny počet položiek, ktoré sa môžu zobraziť v zozname automatického dopĺňania. Ak počet položiek v zozname prekročí hodnotu maxItem, výška zoznamu sa upraví tak, aby sa doň zmestili len položky maxItem.

**sourceArray**- pole predstavujúce zdrojové údaje pre zoznam automatického dopĺňania.

V súhrne táto funkcia pridáva funkcionalitu automatického dopĺňania do vstupného elementu určeného selektorom CSS s maximálnym počtom položiek.

Obrázok 46: Príklad automatického dopĺňania vstupného poľa formulára zdroj: vlastné spracovanie

### Ukážky obrazoviek

**Index.html**

Hlavná stránka celého klienta. Používateľ si tu pomocou tlačidiel môže vybrať, že o akú službu by mal záujem a podľa výberu je presmerovaný na prislúchajúcu stránku.



Obrázok 47: Hlavná obrazovka klienta zdroj: vlastné spracovanie

**singleBookSearcher.html**

Na tejto stránke môže používateľ hľadať informácie o knihách na základe ID, mena alebo ISBN.



Obrázok 48: Obrazovka na vyhľadanie údajov o jednej knihe zdroj: vlastné spracovanie

**booksCrud.html**

****Na tejto stránke sú v tabuľke zobrazené dáta o všetkých knihách. Po kliknutí na fotku knihy si vie požívateľ pozrieť konkrétnejšie údaje o danej knihe. Na tejto obrazovke má používateľ taktiež možnosť pridať, aktualizovať alebo vymazať záznam o knihe. Všetky formuláre sa nachádzajú v modálnych oknách.

Obrázok 49: Formulár na pridanie knihy zdroj: vlastné spracovanie

Obrázok 50: Údaje o jednej knihe v modálnom okne zdroj: vlastné spracovanie

Obrázok 51: Obrazovka s údajmi o všetkých knihách zdroj: vlastné spracovanie



**trasactionCrud.html**

****Na tejto stránke sú v tabuľke zobrazené dáta o všetkých obchodných transakciách. Používateľ tu má možnosť pridať, aktualizovať alebo vymazať záznam o transakcii. Všetky formuláre sa nachádzajú taktiež ako pri knihách v modálnych oknách.

Obrázok 52: Obrazovka s údajmi o všetkých transakciách zdroj: vlastné spracovanie

**amounts.html**

Na tejto obrazovke si používateľ vie zistiť usporiadané údaje o počtoch kníh v zadanom období a pre vybraný atribút. Výsledné dáta sú zobrazené v jednej tabuľke a v dvoch grafoch. Prvý obsahuje začiatočné a konečné počty pre jednotlivé knihy v danom období a druhý obsahuje celkové údaje ako je napr. celkový začiatočný a konečný počet.



Obrázok 53: Obrazovka s údajmi o počtoch kníh v zadanom období a podľa hodnoty vybratého atribútu zdroj: vlastné spracovanie

**sells.html a costs.html**

Tieto stránky majú rovnakú štruktúru s tým rozdielom, že jedna predstavuje jednoduchý dashboard o predajoch a druhá o nákupoch kníh. Používateľovi sú v kartách poskytnuté rôzne údaje ako je napr. celkový príjem/počet, priemerný príjem/počet, ako aj maximálny minimálny príjem/počet pre jednotlivú podkategóriu zvoleného atribútu alebo knihy. Dáta sú zobrazene v dvoch drill down grafoch, ktoré sú dynamicky tvorené na základe vstupov používateľa. Používateľ má možnosť si oba grafy zobraziť na celej obrazovke alebo uložiť ako obrázok v rôznych formátoch. Výsledky si tiež môže uložiť v csv formáte a otvoriť napr. pomocou Excelu.

Obrázok 54: Obrazovka s údajmi o predajoch kníh vo vybranom období a podľa vybraného atribútu zdroj: vlastné spracovanie





Obrázok 55: Ukážka drill down grafu zdroj: vlastné spracovanie

**financial\_indicators.html**

Na tejto stránke si používateľ vie pozrieť hospodárske výsledky kníhkupectva v danom roku, v danom štvrtkou roka alebo mesiaci daného roka.

Obrázok 56: Obrazovka zobrazujúca hospodárske výsledky vo vybranom období zdroj: vlastné spracovanie



# Diskusia

Vytvorená webová služba by spolu s klientom mohla fungovať na správu evidencie a analytiku údajov o knihách v malom kníhkupectve, ktoré by sa rozhodlo v budúcnosti expandovať a chcelo by používať sofistikovanejší nástroj ako je Excel. Hoci je Excel užitočným nástrojom na správu menšej údajovej základne, nemusí byť dostatočný pre rastúce kníhkupectvo, ktoré potrebuje spravovať zložitejšie údaje a procesy. Povedzme napríklad, že kníhkupectvo má dve fyzické pobočky v rôznych mestách. Každé z nich má svoj vlastný systém správy údajov o zásobách predajoch, nákladoch a podobne, ale manažéri chcú mať možnosť robiť analytiku nad týmito údajmi z oboch predajných miest v reálnom čase. Pomocou našej webovej služby, ktorá vystavuje metódy poskytujúce potrebné údaje, môžu tento cieľ dosiahnuť. Webová služba by sa dala napr. využiť, ak by kníhkupectvo potrebovalo zdieľať svoje údaje, ako sú počty kníh na sklade, údaje o predajoch a podobné relevantné údaje v reálnom čase napr. naprieč viacerými predajňami. Manažéri nachádzajúci sa na viacerých miestach by mohli vykonávať rôzne analýzy a bolo by zaručené, že to robia nad rovnakými údajmi. Výsledné údaje majú manažéri možnosť vidieť vo viacerých formátoch a podobách, či už ide o tabuľky alebo grafy nachádzajúce sa v klientovi alebo výstupné XML súbory, ktoré sa vytvárajú po každom zavolaní metód webovej služby s časovou pečiatkou. Údaje z niektorých grafov si manažér môže uložiť ako obrázok PDF alebo dokonca CSV súbor. To im dáva na výber celé spektrum možností, čo s údajmi môžu ďalej robiť.

Je jasné, že nami vytvorené riešenie nemá šancu konkurovať systémom veľkého kníhkupectva, na ktorom pracovali viaceré vývojárske tímy, aké sme si opísali v tretej kapitole. Mohlo by však predstavovať rozšírenie už pre existujúci systém. Webová služba by napríklad mohla fungovať ako samostatný modul, ktorý by svojou funkcionalitou rozširoval už pred tým vytvorený robustnejší systém. Samozrejme je možné, že by muselo dôjsť k niekoľkým úpravám tak, aby boli splnené rôzne požiadavky informačného systému danej firmy.

My sme napríklad webovú službu a klienta nasadili na lokálny IIS server. Firma by mohla nasadiť službu na svoj server prevádzkovaný, či už priamo vo firme alebo napr. na cloude. Taktiež mnohé firmy používajú pri prevádzke svojich informačných systémov aj databázový server, a preto by sme museli niektoré metódy upraviť tak, aby dokázali dopytovať údaje napríklad z nejakej relačnej alebo NoSQL databázy. Bolo by taktiež potrebné vytvoriť systém autentifikácie a autorizácie, ktorý by zabezpečil, aby prístup k službám a údajom mali len zodpovedné osoby. Služba by sa dala samozrejme ďalej rozširovať o nové metódy. Všetko závisí len na firme, v ktorej by webová služba bola implementovaná.

# Záver

Cieľom tejto práce bola analýza využívania XML webových služieb na sledovanie parametrov kníh v informačnom systéme elektronického kníhkupectva, ako aj porovnanie spôsobu, akým tieto parametre sledujú reálne kníhkupectvá. Jedným z hlavných cieľov bolo taktiež vytvorenie ASP .NET XML webovej služby, ktorá svojmu klientovi poskytne usporiadané informácie o parametroch kníh vo vybranom období. Splneniu tohto cieľa predchádzalo dôkladné preštudovanie literatúry o technológii webových služieb, ako aj o všetkých programátorských technikách využitých neskôr na vytvorenie našej webovej služby. Následne bola vykonaná analýza v reálnych podnikateľských subjektoch. V nej sme zistili, že webové služby využívajú hlavne väčšie kníhkupectvá. Tie ich využívajú hlavne v oblastiach, kde je potrebné zabezpečiť integráciu alebo vzájomnú komunikáciu a výmenu údajov štandardizovaným spôsobom medzi viacerými systémami alebo aplikáciami. Dozvedeli sme sa tu, že webové služby môžu zohrávať jeden z kľúčových prvkov, ktorý prispieva k automatizácii niektorých kľúčových alebo podporných procesov v podniku, čím sa zlepšujú služby poskytované zákazníkom, ako aj ziskovosť samotného podniku.

V praktickej časti práce, sme taktiež cieľ splnili a podarilo sa nám vytvoriť ASP .NET webovú službu, ktorá pomocou svojich vystavených webových metód, ale aj nevystavených newebových metód dokáže robiť základnú správu údajov elektronického kníhkupectva, ktorá je uložená v XML súboroch. Nami vytvorená webová služba obsahuje taktiež viaceré metódy, ktoré dokážu svojmu klientovi poskytnúť rôzne analytické a štatistické údaje o počtoch, predajoch, nákladoch a ziskovosti elektronického kníhkupectva. K demonštrácii funkčnosti webových služieb bol taktiež vytvorený dynamický webový klient, pomocou ktorého môže napr. manažér daného kníhkupectva zadávať rôzne vstupy a výstupy dostáva v tabuľke alebo v grafe. Nami vytvorená webová služba, by sa mohla využiť v malom kníhkupectve, ale aj ako rozšírenie už existujúceho systému väčšieho kníhkupectva.

# Zoznam použitej literatúry

1. **GARTNER.** Web Services. [online]. [cit. 22. 11. 2022]. Dostupné na internete: https://www.gartner.com/en/information-technology/glossary/web-services.

2. **TUTORIALSPOINT.** What are Web Services? [online]. [cit. 22. 11. 2022]. Dostupné na internete: https://www.tutorialspoint.com/webservices/what\_are\_web\_services.htm.

3. **JURÍK, P.** *Servisne orientovaná architektúra v procesne riadenom podniku.* Nové Zámky: Tlačiareň Merkur, 2020. 978-80-89996-06-3.

4. **IBM.** Properties of web services. [online]. 14. 4. 2021. [cit. 26. 12. 2022]. Dostupné na internete: https://www.ibm.com/docs/en/cics-tx/10.1.0?topic=overview-properties-web-services.

5. **TECHMACHINA.** The evolution of Web Services. [online]. 14. 8. 2007. [cit. 26. 12. 2022]. Dostupné na internete: http://www.techmachina.com/2007/08/evolution-of-web-services.html.

6. **SUTHERLAND, J.** Web Services: Better than CORBA or DCOM? [online]. 3. 10. 2003. [cit. 27. 12. 2022]. Dostupné na internete: http://jeffsutherland.com/2003/10/web-services-better-than-corba-or-dcom.html.

7. **RYAN, K.** The evolution of web service protokols pt1. [online]. 18. 10. 2020. [cit. 26. 12. 2022]. Dostupné na internete: https://sandigital.uk/blog/1-web-service-history/.

8. **CERAMI, E.** *Web services Essentials.* Sebastopol California: O’Reilly Media, Inc., 2002. 978-0-596-00224-4.

9. **GRAHAM, S, DAVIS, D a SIMEONOV, S.** *Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI, 2nd Edition.* Carmel: Sams Publishing, 2004. 978-0-672-32641-7.

10. **HOFFMAN, J.** SOAP And REST At Odds. [online]. 26. 6. 2017. [cit. 28. 12. 2022]. Dostupné na internete: https://thehistoryoftheweb.com/soap-rest-odds/.

11. **ALTEXSOFT.** Comparing API Architectural Styles: SOAP vs REST vs GraphQL vs RPC. [online]. 29. 5. 2020. [cit. 28. 12. 2022]. Dostupné na internete: https://www.altexsoft.com/blog/soap-vs-rest-vs-graphql-vs-rpc/.

12. **KREGER, H.** *Web Services Conceptual Architecture (WSCA 1.0).* New York: IBM Software Group, 2001.

13. **JAVATPOINT.** Architecture of Web Services. [online]. [cit. 30. 12. 2022]. .Dostupné na internete: https://www.javatpoint.com/restful-web-services-architecture-of-web-services.

14. **SOAPU.** SOAP Service Mocking Overview. [online]. [cit. 30. 12. 2022]. Dostupné na internete: https://www.soapui.org/docs/soap-mocking/service-mocking-overview/.

15. **W3TECHS.** Usage statistics of HTTP/3 for websites. [online]. 25. 1. 2023. [cit. 25. 1 2023]. Dostupné na internete: https://w3techs.com/technologies/details/ce-http3.

16. **W3TECHS.** Usage statistics of HTTP/2 for websites. [online]. 25. 1. 2023. [cit. 25. 1. 2023]. Dostupné na internete: https://w3techs.com/technologies/details/ce-http2.

17. **FIELDING, R, NOTTINGHAM, M a RESCHKE, J.** HTTP Semantics. [online] 6. 2022. [cit. 2023. 1. 2023]. Dostupné na internete: https://datatracker.ietf.org/doc/html/rfc9110. 2070-1721.

18. **MDN WEB DOCS.** An overview of HTTP. [online]. 15. 1.2023. [cit. 25. 1. 2023]. Dostupné na internete: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview.

19. **LUDIN, J a GARZA, S.** *Learning HTTP/2.* Sebastopol: O’Reilly Media, 2017. 978-1-491-96244-2.

20. **MDN WEB DOCS.** HTTP Messages. [online]. 11. 10. 2022. [cit. 25. 1. 2023]. Dostupné na internete: https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages.

21. **IBM.** HTTP responses. [online]. 3. 3. 2021. [cit. 26. 1. 2023]. Dostupné na internete: https://www.ibm.com/docs/en/cics-ts/5.2?topic=protocol-http-responses.

22. **HTTP DEV.** CONNECT. [online]. 20. 6. 2022. [cit. 27. 1. 2023]. Dostupné na internete: https://http.dev/connect.

23. **BOS, A.** What is an OPTIONS HTTP Request? [online]. 14. 10. 2021. [cit. 27. 1. 2023]. Dostupné na internete: https://aaronbos.dev/posts/http-options-introduction.

24. **W3SCHOOLS.** HTTP Request Methods. [online]. [cit. 27. 1. 2023]. Dostupné na internete: https://www.w3schools.com/tags/ref\_httpmethods.asp.

25. **WEBNOTS EDITORIAL STAFF.** List of 1xx HTTP Status Codes for Informational. [online]. 8. 12. 2019. [cit. 27. 1. 2023]. Dostupné na internete: https://www.webnots.com/1xx-http-status-codes/.

26. **WEBNOTS EDITORIAL STAFF.** List of 2xx HTTP Status Codes with Explanation. [online]. 27. 6. 2021. [cit. 27. 1. 2023]. Dostupné na internete: https://www.webnots.com/2xx-http-status-codes/.

27. **WEBSITEPULSE.** HTTP Status Codes - 3xx. [online]. 15. 3. 2020. [cit. 27. 1. 2023]. Dostupné na internete: https://www.websitepulse.com/kb/3xx\_http\_status\_codes.

28. **MARSCHAL, B.** *XML BY EXAMPLE.* Indianapolis: QUE, 2002. 0-7897-2504-5.

29. **W3SCHOOLS.** XML Elements. [online]. [cit. 28. 1. 2023]. Dostupné na internete: https://www.w3schools.com/xml/xml\_elements.asp.

30. **W3SCHOOLS.** XML Tree. [online]. [cit. 29. 1. 2023]. Dostupné na internete: https://www.w3schools.com/xml/xml\_tree.asp.

31. **TUTORIALSPOINT.** XML - Syntax. [online]. [cit. 29. 1. 2023]. Dostupné na internete: https://www.tutorialspoint.com/xml/xml\_syntax.htm.

32. **BRAY T, HOLLANDER D, LAYMAN A, et al.** Namespaces in XML 1.0 (Third Edition). [online]. 8. 12. 2009. [cit. 30. 1. 2023]. Dostupné na internete: https://www.w3.org/TR/xml-names/.

33. **W3SCHOOLS.** XML Namespaces. [online]. [cit. 30. 1. 2023]. Dostupné na internete: https://www.w3schools.com/xml/xml\_namespaces.asp.

34. **BOURHIS, P, a iní.** JSON: Data model, Query languages and. [online]. 9. 5. 2017. [cit. 30. 1. 2023]. Dostupné na internete: https://dl.acm.org/doi/pdf/10.1145/3034786.3056120.

35. **W3SCHOOLS.** JSON - Introduction. [online]. [cit. 30. 1. 2023]. Dostupné na internete: https://www.w3schools.com/js/js\_json\_intro.asp.

36. **ALNAFJAN, K.** *Behavior based Comparative analysis of XML and JSON web technologies.* 11, Riyadh: Wulfenia Journal, 2012, zv. 19. 1561-882X.

37. **ŠIMEC, A a M., Magličić.** *Comparison of JSON and XML Data Formats.* Varazdin: University of Zagreb, Faculty of organization and informatics, 2014.

38. **TUTORIALSPOINT.** WSDL Tutorial. [online.] [cit. 1. 2. 2023]. Dostupné na internete: https://www.tutorialspoint.com/wsdl/index.htm.

39. **VIRENDER R, ANSHU S.** *API Features Individualizing of Web Services: REST and SOAP.* 8, International Journal of Innovative Technology and Exploring Engineering (IJITEE), 2019. 2278-3075.

40. **GILLIS, A.** https://www.techtarget.com/searchapparchitecture/definition/SOAP-Simple-Object-Access-Protocol. [online] 22. 6 2022. [cit. 1. 2. 2023]. Dostupné na internete: https://www.techtarget.com/searchapparchitecture/definition/SOAP-Simple-Object-Access-Protocol.

41. **IBM.** Literal vs. Encoded, RPC- vs. Document-Style. [online]. 28. 2. 2021. [cit. 1. 2 2023]. Dostupné na internete: https://www.ibm.com/docs/en/zvse/6.2?topic=SSB27H\_6.2.0/fa2ws\_ovw\_soap\_syntax\_lit.htm.

42. **W3SCHOOLS.** XML Soap. [online]. [cit. 1. 2. 2023]. Dostupné na internete: https://www.w3schools.com/xml/xml\_soap.asp.

43. **MUMBAUKAR, S a PADIYA, P.** *Web Services Based On SOAP and REST Principles.* 3, International Journal of Scientific and Research Publications, 2013, zv. V. 2250-3153.

44. **HALILI, E a RAMADANI, F.** *Web Services: A Comparison of Soap and Rest Services.* 3, Tetova: Canadian Center of Science and Education, 2018, zv. 12. 1913-1844.

45. **WAGH, R a THOOL, K.** *A Comparative study of SOAP vs REST web services provisioning techniques for mobile host.* 5, Pune: Journal of Information Engineering and Applications, 2012, zv. 2. 2225-0506.

46. **W3TECHS.** Usage statistics of HTTP/2 for websites. [online]. 25. 1. 2023. [cit. 25. 1 2023]. Dostupné na internete: https://w3techs.com/technologies/details/ce-http2.

47. **PANZIERA, L.** Service Matchmaking:. [online]. 2013. [cit. 1. 2. 2021]. Dostupné na internete: https://www.researchgate.net/figure/Comparison-between-WSDL-11-and-WSDL-20-semantics\_fig2\_255963452.

48. **FOWLER, S, HAMEDESER, K a PETERSON, A.** *An Empirical Evaluation of Web System Access for Smartphone Clients.* Norrkoping: Journal of Networks, 2012. 1796-2056.

**Zoznam príloh**

**Príloha 1- Celý zdrojový kód webovej služby, webového klienta a XML dátovej základne zabalený v rar súbore “knihy\_jankech”**

**Príloha 2- Postup spustenia webovej služby a klienta na IIS Express serveri z prostredia Visual Studia (rýchle spustenie)**

**Príloha 3- Postup nasadenia webovej služby a klienta na IIS server**