Edukativni primer generisanja i obrade podataka uz alate dostupne u .NET 5, u domenu Metrologije

Ivan Gutai, Member, IEEE, Prof. dr Platon Sovilj, Member, IEEE, Marina Subotin, Member, IEEE, Đorđe Novaković, Member, IEEE, Nemanja Gazivoda, Member, IEEE, Bojan Vujičić, Member, IEEE

***Apstrakt*— Baze podataka predstavljaju oslonac kompletne aplikacije, a kreiranje arhitekture i izbor adekvatnih tehnologija iz godine u godinu postaje kompleksnije. Ovaj rad je kreiran kao praktično uputstvo i oslanja se na najnoviju iteraciju popularnog programskog jezika C# 9 i na relacionu bazu podataka, Microsoft-ov SQL Server, skraćeno MSSQL. Entity Framework Core je tehnologija koja značajno olakšava kreiranje i rad sa bazom podataka. Primenjen je eng. code first pristup i iz C# koda je kreirana baza podataka sa dve povezane tabele i inicijalno je popunjena tabela koja predstavlja šifarnik. Nakon kreiranja tabela, iz SQL Server Management Studio (SSMS) su kreirane dve skripte koje omogućavaju kreiranje tabela, bez upotrebe C# koda, tj. napisane su kao Structured Query Language (SQL) skripte. Kreirana je C# aplikacija koja generiše milion zapisa, koji zauzimaju 100 MB u relacionoj bazi i 152 MB kada se serijalizuju u JSON fajl uz pomoć Json.NET-a. Dat je primer korišćenja Xunit alata za testiranje. Dati su primeri proširivanja LINQ (Language-Integrated Query) funkcija Median i Mean. Prikazana su dva načina za očitavanje vrednosti iz baze, uz pomoć LINQ-a i uz pomoć SQL-a.**

***Ključne reči*— RDBMS; T-SQL; Entity Framework Core; LINQ; TDD; XUnit; Json.NET; C# 9; .NET 5; Metrologija;**

I. Uvod

Izabrana tehnologija je .NET 5, koja daje naprednije mogućnosti za pisanje višeplatformskih aplikacija. Zbog izuzetne širine, preporučljivo je prvo pročitati [1] i [2], u kojima su detaljno opisane novine koje su unete u programski jezik C# krajem 2020. godine, ali i mnogi osnovni algoritmi. U ovom radu je akcenat stavljen na upotrebu tehnologija za projektovanje i razvoj sistema koji nam omogućava da pisanjem SQL upita ili programski dođemo do rezultata, da ih filtriramo i na kraju sortiramo. Istovremena upotreba LINQ-a i SQL-a za manipulaciju podacima iz baze, predstavlja na neki način verifikaciju LINQ upita.

Ivan Gutai – Fakultet tehničkih nauka, Novi Sad, Srbija (e-mail: gutai@uns.ac.rs).
Platon Sovilj – Fakultet tehničkih nauka, Novi Sad, Srbija (e-mail: platon@uns.ac.rs).
Marina Subotin – Fakultet tehničkih nauka, Novi Sad, Srbija (e-mail: marina.bulat@uns.ac.rs).
Đorđe Novaković – Fakultet tehničkih nauka, Novi Sad, Srbija (e-mail: djordjenovakovic@uns.ac.rs).
Nemanja Gazivoda – Fakultet tehničkih nauka, Novi Sad, Srbija (e-mail: nemanjagazivoda@uns.ac.rs).
Bojan Vujičić – Fakultet tehničkih nauka, Novi Sad, Srbija (e-mail: bojanvuj@uns.ac.rs).

II. Izbor integrisanog razvojnog okruženja

Za razvoj .NET aplikacija su u startu potrebni Visual Studio IDE (Integrated Development Environment) [3] i SDK (Software Development Kit) [4]. Visual Studio Community 2019 je alat, čije će mogućnosti kasnije biti proširivane sa paketima iz NuGet Package Manager-a kao što su: EntityFrameworkCore, EntityFrameworkCore Design, EntityFrameworkCore SqlServer, EntityFrameworkCore Tools, NET.Test.Sdk, NETCore TestHost, xunit, xunit.runner.console, xunit.runner.visualstudio i Newtonsoft.Json. Navedeni alat je besplatan, a samo zahteva logovanje korisnika sa svojim Hotmail nalogom u Visual Studio.

III. .NET 5

Nekoliko godina Microsoft potencira tzv. cross platform razvoj i svake godine nam daje alate, u okviru . NET Core-a, koji to olakšavaju. Naravno, .NET pruža i dalje razvoj aplikacija koje će raditi isključivo na Windows operativnom sistemu, kao što su UWP (Universal Windows Platform) aplikacije, ali omogućava i pravljenje servisa, web ili desktop aplikacija, koje će biti dostupne na različitim platformama. Razlika između .NET Framework i .NET Core je što je inicijalno .NET Core mnogo manji i mi sami biramo koju komponentu framework-a želimo da koristimo u aplikaciji koju pravimo. Da ne bi bilo zabune, definisan je .NET Standard, koji npr. u verziji 2.0 označava koje zajedničke funkcionalnosti imaju .NET Core 2.0 i klasični .NET Framework 4.6.1. [5]. Takođe, .NET Core je izlazio u verzijama 1, 2, 3, pa je 4 preskočena, a aktuelna verzija 5, se označava samo kao .NET 5 (bez Core).

IV. Izbor baze podataka

Dva osnovna mesta za skladištenje podataka su RDBMS (Relational Database Management System) i NoSQL baze podataka. Nekoliko popularnijih RDBMS su: Microsoft SQL Server, PostgreSQL, MySQL i SQLite. Nekoliko popularnijih NoSQL su: Microsoft Azure Cosmos DB, Redis, MongoDB i Apache Cassandra. Ukoliko neko želi da pristupa podacima direktno iz baze, logičan izbor je pisanje SQL upita za RDBMS i pisanje GraphQL upita za NoSQL. U daljem tekstu RDBMS je označen kao relaciona baza, a NoSQL kao nerelaciona baza. Priča o tome koji tip baze podataka je bolji, podseća na priču o tome koji je programski jezik bolji. Jednostavno, ta odluka je na osobi koja projektuje sistem, a u ovom radu je izabrana relaciona baza Microsoft

SQL Server 2019 Express [6], a direktan pristup se vrši preko SSMS (SQL Server Management Studio) [7].

V. KREIRANJE RELACIONE BAZE PODATAKA

U nazivu "EDUG\GI_JOE", "EDUG" je naziv servera, a "GI_JOE" je naziv instance. Baza podataka ima naziv "Expo2021". A u njoj se nalaze 3 tabele: BME280Results, Locations i __EFMigrationsHistory, koja služi za praćenje izmena, a "EF" je skraćeno od "Entity Framework". Tabela Locations ima kolone LocationID i LocationName i stavke 1 i 2 za Office i Balcony, respektivno. Mala i praktična optimizacija, da se vrednosti koje se ponavljaju ne bi iznova upisivale u BME280Results tabelu. BME280ResultID, Timestamp, Temperature, RelativeHumidity, Pressure, LocationID predstavljaju nazive kolona u BME280Results tabeli. Ključevi u tabelama su sledeći:

- ✓ BME280Results tabela, PK_BME280Results i FK_BME280Results_Locations_LocationID,
- ✓ Locations tabela, PK_Locations.

Ukratko, tabela BME280Results ima sekundarni ključ u tabeli Locations, a vezani su sa LocationID-jem. Primarni ključevi u navedenim tabelama su njihove ID kolone. PK i FK su skraćenice od "primary" i "secondary" key, respektivno.

Prvi način kreiranja tabela je bio preko Entity Framework-a. Tačnije, napiše se C# kod (code first pristup) i u konzoli (Package Manager Console) upišemo komande: "Add-Migration" sa nazivom npr. "Initial" i "Update-Database". Za razvoj je korišćen konekcioni string: "Data Source=EDUG\GI_JOE;Initial Catalog=Expo2021;Integrated Security=true;". Drugi način je da se sve to uradi direktno iz SQL-a, što zahteva znatno više vremena. Treći način je da izaberemo u koju bazu ćemo dodavati tabele, a zatim da pokrenemo SQL skripte sa slike 1 i slike 2.

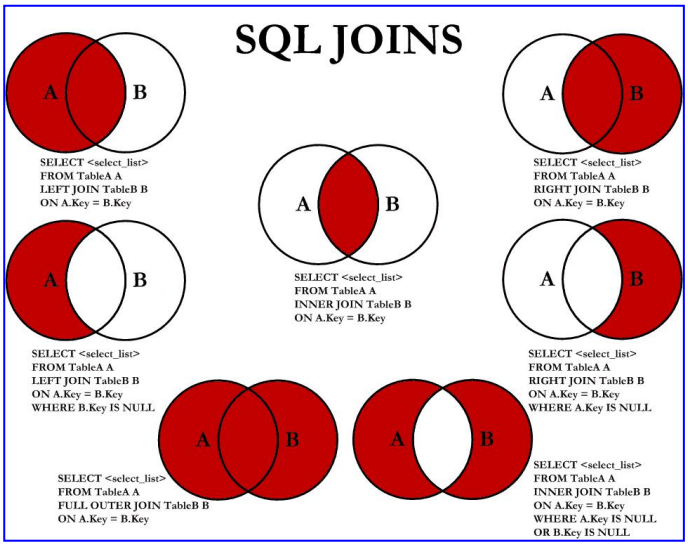
```
USE [Expo2021]
GO

/****** Object:  Table [dbo].[Locations]
Script Date: 26.3.2021. 13:16:24 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Locations](
    [LocationID] [int] IDENTITY(1,1) NOT NULL,
    [LocationName] [nvarchar](max) NULL,
 CONSTRAINT [PK_Locations] PRIMARY KEY CLUSTERED
(
    [LocationID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
```

Sl. 1. SQL skripta koja omogućava kreiranje tabele Locations, koja je povezana sa tabelom BME280Results.

```
USE [Expo2021]
GO

/****** Object:  Table [dbo].[BME280Results]
Script Date: 26.3.2021. 13:16:43 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[BME280Results](
    [BME280ResultID] [int] IDENTITY(1,1) NOT NULL,
    [Timestamp] [datetime2](7) NOT NULL,
    [Temperature] [money] NOT NULL,
    [RelativeHumidity] [money] NOT NULL,
    [Pressure] [money] NOT NULL,
    [LocationID] [int] NOT NULL,
 CONSTRAINT [PK_BME280Results] PRIMARY KEY CLUSTERED
(
    [BME280ResultID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[BME280Results]
ADD  DEFAULT (getdate()) FOR [Timestamp]
GO

ALTER TABLE [dbo].[BME280Results]  WITH CHECK
ADD  CONSTRAINT [FK_BME280Results_Locations_LocationID]
FOREIGN KEY([LocationID])
REFERENCES [dbo].[Locations] ([LocationID])
ON DELETE CASCADE
GO

ALTER TABLE [dbo].[BME280Results]
CHECK CONSTRAINT [FK_BME280Results_Locations_LocationID]
GO
```

Sl. 2. SQL skripta koja omogućava kreiranje tabele BME280Results, koja je povezana sa tabelom Locations.

Bitan je redosled kojim se navedene skripte pozivaju, zbog međusobne zavisnosti ključeva. Interesantna je i opcija kreiranje View-a u SQL-u, koji na prvi pogled liči na tabelu, a u stvari prikazuje sadržaj iz jedne ili više tabela, u zavisnosti kako to definišemo. Praktičan podsetnik koji detaljnije objašnjava pisanje upita za tzv. spajanje tabela [8] je prikazan na slici 3.



Sl. 3. Vizuelni prikaz razlika između različitih JOIN funkcija.

VI. ENTITY FRAMEWORK CORE TEHNOLOGIJA

Ko se opredeli za .NET tehnologiju, jedan od benefita je Entity framework Core tehnologija, koja omogućava da se upotrebom C# manipuliše bazom podataka, od kreiranja, preko menjanja kolona, do rada sa sadržajem. Na slici 4 je

prikazan C# kod koji definiše imena tabela, nazive i tipove kolona koji su kreirani u bazi, a "[Column(TypeName = "money")]" predstavlja jedan od atributa anotacije u Entity Framework Core-u.

```
public class Location
{
    [Key]
    public int LocationID { get; set; }
    public string LocationName { get; set; }
    public virtual ICollection<BME280Result> BME280Results { get; set; }
    public Location()
    {
        this.BME280Results = new List<BME280Result>();
    }
    public class BME280Result
    {
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public int? BME280ResultID { get; set; }
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public DateTime Timestamp { get; set; }
        [Column(TypeName = "money")]
        public decimal Temperature { get; set; }
        [Column(TypeName = "money")]
        public decimal RelativeHumidity { get; set; }
        [Column(TypeName = "money")]
        public decimal Pressure { get; set; }
        public int LocationID { get; set; }
        public virtual Location Location { get; set; }
    }
}
```

Sl. 4. C# kod koji definiše imena tabela, nazive i tipove kolona koji su kreirani u bazi.

VII. Generisanje i učitavanje podataka upotrebom C# programskog jezika

Na slikama 5 i 6 su prikazane funkcije koje omogućavaju manipulaciju nad podacima u bazi.

```
public void GenerateNItems(int NoOfItems)
{
    using (var db = new DatabaseContext())
    {
        for (int i = 1; i < NoOfItems + 1; i++)
        {
            var newEntry = new BME280Result();
            //newEntry.BME280ResultID = i;
            newEntry.Temperature = GetRandomNumberInRange(23, 32);
            newEntry.RelativeHumidity = GetRandomNumberInRange(40, 60);
            newEntry.Pressure = GetRandomNumberInRange(1010, 1030);
            newEntry.LocationID = (int)GetRandomNumberInRange(1, 3);
            db.BME280Results.Add(newEntry);

            if (i % 10000 == 0)
            {
                var count = db.SaveChanges();
                Console.WriteLine("{0} records saved to database", count);
            }
            else if (i == NoOfItems)
            {
                var count = db.SaveChanges();
                Console.WriteLine("{0} records saved to database", count);

            }
            //count = db.SaveChanges();
        }
        //Console.WriteLine("{0} records saved to database", count);
    }
}
```

Sl. 5. C# kod koji predstavlja funkciju za kreiranje n stavki u bazi.

```
public void LoadNItems(int NoOfItems)
{
    using (var db = new DatabaseContext())
    {
        Console.WriteLine("{0,16} | {1,-27} | {2,13:N2} | {3,18:N2}" +
            " | {4,10:N2} | {5,14:N2} |", "[BME280ResultID]",
            "[Timestamp]", "[Temperature]", "[RelativeHumidity]",
            "[Pressure]", "[LocationName]");

        var data = db.BME280Results.Join(
        inner: db.Locations,
        outerKeySelector: bm => bm.LocationID,
        innerKeySelector: lo => lo.LocationID,
        resultSelector: (b, l) =>
        new { b.BME280ResultID, b.Timestamp, b.Temperature,
            b.RelativeHumidity, b.Pressure, l.LocationName }).Take(NoOfItems);

        foreach (var item in data)
        {
            Console.WriteLine("{0,16} | {1,27:dd.MM.yyyy. H:mm:ss zzz}" +
                " | {2,-13:N2} | {3,-18:N2} | {4,-10:N2} | {5,14} |"
                , item.BME280ResultID, item.Timestamp, item.Temperature,
                item.RelativeHumidity, item.Pressure, item.LocationName);
        }

    }
}
```

Sl. 6. C# kod koji predstavlja funkciju za učitavanje n stavki iz baze uz upotrebu LINQ-a.

VIII. Osnovni SQL upiti

Na slikama 7 i 8 su prikazani SQL upiti i rezultati, koji omogućavaju sledeće:

- ✓ Selektovanje i kombinovanje rezultata iz dve tabele, od kojih je jedna šifarnik.
- ✓ Selektovanje i grupisanje rezultata u odnosu na lokacije sa kojih su preuzeti podaci.
- ✓ Prikaz modusa iz određene kolone, npr. RelativeHumidity.
- ✓ Funkcija TRUNCATE koju treba koristiti isključivo prilikom testiranja i razvoja. Ova funkcija momentalno briše sve stavke i brojač ID-a vraća na inicijalnu vrednost. Npr. Ako se obrišu sve stavke na ovaj način, prvi sledeći ID koji će biti dodeljen je podrazumevani, u ovom slučaju 1.
- ✓ Funkcija DELETE briše stavke po određenom kriterijumu, a njenom upotrebom se ne gubi zapis o ID-ju stavki. Npr. Ako se obrišu sve stavke na ovaj način, prvi sledeći ID koji će biti dodeljen je 1000001.

```
SELECT TOP (2) B.BME280ResultID, B.Timestamp,
B.RelativeHumidity, L.LocationName
FROM [Expo2021].[dbo].[BME280Results] AS B
INNER JOIN [Expo2021].[dbo].[Locations] AS L
ON B.LocationID = L.LocationID
ORDER BY BME280ResultID DESC

SELECT L.LocationName, COUNT(*) AS Items
FROM [Expo2021].[dbo].[BME280Results] AS B
INNER JOIN [Expo2021].[dbo].[Locations] AS L
ON B.LocationID = L.LocationID
GROUP BY L.LocationName;

SELECT TOP (2) B.RelativeHumidity, COUNT(*) AS NoOfItems
FROM [Expo2021].[dbo].[BME280Results] AS B
INNER JOIN [Expo2021].[dbo].[Locations] AS L
ON B.LocationID = L.LocationID
GROUP BY B.RelativeHumidity
ORDER BY NoOfItems DESC

-- TRUNCATE TABLE [Expo2021].[dbo].[BME280Results]

/*
    DELETE FROM [Expo2021].[dbo].[BME280Results]
    WHERE BME280ResultID > 1000000
*/
```

Sl. 7. SQL upiti koji omogućuju: izlistavanje rezultata, grupisanje rezultata po lokacijama, prikaz modusa za izabranu veličinu i brisanje podataka.

100 %

Results | Messages

	BME280ResultID	Timestamp	RelativeHumidity	LocationName
1	1000000	2021-03-26 14:10...	55,4484	Office
2	999999	2021-03-26 14:10...	44,8757	Office

	LocationName	Items
1	Balcony	499152
2	Office	500848

	RelativeHumidity	NoOfItems
1	55,9208	20
2	58,8922	17

Sl. 8. Rezultati SQL upita koji prikazuju TOP n stavki, agregaciju podataka sa dve lokacije i modusa.

IX. Programiranje vođeno testom

TDD (Test driven development) i Unit testovi su postali deo žargona u IT industriji. Najprostiji opis je pisanje koda koji testira kod i daje rezultate o njegovoj ispravnosti. Zasniva se na "AAA principu", tj. "Arrange", "Act" i "Assert", kako god se test framework zvao, a u ovom slučaju je to xUnit [9]. U delu "Arrange" pišemo pretpostavku, u "Act" testiramo funkciju, a u zavisnosti od rezultata, u "Assert"-u je određeno da li je test uspešan ili ne. Interesantno je da ukoliko test "padne", opisano je zbog čega se to desilo. TDD je praktičan na velikim projektima i olakšava regresione testove, zato što automatski proverava da li je nova funkcionalnost narušila neku staru. Na slici 9 je prikazan kod jednog testa i rezultati svih testova u Test explorer-u.

```
67          [Fact]
68          public void CalculateModeForRelativeHumidity()
69          {
70              // arrange
71              decimal expected = 55.9208M;
72              // act
73              decimal? actual = Mode.Mode_RelativeHumidity();
74              // assert
75              Assert.Equal(expected, actual);
76          }
```

109 % No issues found

Test Explorer

Test	Duration	Traits	Error Message
GenerateAndProcess (3)	21,6 sec		
GenerateAndProcess (3)	21,6 sec		
CustomUnitTests (3)	21,6 sec		
CalculateModeForPressure	8,1 sec		
CalculateModeForRelativeHumi...	7,1 sec		
CalculateModeForTemperature	6,4 sec		Assert.Equal() Failure Expected: 10 Actual: 29,4203

Sl. 9. C# kod jednog testa i rezultati svih testova u Test explorer-u.

Testove treba pisati tako da testiraju stvarne funkcionalnosti, a ne da povećavamo broj testova koji uspešno prolaze. U xUnit-u test je označen sa "[Fact]".

X. Proširivanje ugrađenih LINQ funkcija

Osnovna ideja matematičke statistike jeste da se istraživanja sprovedu na uzorku i da se na osnovu njih donesu zaključci koji se proširuju na populaciju. Statistika koju poseduje LINQ je srednja vrednost (average), a statistike koje su dodate jesu medijana (median) i modus (mean), slika 10.

```
namespace System.Linq
{
    public static class MyLinqExtensions
    {
        // this is a chainable LINQ extension method
        public static IEnumerable<T> ProcessSequence<T>(this IEnumerable<T> sequence)
        {
            // you could do some processing here
            return sequence;
        }
        integers
        public static decimal? Median(this IEnumerable<decimal?> sequence)
        {
            int numberCount = sequence.Count();
            int halfIndex = sequence.Count() / 2;
            var sortedNumbers = sequence.OrderBy(n => n);
            decimal median;
            if ((numberCount % 2) == 0)
            {
                return median = Convert.ToDecimal(((sortedNumbers.ElementAt(halfIndex)
                    + sortedNumbers.ElementAt((halfIndex - 1))) / 2));
            }
            else
            {
                return median = Convert.ToDecimal(sortedNumbers.ElementAt(halfIndex));
            }
        }

        public static decimal? Median<T>(this IEnumerable<T> sequence, Func<T, decimal?> selector)
        {
            return sequence.Select(selector).Median();
        }

        public static decimal? Mode(this IEnumerable<decimal?> sequence)
        {
            var mode = sequence.GroupBy(n => n)
                .OrderByDescending(g => g.Count()).Select(g => g.Key).FirstOrDefault();
            return mode;
        }

        public static decimal? Mode<T>(this IEnumerable<T> sequence, Func<T, decimal?> selector)
        {
            return sequence.Select(selector).Mode();
        }
    }
}
```

Sl. 10. C# kod koji služi za proširivanje funkcija medijane i modusa.

Na slici 11 su prikazane srednje vrednosti, medijane i modusi za 1000000 zapisanih vrednosti, koje sadrže informacije o temperaturi, relativnoj vlažnosti vazduha i pritisku.

```
Mean Temperature: 27,5015
Mean RelativeHumidity: 50,0046
Mean Pressure: 1.020,0059
Median Temperature: 27,5001
Median RelativeHumidity: 50,0038
Median Pressure: 1.020,0053
Mode Temperature: 29,4203
Mode RelativeHumidity: 55,9208
Mode Pressure: 1.018,8240
```

Sl. 11. Rezultati LINQ upita za Average, Mean i Median

Poređenjem rezultata sa slika 8 i 11 se može primetiti da je i LINQ upit ispravno napisan i da rezultati odgovaraju rezultatima dobijenim preko SQL upita.

XI. Primer serijalizacije podataka

Svaki put kada se spominju podaci, na backend-u se često priča o: boxing, unboxing, serialization i deserialization. Prva dva se odnose na objekat, a druga dva na "spuštanje" ili uzimanje objekta iz npr. fajla. Na slikama 12 i 13 je prikazana funkcija koja omogućava serijalizaciju podataka u zadatom obliku, a zatim su prikazani rezultati sa dve stavke, pošto nije baš jednostavno ni na modernom PC-ju otvoriti json fajl od 152 MB, sa svih 1000000 zapisa.

```
public IQueryable GenerateJSON(int NoOfItems)
{
    using (var db = new DatabaseContext())
    {
        var data = db.BME280Results.Join(
        inner: db.Locations,
        outerKeySelector: bm => bm.LocationID,
        innerKeySelector: lo => lo.LocationID,
        resultSelector: (b, l) =>
        new { b.BME280ResultID, b.Timestamp, b.Temperature, b.RelativeHumidity,
            b.Pressure, l.LocationName }).Take(NoOfItems).OrderByDescending(b => b.BME280ResultID);

        var results = data.ToList();

        JObject json =
            new JObject(
                new JProperty("id", "MillenialDIY2020LE"),
                new JProperty("BME280",
                    new JArray(
                            from p in results
                            orderby p.Timestamp
                            select new JObject(
                            new JProperty("BME280ResultID", p.BME280ResultID),
                            new JProperty("Timestamp", p.Timestamp),
                            new JProperty("Temperature", p.Temperature),
                            new JProperty("RelativeHumidity", p.RelativeHumidity),
                            new JProperty("Location", p.LocationName),
                            new JProperty("Pressure", p.Pressure)))));

        File.WriteAllText(@"c:\results.json", JsonConvert.SerializeObject(json));

        // serialize JSON directly to a file
        using (StreamWriter file = File.CreateText(@"c:\results.json"))
        {
            JsonSerializer serializer = new JsonSerializer();
            serializer.Serialize(file, json);
        }

        return data;
    }
}
```

Sl. 12. C# kod koji omogućava da se pomoću LINQ upita izvezu podaci za Average, Mean i Median.


```
{
    "id": "MillenialDIY2020LE",
    "BME280": [
        {
            "BME280ResultID": 1000001,
            "Timestamp": "2021-03-30T10:41:23.0433333",
            "Temperature": 28.9234,
            "RelativeHumidity": 57.7886,
            "Location": "Office",
            "Pressure": 1020.5552
        },
        {
            "BME280ResultID": 1000002,
            "Timestamp": "2021-03-30T10:41:23.0833333",
            "Temperature": 27.1621,
            "RelativeHumidity": 47.7860,
            "Location": "Office",
            "Pressure": 1019.2326
        }
    ]
}
```


Sl. 13. Sadržaj json fajla.

ZAHVALNICA

Rad su podržali Centar za metrologiju, Fakulteta tehničkih nauka i projekti KALCEA "Knowledge Triangle for a Low Carbon Economy" 618109-EPP-1-2020-1-EL-EPPKA2- CBHE-JP i "Razvoj naučno-stručnih metoda u oblasti metrologije, industrijsko-tehničkih merenja u digitalnom konceptu Industrije 4.0, biomedicinskih mernih sistema i kognitivnih neuronauka primenom napredne metodologije i digitalne tehnologije". Hvala Microsoft-u što na godišnjem nivou daje nove alate koji olakšavaju rad programerima.

ZAKLJUČAK

Iako je programiranje podeljeno na backend i frontend, tzv. "fullstack development" omogućava korišćenje najboljih stvari iz oba domena, a to su pisanje algoritama i optimizovanje, kao i smišljanje korisničkog interfejsa. Dobre stvari koje postoje u programiranju ne nestaju, već jednostavno menjaju ime. Npr. Microsoft-ova tehnologija Silverlight i WPF (Windows Presentation Foundation) aplikacije jesu sad stvar prošlosti, ali se XAML (Extensible Application Markup Language), koji je bio sastavna komponenta, jednostavno nastavio koristiti i dalje. Sada se XAML koristi u Xamarin-u, koji omogućava pisanje aplikacija namenjenih za Android operativni sistem. Sada je aktuelna verzija C# 9, a interesantan zapis brojeva je omogućen od verzije 8, npr. da se milion iskaže kao 1_000_000. Još jedan pokazatelj da programski jezici postaju napredniji, je dodavanje podrazumevanog metoda u interfejs, što je u ranijim verzijama C# bilo nezamislivo. U vremenu kada se pojavljuje mnogo novih tehnologija, ko savlada bilo koji "fullstack" set tehnologija, dobija mogućnost efektivnijeg prilagođavanja na novine koje se dešavaju.

LITERATURA

[1] C# 9 and .NET 5 – Modern Cross-Platform Development: Build intelligent apps, websites, and services with Blazor, ASP.NET Core, and Entity Framework Core using Visual Studio Code, 5th Edition ISBN-13 : 978-1800568105

[2] Fundamentals of Computer Programming with C#: Programming Principles, Object-Oriented Programming, Data Structures (free programming books) ISBN-13 : 978-9544007737

[3] https://visualstudio.microsoft.com/

[4] https://dotnet.microsoft.com/download/visual-studio-sdks

[5] https://docs.microsoft.com/en-us/dotnet/standard/net-standard

[6] https://www.microsoft.com/en-us/sql-server/sql-server-downloads

[7] https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15

[8] https://clmoffatt.com/visual-sql-joins/

[9] https://xunit.net/

ABSTRACT

Databases represent the backbone of entire applications and creating architecture and choice of adequate technologies annually get more complex. This paper is created as a user manual and uses the newest iteration of popular programming language C# 9 and relational database Microsoft's SQL Server abbreviated MSSQL. Entity Framework Core is a technology that eases the creation and usage of a database. The code first approach is applied, and a database is created from C# code. It has two interconnected tables, and one has initial data and serves as a codebook. After creating tables, from SQL Server Management Studio (SSMS) two scripts are created, without the usage of C#, instead, they are written as Structured Query Language (SQL) scripts. C# application that generates one million records. This takes 100 MB in a relational database and 152 MB when serialized in a JSON file using Json.NET. An example with testing tool Xunit is described. LINQ (Language-Integrated Query) functions Mean and Median are extended. Two ways for getting data from the database are described, using LINQ and using SQL.

AN EDUCATIONAL APPROACH TO GENERATING AND ANALYSING DATA WITH TOOLS AVAILABLE IN .NET 5, IN METROLOGY DOMAIN

Ivan Gutai, Platon Sovilj, Marina Subotin, Đorđe Novaković, Nemanja Gazivoda, Bojan Vujičić