SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

.NET ENTITY FRAMEWORK VS NHIBERNATE

SEMINÁRNA PRÁCA

Palo Litauszki,
Andrej Urbanek,
Lívia Cigániková,
Adam Trebichalský

2022

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

.NET ENTITY FRAMEWORK VS NHIBERNATE

SEMINÁRNA PRÁCA

Študijný program:	Aplikovaná informatika
Predmet:	I-ASOS – Architektúra softvérových systémov
Prednášajúci:	Kossaczký Igor, RNDr., CSc.
Cvičiaci:	Ing. Stanislav Marochok

Palo Litauszki,
Andrej Urbanek,
Lívia Cigániková,
Adam Trebichalský

Bratislava 2022

Obsah

1 Zadanie **1**

2 Teoretická časť **3**

2.1 .NET . . . 3

2.1.1 .NET Framework . . . 4

2.1.2 .NET Core . . . 7

2.1.3 Mono / Xamarin . . . 7

2.2 Čo je Object–relational Mapping (ORM)? . . . 8

2.3 .NET Entity Framework (EF) . . . 9

2.3.1 Výhody EF . . . 10

2.3.2 Nevýhody EF . . . 10

2.4 NHibernate . . . 10

2.4.1 Nevýhody . . . 11

2.4.2 Výhody . . . 11

2.5 Vlastnosti Entity Framework a NHibernate . . . 11

2.5.1 LINQ . . . 12

2.5.2 Vlastné implementácie Entity SQL a HQL . . . 12

2.5.3 Lazy fetching . . . 13

2.5.4 Typy ORM mapovania . . . 13

2.5.5 Databázová podpora . . . 15

2.5.6 Dokumentácia . . . 15

3 Zhodnotenie .NET EF vs NHibernate **16**

3.1 EF . . . 16

3.2 NHibernate . . . 16

4 Praktická časť **17**

4.1 Príprava prostredia . . . 17

4.1.1 Windows . . . 17

4.1.2 Linux Debian . . . 18

4.2 Návrhy riešenia . . . 18

4.3 Výsledky . . . 19

Záver **22**

Zoznam použitej literatúry **23**

Zoznam obrázkov a tabuliek

Obrázok 1 Zoznam .NET súčastí, ktoré sa vyvíjali v minulosti zvlášť, ale mali rovnaký základ a cieľ [3] 3

Obrázok 2 Zoznam aplikácií, ktoré sa môžu vyvíjať pomocou .NET [3] . . . 4

Obrázok 3 Spôsob, akým sa rôzne programovacie jazyky kompilujú a prekladajú do strojového kódu v Common Language Runtime (CLR) pomocou Common Intermediate Language (CIL) cez Common Language Infrastructure (CLI). [4] 5

Obrázok 4 Súčasti .NET Frameworku a ich postupný vývoj [4] 6

Obrázok 5 Príklad SELECT-u z databázy pomocou SQL príkazu. 8

Obrázok 6 Príklad získania dát z databázy pomocou ORM. 9

Obrázok 7 Vzorový SELECT pre EF ,(vzor je upravený pre zachovanie jednoduchosti a prehľadnosti) 20

Obrázok 8 Vzorový SELECT pre NHibernate 21

Tabuľka 2 Tabuľka pokusu pre SELECT z výpisov 7 a 8 21

Zoznam skratiek

API	Application Programming Interface
BCL	Base Class Libraries
CIL	Common Intermediate Language
CLI	Common Language Infrastructure
CLR	Common Language Runtime
CLS	Common Language Specifications
DB	Databáza
DLL	Dynamically linked library
EDMX	Entity Data Model XML
EF	Entity Framework
FCL	Framework Class Library
HQL	Hibernate Query Language
LINQ	Language-Integrated Query
MS	Microsoft
ORM	Object–relational Mapping
OS	Operating System
OSI	Open Source Initiative
POCO	Plain CLR Objects
RDMS	Relational Database Management System
SDK	Software Development Kit
SQL	Structured Query Language
TLDR	Too long; didn't read
VS	Visual Studio
XML	Extensible Markup Language

1 Zadanie

Vašou úlohou v rámci tohto projektu je získať odpovede na nasledujúce otázky:

- Čo je .NET?
- Čo je to .NET Entity Framework?
- Načo je .NET Entity Framework dobré?
- Kedy použiť .NET Entity Framework?
- Čo je NHibernate?
- Načo je NHibernate dobré?
- Kedy použiť NHibernate?
- Aký je rozdiel medzi .NET Entity Framework a NHibernate?

Odpovedzte na vyššie uvedené otázky vo svojej prezentácii.

V rámci tohto projektu budete pracovať s platformou .NET. Implementujte softvér, ktorý bude pracovať s .NET Entity Framework. Zmerajte čas, ktorý zaberú rôzne operácie s databázou (skúste dátové množiny inej veľkosti, skúste spustiť aj zložitejšie SQL dotazy (JOINy, vnorené dopyty)). Skúste tiež uviesť príklad SQL dotazu skonštruovaného Entity Frameworkom (môžete to urobiť napríklad Profilerom). Zhrňte svoje skúsenosti s .NET Entity Framework. Potom urobte všetko znova, ale namiesto Entity Framework použite NHibernate. Zahrňte aj zložitosť počiatočného nastavenia a konfigurácie. Pridajte nejaké namerané hodnoty, ak nejaké budú. Môžete si vytvoriť jeden alebo viacero programov a vybrať si, čo najlepšie vyhovuje vašim potrebám.

Zverejnite svoj softvér na GitHub, pridajte tam README.md s vašou dokumentáciou (upravte na markdown).

Vo svojej dokumentácii musíte uviesť nasledujúce informácie:

- Všetko uvedené vyššie napísané akademickým štýlom.
- Popíšte architektúru vášho softvéru. Zahrňte UML diagramy na vysvetlenie prípadov použitia a architektúry.
- Opíšte vstupné argumenty, ak vytvárate konzolovú aplikáciu.
- Popíšte príklady používania vášho softvéru.
- Popíšte výstupy, ak váš softvér generuje nejaké výstupy.
- Popíšte varovania a známe chyby, ak má váš softvér nejaké chyby.
- Popíšte niektoré zaujímavé prístupy používané na riešenie problémov s programovaním, ktorým ste čelili počas implementácie softvéru (ak je niečo zaujímavé).
- Popíšte, kedy a prečo sa odporúča používať .NET Entity Framework alebo NHibernate.
- Zhrňte svoje skúsenosti s .NET Entity Framework a NHibernate.

Niekoľko užitočných odkazov, ktoré by sa vám mohli hodiť:

- `https://bit.ly/3Vzmxow` Entity Framework vs NHibernate: Pochopte podobnosti a rozdiely
- `https://www.devbridge.com/articles/entity-framework-6-vs-nhibernate-4/` Entity Framework 6 vs NHibernate 4
- `https://stackoverflow.com/questions/17998609/entity-framework-vs-nhibernate-performance`
  God Bless StackOverflow

2 Teoretická časť

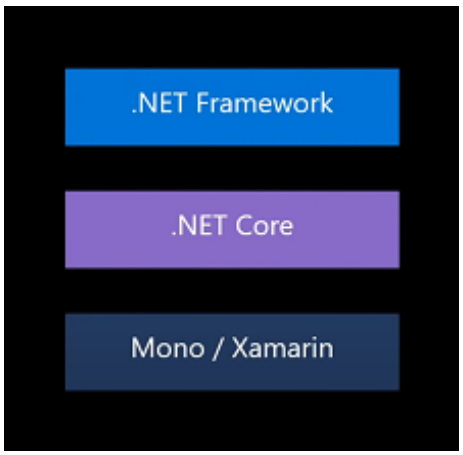
V tejto kapitole sa budeme venovať teoretickej časti práce. Naštudujeme si základné pojmy k téme .NET a NHibernate a detailnejšie sa oboznámime s ich frameworkami (po slovensky: *rámec*).

2.1 .NET

.NET (predtým sa nazývalo *.NET Core*) je bezplatný, open-source a manažovaný počítačový softvér pre operačné systémy *Windows, Linux a macOS*. Je to multiplatformový nástupca *.NET Frameworku*, ktorý bol iba pre Windows operačné systémy. Projekt je primárne vyvíjaný zamestnancami *Microsoftu* prostredníctvom *.NET Foundation* a vydaný pod licenciou MIT [1]. [1]

.NET by sa dal nazvať aj celkovým ekosystémom, ktorý pozostáva z viacerých častí ako čo sú viaceré programovacie jazyky ako napr. *C#, Visual Basic, F#*, naprogramovanými triedami a funkciami z Base Class Libraries (BCL) a platformami, ktoré boli vyvinuté pôvodne, ktoré vývojári neskôr spojili do jednotného softvérového development kitu:[2]

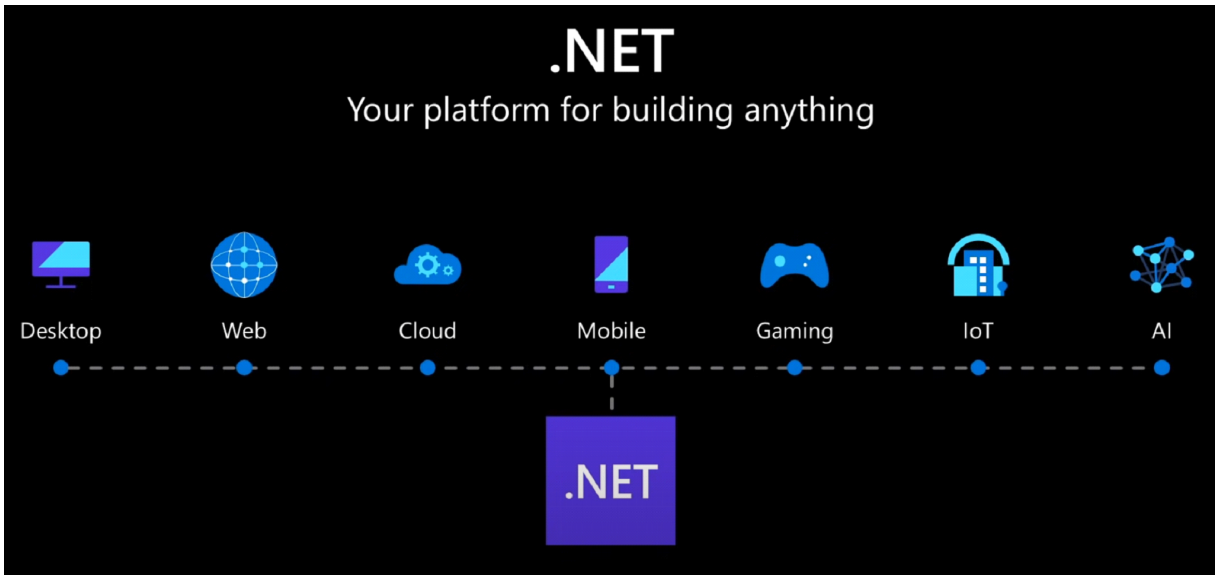
- .NET Framework (Windows OS, od 2002 r.)
- Mono / Xamarin (Linux OS od 2004 r.)
- .NET Core (Windows OS, Linux OS, MacOS, od 2016 r.) [3]



Obr. 1: Zoznam .NET súčastí, ktoré sa vyvíjali v minulosti zvlášť, ale mali rovnaký základ a cieľ [3]

[1] `https://opensource.org/licenses/MIT`

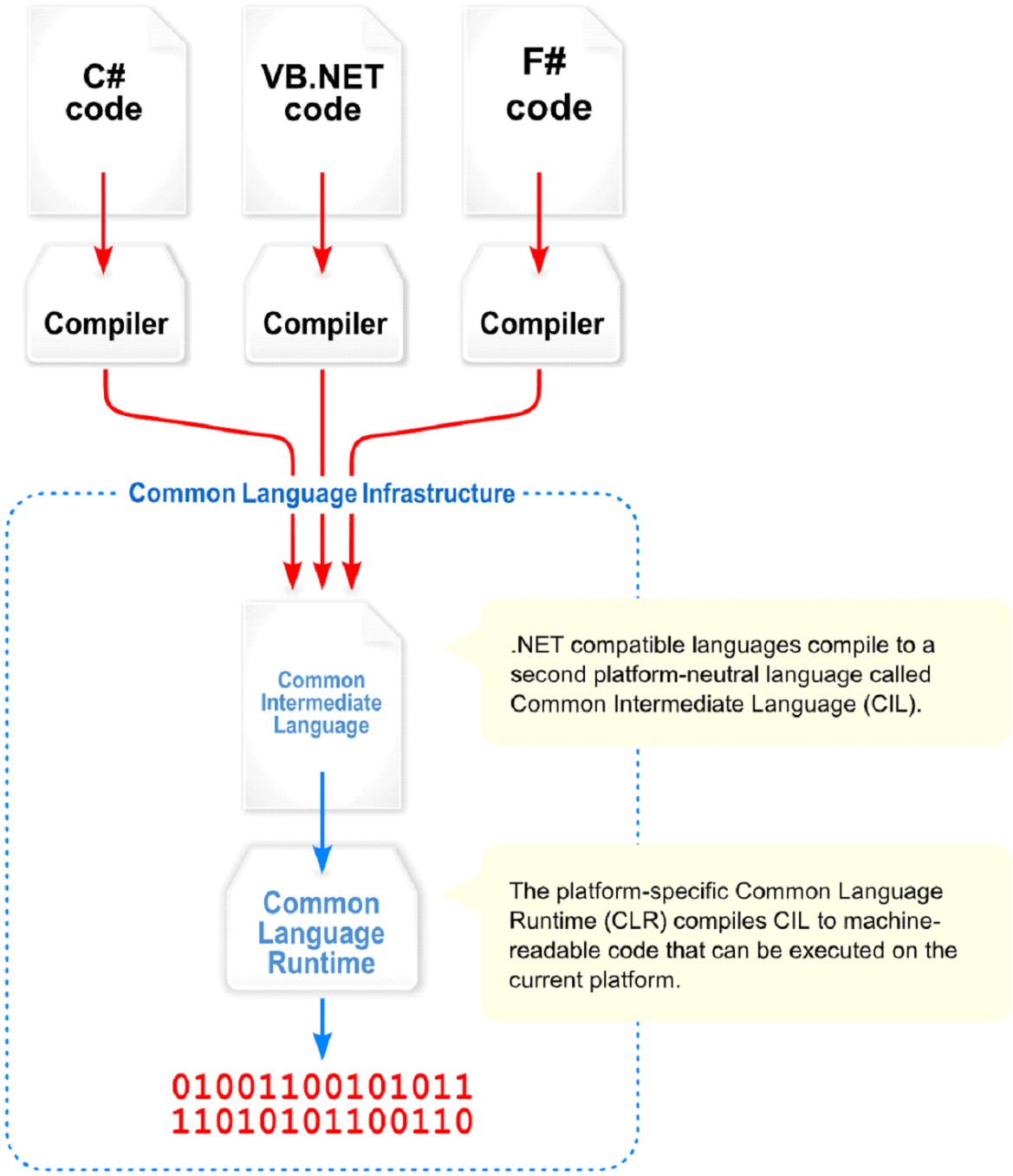
Všetky tieto časti .NET, spomenuté na obrázku č. 1 tvoria jeden veľký Software Development Kit (SDK), pomocou ktorého je možné vyvíjať širokú škálu multiplatformových aplikácií napr. ako to je ukázané na obrázku č. 2.



Obr. 2: Zoznam aplikácií, ktoré sa môžu vyvíjať pomocou .NET [3]

2.1.1 .NET Framework

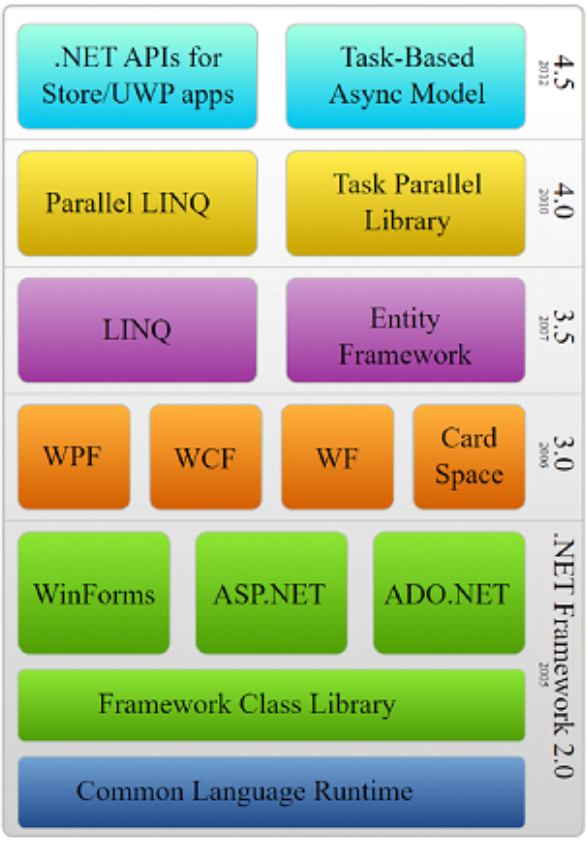
.NET alebo .NET Framework je proprietárny softvérový framework vyvinutý spoločnosťou Microsoft, ktorý beží predovšetkým na systéme Microsoft Windows. V minulosti tento framework bol ako prevládajúca implementácia spoločnej jazykovej infraštruktúry Common Language Infrastructure (CLI), kým nebol nahradený multi-platformovým projektom .NET.



Obr. 3: Spôsob, akým sa rôzne programovacie jazyky kompilujú a prekladajú do strojového kódu v Common Language Runtime (CLR) pomocou Common Intermediate Language (CIL) cez Common Language Infrastructure (CLI). [4]

.NET zahŕňa veľkú knižnicu tried s názvom Framework Class Library (FCL) a poskytuje jazykovú interoperabilitu, čo znamená, že každý programovací jazyk môže používať kód napísaný v iných jazykoch) v niekoľkých programovacích jazykoch. Všetky

časti .NET Frameworku môžete vidieť na obrázku č. 4. Programy napísané pre .NET Framework sa vykonávajú v softvérovom prostredí (na rozdiel od hardvérového prostredia) s názvom Common Language Runtime (CLR). CLR je aplikačný virtuálny stroj, ktorý poskytuje služby, ako je bezpečnosť, správa pamäte a spracovanie výnimiek. Počítačový kód napísaný pomocou .NET Framework sa preto nazýva „riadený kód“. FCL a CLR spolu tvoria .NET Framework.



Obr. 4: Súčasti .NET Frameworku a ich postupný vývoj [4]

FCL poskytuje užívateľské rozhranie, prístup k údajom, pripojenie k databáze, kryptografiu, vývoj webových aplikácií, numerické algoritmy a sieťovú komunikáciu. Programátori vytvárajú softvér kombináciou ich zdrojového kódu s .NET Framework a ďalšími knižnicami. Framework je určený na použitie vo väčšine nových aplikácií vytvorených pre platformu Windows. Microsoft tiež vyrába integrované vývojové prostredie pre softvér .NET s názvom *Visual Studio* [2]. [4]

.NET Framework začal ako proprietárny softvér, hoci firma pracovala na štandardizácii balíka softvéru takmer okamžite, ešte pred jeho prvým vydaním.

[2] `https://visualstudio.microsoft.com/`

2.1.2 .NET Core

V roku 2014 Microsoft oznámil .NET Core v snahe zahrnúť podporu pre .NET medzi platformami vrátane Linuxu a MacOS, zdroj pre implementáciu .NET Core CoreCLR, zdroj pre *„celú [...] knižnicu stack"* pre .NET Core a prijatie konvenčného ("bazárového") vývojového modelu s otvoreným zdrojom pod dohľadom nadácie .NET Foundation. Miguel de Icaza opisuje .NET Core ako *„prepracovanú verziu .NET, ktorá je založená na zjednodušenej verzii triednych knižníc"* a Immo Landwerth z Microsoftu vysvetlil, že .NET Core bude *„základom všetkých budúcich .NET platformy"*. V čase oznámenia bolo prvé vydanie projektu .NET Core nasadené podmnožinou zdrojových kódov knižníc a časovo sa zhodovalo s opätovným licencovaním existujúceho referenčného zdroja .NET od spoločnosti Microsoft mimo obmedzení Ms-RSL. Landwerth uznal nevýhody predtým vybranej zdieľanej licencie a vysvetlil, že kódové označenie Rotor urobilo *„nezačiatočným"* projektom s otvoreným zdrojovým kódom vyvinutým komunitou, pretože nespĺňalo kritériá licencie schválenej iniciatívou Open Source OSI.

.NET Core 1.0 bol vydaný v roku 2016 spolu s *Microsoft Visual Studio 2015 Update 3*, ktorý umožnil vývoj pomocou .NET Core. Neskoršie .NET Core verzie *1.0.4* a *1.1.1* boli vydané spolu s *.NET Core Tools 1.0* a *Visual Studiom 2017* v roku 2017.

2.1.3 Mono / Xamarin

Keď Microsoft prvýkrát predstavil svoj .NET Framework v decembri toho istého roku bola základná verzia Common Language Infrastructure (CLI) publikovaná ako otvorený štandard, „ECMA-335", čo otvorilo potenciál pre nezávislé implementácie. *Miguel de Icaza z Ximian* veril, že .NET má potenciál zvýšiť produktivitu programátorov a začal skúmať, či je linuxová verzia realizovateľná. Uvedomujúc si, že ich malý tím nemôže očakávať, že vytvorí a bude môcť podporovať kompletný produkt, takže na konferencii O'Reilly spustil projekt Mono s otvoreným zdrojovým kódom. [5]

Po troch rokoch vývoja bol vydaná prvá verzia Mono 1.0. Mono sa vyvinulo z počiatočného zamerania vývojárskej platformy pre linuxové desktopové aplikácie na podporu širokej škály architektúr a operačných systémov – vrátane vstavaných systémov.

Ximian bol v roku 2003 kúpený spoločnosťou Novell. A neskôr v roku 2011 Attachmate kúpil Novell. Attachmate oznámil prepúšťanie stovky zamestnancov a developérov Novellu, čo spochybnilo budúcnosť softweru Mono.

Miguel de Icaza na svojom blogu oznámil, že Mono bude naďalej podporovať Xamarin,

spoločnosť, ktorú založil po prepustení z Novellu. Pôvodný tím Mono sa tiež presťahoval do novej spoločnosti. Xamarin plánoval pokračovať v práci na Mono a plánoval prepísať proprietárne .NET zásobníky pre iOS a Android od nuly, pretože Novell v tom čase stále vlastnil MonoTouch a Mono pre Android. MonoTouch a Mono pre Android boli v priamej konkurencii s existujúcimi komerčnými ponukami, ktoré teraz vlastnil Attachmate, a vzhľadom na to, že tím Xamarin by mal problém dokázať, že nepoužil technológie, ktoré predtým vyvinul pre Novell. V 2011 však Novell, teraz dcérska spoločnosť Attachmate oznámili, že udelili trvalú licenciu spoločnosti Xamarin pre Mono, MonoTouch a Mono pre Android, ktorá oficiálne prevzala správu nad projektom. [5]

Xamarin je softvérová spoločnosť so sídlom v San Franciscu vlastnená spoločnosťou Microsoft a založená v roku 2011 inžiniermi, ktorí vytvorili Mono, Xamarin.Android (predtým Mono pre Android) a Xamarin.iOS (predtým MonoTouch), ktoré sú vzájomne prepojené platformové implementácie Common Language Infrastructure (CLI) a Common Language Specifications (často nazývané Microsoft .NET). [6]

Vďaka kódovej základni zdieľanej v jazyku C# môžu vývojári používať nástroje Xamarin na písanie natívnych aplikácií pre Android, iOS a Windows s natívnym používateľským rozhraním a zdieľanie kódu na viacerých platformách vrátane Windows, macOS a Linux. Podľa Xamarin viac ako 1,4 milióna vývojárov používalo produkty Xamarin v 120 krajinách po celom svete (dáta k aprílu 2017). [6]

2016 Microsoft oznámil, že podpísal definitívnu dohodu o akvizícii Xamarinu. [6]

2.2 Čo je Object–relational Mapping (ORM)?

ORM je framework, ktorý nám umožňuje mapovať objekty na riadky v relačných tabuľkách v relačných databázach, čím vznikne ľahší prístup k údajom a aktualizovanie cez objektový model.

```
SELECT * FROM users WHERE email = 'test@test.com';
```

Obr. 5: Príklad SELECT-u z databázy pomocou SQL príkazu.

Je schopnosť písať klasické queries, ako napríklad na obrázku vyššie, či aj oveľa komplikovanejšie, pomocou objektovo orientovanej paradigmy určitého preferovaného programovacieho jazyka. V skratke, snažíme sa interagovať s našou databázou pomocou

nami zvoleného jazyka namiesto SQL príkazov.

```
var orm = require('generic-orm-libarry');
var user = orm("users").where({ email: 'test@test.com' });
```

Obr. 6: Príklad získania dát z databázy pomocou ORM.

Objektovo-relačné mapovače nám vytvoria spojenie medzi objektovým a relačným svetom. NHibernate bol dlhú dobu hlavnou voľbou pri výbere ORM pre .NET platformu. Je to vyspelý a overený framework so širokou užívateľskou základňou. V poslednej dobe však výrazne rastie jeho hlavný konkurent, Entity Framework, obzvlášť od roku 2012, kedy bol vydaný ako open-source.
Pri porovnávání sme sa sústredili na vlastnosti, ktoré sú dôležité ako pri vývoji velkých enterprise aplikací pracujúcich s veľkými dátami, tak aj pri vývoji miniaplikací a prototypov, kde je prvoradou požiadavkou rýchlosť a nízka cena vývoja. Je jasné, že z takéhoto porovnania nie je možné vyvodiť jednoznačný záver, keďže ide o typy aplikácií s navzájom do veľkej miery protichodnými požiadavkami.

2.3 .NET Entity Framework (EF)

Entity Framework (EF) je lightweight, rozšíriteľný, open-source a cross-platformový objektovo relačný model, ktorý pomáha pri prístupe k databázovým záznamom a pri vykonávaní hrubých operácií s databázami. Pomáha tiež pri mapovaní entít na objekty špecifické pre ich konkrétnu doménu, čím umožňuje písať menej kódu na spracovanie databázovej vrstvy. [7]

Entity Framework je set technológií v ADO.NET, ktorý podporuje vývoj dátovo orientovaných softvérových aplikácií. Architekti a vývojári dátovo orientovaných aplikácií zvyčajne zápasili s potrebou dosiahnuť dva veľmi odlišné ciele. Bolo treba modelovať entity, vzťahy a logiku obchodných daných problémov, ktoré sa pokúšali riešiť, a tiež treba pracovať s dátovými enginmi používanými na ukladanie a získavanie údajov. Dáta môžu pokrývať viacero úložných systémov, z ktorých každý má svoje vlastné protokoly; dokonca aj aplikácie, ktoré pracujú s jedným úložným systémom, musia vyvážiť požiadavky úložného systému s požiadavkami na písanie efektívneho a udržiavateľného aplikačného kódu. Tento problém sa vo všeobecnosti označuje ako „nesúlad objekt-relačná impedancia“. [7]

Mnoho nástrojov na objektovo-relačné mapovanie (ORM) (známy ako „objektovo-relační manažéri“) bolo vyvinutých, aby umožnili vývojárom pracovať s údajmi vo forme objektov a vlastností špecifických pre doménu, ako sú zákazníci a adresy zákazníkov, bez toho, aby sa museli obávať so základnými databázovými tabuľkami a stĺpcami, kde sú tieto údaje uložené. S ORM môžu vývojári pracovať na vyššej úrovni abstrakcie pri práci s dátami a môžu vytvárať a udržiavať dátovo orientované aplikácie s menším množstvom kódu ako v tradičných aplikáciách. Entity Framework je riešenie ORM, ktoré sa v súčasnosti propaguje na použitie v rámci vývojového balíka spoločnosti Microsoft. [8]

2.3.1 Výhody EF

- Poskytuje automaticky generovaný kód,
- Umožňuje vývojárom vizuálne navrhovať modely a mapovať databázu,
- Umožňuje mapovanie viacerých koncepčných modelov do jednej schémy úložiska,
- Ľahké mapovanie business objektov (pomocou tabuliek a dragdrop).
- Znižuje náklady na vývoj [9]

2.3.2 Nevýhody EF

- Komplikovaná syntax,
- Logická schéma databázy nie je schopná využívať niektoré časti aplikácie,
- Nie je k dispozícii pre každý RDMS. [9]

2.4 NHibernate

NHibernate je open source objektovo-relačný mapovač pre .NET. framework. Jeden z najstarších a najviac rešpektovaných objektovo-relačných mapovačov(OMRs). Existuje od roku 2003. NHibernate bol dostupný ešte pred prvým uvedením Entity Frameworku. Je to časť Hibernate - Java objektovo-relačný mapovač. Umožňuje mapovať doménové objekty do tradičných databáz. Úzko spolupracuje s databázovou vrstvou a Plain CLR Objects (POCO). NHibernate vytvára SQL na načítanie a ukladanie objektov na základe XML popisu vašich entít a vzťahov. Možno ho považovať za systém správy databáz. [10]

2.4.1 Nevýhody

- Dlhší čas spustenia v dôsledku prípravy metadát (nevhodné pre aplikácie typu desktop).
- Vysoký learning curve bez predchádzajúcej skúsenosti s ORM. Jeho zvládnutie na vyššiu úroveň trvá dlhší čas.
- Vysoké nároky na DB schému. Je nutné aby DB bola dobre nadizajnovaná a aby boli spoľahlivo určené obmedzenia.
- Vyžaduje sa určitá konfigurácia XML súborov. Implementácia mapovania do XML môže byť veľmi únavná najmä pre veľké databázy so stovkami až tisíckami tabuliek. [11]

2.4.2 Výhody

- Flexibilné a veľmi bohaté možnosti mapovania.
- Má podporu cacheovania druhej úrovne (často využívané)
- Podporuje 6 druhov kolekcií (list, set, bag, map, array, id array), EF podporuje iba jednu.
- Má 10+ id generátorov (IDENTITY, sequence, HiLo, manual, increment, niekoľko GUIDs...), EF má iba manual alebo SQL Server IDENTITY.
- NH podporuje primitive types (strings, integers, atď) a aj komponenty (complex types without identity), EF nepodporuje. [12]

2.5 Vlastnosti Entity Framework a NHibernate

V tejto podkapitole si povieme všetky najdôležitejšie vlastnosti oboch frameworkov, porovnáme jednotlivé vlastnosti a poukážeme na rozdiely, ktoré by mohli ovplyvniť naše výsledky v praktickej časti našej práce.

2.5.1 LINQ

Zásadnou súčasťou frameworku je spôsob dopytovania do databázy. To má vplyv hlavne na rýchlosť vývoja. Poďme sa pozrieť na podporu LINQ, čo je pre .NET programátora elegantný a hlavne rýchly prístup pre písanie queries nad objektami, nielen entitnými (databázovými). Preto jeho podporu v spojení s DB frameworkom vidíme ako dôležitú. Na druhú stranu sa predpokladá, aby si programátor dokázal uvedomiť, ako bude v princípe takto zapísaný query preložený do SQL, a teda čo to bude znamenať pre DB server z pohľadu výkonnosti a veľkosti prenášaných dát medzi databázou a aplikáciou.

Zatiaľ čo NHibernate sa vyvíjal dávno pred dobami jazyka LINQ a má preto niekoľko iných spôsobov, ako vytvárať queries, EF stojí a padá na LINQ ako na primárnom query mechanizme. Z toho je pochopiteľné, že podpora LINQ v EF je na vysokej úrovni, zatiaľ čo NHibernate má v podpore LINQ slabiny (jeho provider LINQ to NHibernate nepodporuje napr. ORDER BY, LEFT JOIN a pod.). [13]

2.5.2 Vlastné implementácie Entity SQL a HQL

Ako NHibernate tak EF definujú vlastné query jazyky, ktoré je možné použiť v situáciách, kedy LINQ alebo iné prístupy nestačia. Syntakticky ide veľmi zjednodušene o dialekty jazyka SQL, ktoré naviac umožňujú pracovať s ORM objektami (entitami) daného frameworku.

NHibernate má query jazyk zvaný HQL, ktorý použijeme hlavne pre písanie zložitých queries, napríklad s kombináciami inner, outer a cross joinu, pri potrebe komplikovaných podmienok, SQL funkcií, atď. Neumožňuje ho však na rozdiel od EF mapovať na použitie LINQ, je teda určený len pre explicitné dotazy.

EF definuje jazyk Entity SQL. Podobne ako v prípade HQL u NHibernate ho možno použiť pre zložitejšie queries, ktoré by pomocou štandardného LINQ boli zložité či neoptimálne z pohľadu výkonnosti alebo nešli napísať vôbec. Naviac ale EF umožňuje namapovať Entity SQL kód na vlastnú .NET metódu. Môžeme tak doplniť do EF napr. SQL funkciu LIKE alebo akúkoľvek ďalšiu vlastnú špecifickú funkciu. Takto definovanú funkciu potom môžeme volať v LINQ dotazoch, čo NHibernate neumožňuje. EF sa pri serializácií LINQ dotazu postará o preloženie z .NET cez Entity SQL až do vlastného SQL kódu, ktorý je následne interpretovaný na databázu. Na druhú stranu, bez Entity SQL funkcií sa možno obísť, buď o niečo zložitejším .NET kódom, alebo použitím uložených procedúr. To predstavuje malé zvýšenie pracnosti.

Obecne použitie HQL tak isto ako použitie Entity SQL má nevýhody, ako pri programování, tak hlavne pri refaktoringu. Queries sú totiž zapísané v podobe reťazcového

literálu, a nie sú tak syntakticky kontrolované pri preklade. [13]

2.5.3 Lazy fetching

Lazy fetching (v EF označovaný aj Lazy loading) – V NHibernate je výrazne konfigurovateľný, vrátane načítania jednotlivých stĺpcov (typu BLOB). V EF je možné ho zapnúť/vypnúť globálne na celý model, kde sú načítané vždy celé kolekcie entít, tj. join do referencovanej tabuľky. Možnost lazy fetchingu na úrovni properties (stĺpcov) je sľúbená pro EF7.

Pri nesprávnom použití má lazy fetching velké problémy s výkonom. Napríklad pri načítaní mnoho prvkov z podriadenej kolekcie, kde načítanie každého prvku kolekcie môže spôsobiť jeden vstup do databázy (toto správanie je možné u NHibernate ovplyvniť nastavením tzv. fetching strategy a načítať napr. viac záznamov z kolekcie naraz). U správne navrhnutej aplikácie by sme mali dopredu vedieť, ktoré všetky vzťahy alebo stĺpce potrebujeme do aplikacie načítať, a načítať ich iba ak to je možné naraz, v jednej query do DB. Tento prístup sa nazýva *eager fetching* a oba frameworky ho podporujú.

Chovanie NHibernate a EF sa značne líši v prípade vypnutia lazy fetchingu (oba frameworky to povolujú). V takomto prípade NHibernate načíta podriadenej kolekcie ihneď (rekurzívne), čo môže najhoršie skončiť načítaním celé databáze. Preto se v prípade NHibernate nedoporučuje lazy fetching globálne vypínať (umožňuje ho vypnúť v niektorých prípadoch na niektorých entitách). V EF vypnutie lazy fetchingu nevadí, pretože pri načítaní entity nechá referencie na podriadené kolekcie na null a umožňuje neskôr dočítať ich obsah, čo sa nazýva explicit loading.

TLDR - čo sa týka lazy fetchingu, je o niečo lepší NHibernate vďaka jemnejšej konfigurovateľnosti. [13]

2.5.4 Typy ORM mapovania

Rôzne možnosti, ako definovať ORM mapovanie – mapovanie objektov (properties) na tabuľky resp. stĺpce v databáze. [13]

- Mapovanie pomocou XML
  - EF

    EDMX Model EF je reprezentovaný v jedinom XML súbore, ktorý obsahuje časť popisujúcu databázovú schému, časť objektového modelu, a nakoniec mapovanie medzi nimi. Model je možné editovať modelerom vo VS, nástrojmi tretích strán (angl. *third-party*), alebo ručne. Vo VS je možné:

    - vytvoriť EDMX model a z neho vygenerovať skripty pre vytvorenie alebo zmenu DB schémy (model first prístup)
    - model vygenerovať priamo z databázy (tzv. *DB first access*, po slovensky DB prvý prístup)
  - NHibernate
    Pôvodná metóda mapovania. Mapovanie je uložené v samostatných XML súboroch. Nevýhodou je, že správnosť mapovania nie je kontrolovaná pri kompilácii.

- Mapovanie atribútmi
  Označovanie tried a properties entity atribúty určujúce tabuľku, stĺpec a ďalšie informácie. Nevýhodou je zanášanie závislosti na frameworku do entitnej vrstvy/doménového modelu.

  - EF
    Podporované, ako súčasť Code First prístupu k vývoju.
  - NHibernate
    Podporované v externom projekte *NHibernate.Mapping.Attributes*. Výhodou oproti XML je lepšia kontrola pri kompilácii.

- Mapovanie kódom
  Mapovanie špecifikované v kóde, volaním API metód, ktoré framework poskytuje. Výhodami sú kontrola pri preklade a možnosť mapovanie oddeliť od projektu entitnej vrstvy/doménového modelu.

  - EF
    Podporované, ako súčasť Code First prístupu k vývoju (Fluent API).
  - NHibernate
    Podporované od verzie 3.2. V prípade použitia s nižšou verziou, je nutné použiť externú knižnicu Fluent NHibernate.

- Konvenčné mapovanie (Automapping)
  Namiesto explicitného mapovania každej property na stĺpec stanovíme tzv. konvencie, čo je napr. „každá property končiaca na ID je primárny kľúč“. Obvyklou konvenciou je, že stĺpce tabuľky sa volajú rovnako ako properties a tabuľky rovnako ako triedy.

  - EF
    Podporované od verzie 6 ako súčasť Code First.

– NHibernate
  Podporované od verzie 3.2. V prípade použitia s nižšou verziou, je nutné použiť externú knižnicu Fluent NHibernate.

Oba frameworky majú v základných princípoch podobné možnosti, čo sa týka mapovania, napriek tomu, že realizácia je značne odlišná. Nie je to ale žiadny podstatný nedostatok, ktorý by stál pri porovnaní frameworkov za zmienku. V prospech EF je možno iba editor EDMX mapovania, dodávaný ako súčasť Visual Studia. Nie je teda potrebné spoliehať sa na 3rd party nástroj, pokiaľ by bol pre zvolený typ mapovania potrebný.

2.5.5 Databázová podpora

- EF
  V prípade EF je primárna podpora pre MS SQL server, s ostatnými DB už nie je EF integrovaný na tak vysokej úrovni. Je to dané tiež tým, že EF je navrhnutý primárne pre MS SQL. Niektoré jeho špecifické vlastnosti tak nemôžu byť z princípu pre ostatné DB implementované a naopak iné špecifické vlastnosti iných DB nemôžu byť plne využité v EF. Je preto potrebné sa pripraviť na drobné problémy, pokiaľ použijete EF s inou databázou, než MS SQL napríklad vyžadovanie dodatočných pripojení na jej spracovanie.

- NHibernate
  Podporuje MS SQL Server, Oracle, DB2, MySQL, SQLite, PostgreSQL a Firebird.

V oblasti podpory jednotlivých databáz má navrch NHibernate.

2.5.6 Dokumentácia

- EF
  EF sa môže pýšiť kvalitnou dokumentáciou, silnou programátorskou komunitou, a takisto aj veľkým množstvom informácií, problémov a riešení na internete. V tomto si EF vedie v porovnaní s NHibernate oveľa lepšie. [14]

- NHibernate
  Dokumentácia NHibernate má svoje slabé stránky a nepatrí medzi tie najrozsiahlejšie, môže sa stať, že podrobný popis určitej funkcionality tam nie je a programátor je nútený hľadať z iných zdrojov. Na druhú stranu, komunita používateľov je veľká a nie je problém nájsť riešenie väčšiny problémov, s ktorými sa vývojár stretne. [11]

3 Zhodnotenie .NET EF vs NHibernate

V tejto kapitole si zhrnieme naštudované poznatky, zhodnotíme si porovnania dvoch frameworkov a zároveň si overíme či sú naše poznatky pravdivé v kapitole č. 4.1.1.

3.1 EF

Pri použití vlastností podporovaných v EF je vývoj v tomto frameworku rychlejší v porovnání s NHibernate. Pre C# programátora vypadá EF vyspelejšie, pomerne ľahko sa s ním pracuje, a má menšiu learning curve. Entity Framework by sme vybrali určite pre nové menšie projekty. V prospech EF hovorí aj fakt, že sa rýchlo vyvíja a väčšinu slabín je možné s určitým navýšením pracnosti úspešne riešiť.

Na druhú stranu EF nie je v niektorých prípadoch vhodný na prácu s veľkými dátami a má radu nedostatkov. Väčšinu z nich je možné riešiť rôznymi workaroundami, third-party knižnicami, atď. To už ale zase na druhú stranu jeho použitie komplikuje, vytvára nároky na špecifické vedomosti vývojárov a zvyšuje riziko pri použití vo veľkých enterprise projektoch. O EF môžeme aj na veľkých projektoch uvažovať v prípade, keď máme pod kontrolou aj návrh fyzického DB modelu a sme teda schopní bez zásadných komplikácií obmedziť vyššie popísané situácie, s ktorými si EF nedokáže rozumne poradiť.

3.2 NHibernate

NHibernate je odladený framework, dlhú dobu používaný mnohými vývojármi. Použiť ho je vhodné v prípade inej DB než MS SQL Server, pretože má väčšiu a viac konzistentnú podporu iných DB. NHibernate je vhodné ďalej zvážiť vo veľkých projektoch, kde je zásadný výkon a škálovateľnosť.

Hlavné riziko NHibernate je však jeho podpora do budúcnosti. Implementácia funkcií z nových verzií .NET frameworku sa výrazne spomalila a znížil sa počet vývojárov. Použitie niektorých štandardných vlastností je v NHibernate o niečo pračnejší než v EF, slabinou NHibernate je tiež dokumentácia.

4 Praktická časť

Praktická časť pozostáva z porovnania rýchlosti a zložitosti oboch riešení našej implementácie. Táto kapitola obsahuje aj tabuľku dosiahnutých výsledkov a prípravu pracovného prostredia pre Windows a Linux Debian.

4.1 Príprava prostredia

V tejto podkapitole si vysvetlíme postup nastavenia prostredia, nainštalovania si potrebných aplikácií, programovacích prostredí, SDK inštancií, vytvorenia, spustenia databázy, celkového nasadenia projektu pre testovanie výkonu EF a NHibernate a ich prístupu k databáze.

4.1.1 Windows

- Nainštalovanie si vývojového prostredia *Visual Studio* [3]
- Nainštalovanie si potrebných .NET súčastí [4][5][6][7][8][9]
- Vytvorenie si databazy pomocou príručky [10]
- Naklonovanie si github repozitára do Visual Studio prostredia a spustenie testovania vykonu [11]

[3] https://visualstudio.microsoft.com/
[4] https://dotnet.microsoft.com/en-us/download
[5] https://learn.microsoft.com/en-us/sql/connect/ado-net/microsoft-ado-net-sql-server?view=sql-server-ver16
[6] https://learn.microsoft.com/en-us/ef/
[7] https://learn.microsoft.com/en-us/ef/core/
[8] https://dotnet.microsoft.com/en-us/apps/aspnet
[9] https://www.microsoft.com/en-us/sql-server/sql-server-downloads
[10] https://learn.microsoft.com/en-us/sql/database-engine/install-windows/install-sql-server?view=sql-server-ver16
[11] https://bit.ly/asos-efvsnh

4.1.2 Linux Debian

- CLI nástroj dotnet[12] - návod na inštaláciu nástroja pre spustenie hlavnej aplikácie
- Dockerizácia existujúci aplikácií[13] - v Dockerfile stačí zmeniť cestu k vytvorej DLL aplikácii

Následne treba vytvoriť a spustiť vytvorenú aplikáciu v dockeri pomocou nasledovných príkazov:

```
docker build -t asos_app .
docker run -d -p 8080:80 -name myapp asos_app
```

4.2 Návrhy riešenia

Pri návrhu riešenia sme sa inšpirovali existujúcimi riešeniami dostupnými na platforme Github[14],[15],[16],[17].

Všetky z riešení vyžadovali tvorbu databázy, či už typu *Database SQL Sever* alebo *Database SQL File*.

Riešenie používa testovacie dáta[18]. Naplnenie databázy bolo vyrobené pomocou odporúčaného návodu z Microsoft webstránky[19]. V niektorých prípadoch testovania, bolo potrebné vstupné dáta dodatočne upraviť.

[12]https://github.com/dotnet/dotnet-docker/blob/main/documentation/scenarios/installing-dotnet.md

[13]https://docs.docker.com/samples/dotnetcore/

[14]https://github.com/dmcliver/NHibernateVsEf

[15]https://github.com/nhibernate/NHibernate.Mapping.Attributes

[16]https://github.com/codexguy/CodexMicroORM

[17]https://github.com/leotx/performance-test

[18]http://ocelma.net/MusicRecommendationDataset/lastfm-1K.html

[19]https://learn.microsoft.com/en-us/visualstudio/data-tools/create-a-sql-database-by-using-a-designer?view=vs-2022

4.3 Výsledky

Zo testovacích dát[20] v databáze sme vybrali query , ktorá nám vrátila najpopulárnejšieho hudobného umelca podľa počtu prehratí piesní. Jedna zo vzoriek obsahuje 1000 záznamov a druhý viac ako 100 000. V tomto príklade sme používali dataset s väčším počtom vzoriek, aby sme mohli poukázať na efektivitu frameworku NHibernate.

[20] http://ocelma.net/MusicRecommendationDataset/lastfm-1K.html

```
01 | SELECT
02 | [Project4].[C2] AS [C1],
03 | [Project4].[Name] AS [Name],
04 | [Project4].[C1] AS [C2]
05 | FROM ( SELECT
06 |         [GroupBy1].[A1] AS [C1],
07 |         [GroupBy1].[K1] AS [Name],
08 |         1 AS [C2]
09 |         FROM ( SELECT
10 |                 [Extent1].[Name] AS [K1],
11 |                 COUNT(1) AS [A1]
12 |                 FROM  [dbo].[Artist] AS [Extent1]
13 |                 INNER JOIN [dbo].[Track] AS [Extent2] ON  EXISTS (SELECT
14 |                         1 AS [C1]
15 |                         FROM    ( SELECT 1 AS X ) AS [SingleRowTable1]
16 |                         LEFT OUTER JOIN  (SELECT
17 |                                 [Extent3].[Id] AS [Id]
18 |                                 FROM [dbo].[Artist] AS [Extent3]
19 |                                 WHERE [Extent2].[ArtistId] = [Extent3].[Id] )
        AS [Project1] ON 1 = 1
20 |                         LEFT OUTER JOIN  (SELECT
21 |                                 [Extent4].[Id] AS [Id]
22 |                                 FROM [dbo].[Artist] AS [Extent4]
23 |                                 WHERE [Extent2].[ArtistId] = [Extent4].[Id] )
        AS [Project2] ON 1 = 1
24 |                         WHERE [Extent1].[Id] = [Project1].[Id]
25 |                 )
26 |                 GROUP BY [Extent1].[Name]
27 |         )  AS [GroupBy1]
28 | )  AS [Project4]
29 | ORDER BY [Project4].[C1] DESC
```

Obr. 7: Vzorový SELECT pre EF ,(vzor je upravený pre zachovanie jednoduchosti a prehľadnosti)

```
01 |  SELECT TOP (1)  a.Name as ArtistName, count(t.Id) as TrackCount
02 |  FROM Track t
03 |  INNER JOIN Artist a ON t.ArtistId = a.Id
04 |  GROUP BY a.Name
05 |  ORDER BY COUNT(t.Id) DESC
```

Obr. 8: Vzorový SELECT pre NHibernate

Medzi skonštruovanými príkazmi vo výpisoch 8 a 7 si môžeme všimnúť značný rozdiel vo veľkosti zvolených SELECT-ov.

Okrem pozorovania značnej jednoduchosti v zápise SELECTU pre NHibernate, sme otestovali aj rýchlosť oboch riešení a výsledky zapísali do tabuľky 2. Pre stĺpce *NHibernate* a *EF* v tabuľke uvádzame počet vrátených záznamov za daný čas.

Číslo pokusu	*Čas spustenia*	*NHibernate*	*EF*
1	3ms	98	96
2	4.5ms	251	245
3	9ms	449	440

Tabuľka 2: Tabuľka pokusu pre SELECT z výpisov 7 a 8

Záver

V dokumente sme si predstavili teoretické a praktické porovnanie frameworkov Entity Framework a NHibernate. Prostredie sme si vytvorili na operačnom systéme Windows 10 a Ubuntu 18.04. Na realizáciu projektu sme použili zodpovedajúce nástroje, vhodné na prácu ako je Visual Studio alebo Docker. Databáza MS SQL Server k projektu bola vytvorená lokálne. Iný návrh databázového servera nebol možný, nakoľko jedno z riešení používa len tento návrh. Práca obsahuje detailný rozpis štruktúry oboch elementov ako aj porovnanie rýchlosti pri zložitom SELECT príkaze z lokálnej databázy.

Zoznam použitej literatúry

1. *.NET* [online]. [cit. 2022-11-01]. Dostupné z : https://en.wikipedia.org/wiki/.NET.
2. *What is .NET? | .NET Core 101 [1 of 8]* [online]. [cit. 2022-11-01]. Dostupné z : https://www.youtube.com/watch?v=eIHKZfgddLM&list=PLdo4fOcmZOoWoazjhXQzBKMrFuArxpW80&index=1.
3. *What is .NET? What's C# and F#? What's the .NET Ecosystem? .NET Core Explained, what can .NET build?* [online]. [cit. 2022-11-01]. Dostupné z : https://www.youtube.com/watch?v=bEfBfBQq7EE.
4. *.NET Framework* [online]. [cit. 2022-11-01]. Dostupné z : https://en.wikipedia.org/wiki/.NET_Framework.
5. *Mono (software)* [online]. [cit. 2022-11-01]. Dostupné z : https://en.wikipedia.org/wiki/Mono_(software).
6. *Xamarin* [online]. [cit. 2022-11-01]. Dostupné z : https://en.wikipedia.org/wiki/Xamarin.
7. *.NET Entity Framework* [online]. [cit. 2022-11-01]. Dostupné z : https://en.wikipedia.org/wiki/Entity_Framework.
8. *Entity Framework overview* [online]. [cit. 2022-11-01]. Dostupné z : https://learn.microsoft.com/en-us/dotnet/framework/data/adonet/ef/overview.
9. *Advantages and Disadvantages of Entity Framework* [online]. [cit. 2022-11-01]. Dostupné z : https://cybarlab.com/advantages-and-disadvantages-of-ef.
10. *Wikipedia - Nhibernate* [online]. [cit. 2022-11-01]. Dostupné z : https://en.wikipedia.org/wiki/NHibernate.
11. *Advantages and Disadvantages of NHibernate* [online]. [cit. 2022-11-01]. Dostupné z : https://stackoverflow.com/questions/1278094/advantages-and-disadvantages-of-nhibernate/1278154.
12. *Why use NHibernate ? Advantages of NHibernate* [online]. [cit. 2022-11-01]. Dostupné z : https://mudassirshahzad.com/why-use-nhibernate-advantages-of-nhibernate/.
13. *Nhibernate nebo entity framework dil c.1* [online]. [cit. 2022-11-01]. Dostupné z : https://profinit.eu/blog/nhibernate-nebo-entity-framework-dil-c-1/.

14. *What is Entity Framework?* [online]. [cit. 2022-11-01]. Dostupné z : https://www.entityframeworktutorial.net/what-is-entityframework.asp.