# Automatizované testy aplikací ve frameworku ASP.Net Core

Bc. Lukáš Machajdík

Diplomová práce
2022

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

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

Akademický rok: 2021/2022

# ZADÁNÍ DIPLOMOVÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

| | |
|---|---|
| Jméno a příjmení: | **Bc. Lukáš Machajdík** |
| Osobní číslo: | **A20190** |
| Studijní program: | **N0613A140022 Informační technologie** |
| Specializace: | **Softwarové inženýrství** |
| Forma studia: | **Prezenční** |
| Téma práce: | **Automatizované testy aplikací ve frameworku ASP.Net Core** |
| Téma práce anglicky: | **Automated Application Tests in ASP.Net Core** |

## Zásady pro vypracování

1. Popište současný stav technologií pro testování softwaru.
2. Zaměřte se na automatizované testy webových aplikací pomocí knihovny Selenium, knihovnu bUnit, framework .NET Core, Blazor a Entity Framework.
3. Navrhněte ukázkovou webovou aplikaci a implementujte automatizované testy.
4. Popište klíčové části řešení a doporučené postupy.
5. Demonstrujte výsledky a formulujte závěr.

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

Seznam doporučené literatury:


1. Blazor [online]. Redmond, Washington, USA: Microsoft, 2021 [cit. 2021-10-18]. Dostupné z: https://blazor.net/
2. BUnit: a testing library for Blazor components [online], c2021. Egil Hansen [cit. 2021-10-18]. Dostupné z: https://bunit.dev
3. Selenium [online], c2021. Chicago: ThoughtWorks [cit. 2021-10-18]. Dostupné z: https://www.selenium.dev
4. ASP.NET [online]. Redmond, Washington, USA: Microsoft, 2021 [cit. 2021-10-18]. Dostupné z: https://docs.microsoft.com/cs-cz/aspnet/core/?view=aspnetcore-5.0
5. FREEMAN, Adam, 2020 ;. Pro ASP.NET Core 3: Develop Cloud-Ready Web Applications Using MVC, Blazor, and Razor Pages. 8th ed. 2020. Berkeley, CA: APress, 1 online zdroj (XXIX, 1080 stran). ISBN 9781484254400.
6. PECINOVSKÝ, Rudolf, 2007. Návrhové vzory: [33 vzorových postupů pro objektové programování]. Brno: Computer Press, 527 s. ISBN 9788025115824.
7. ANGELOV, Anton, 2020. Design Patterns for High-Quality Automated Tests: High-Quality Test Attributes and Best Practices. Independently published. ISBN 9798621316983.


Vedoucí diplomové práce: **Ing. Erik Král, Ph.D.**
Ústav počítačových a komunikačních systémů

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

L.S.

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

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

Ve Zlíně dne 24. ledna 2022

**Prohlašuji, že**

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

**Prohlašuji,**

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

Ve Zlíně, dne Lukáš Machajdík, v.r.

## ABSTRAKT

V teoretickej časti tejto práce si povieme o frameworkoch .Net Core, načrtneme Asp.Net, a ďalej sa budeme venovať frameworku Blazor, Entity Framework a automatizovaním testov webových stránok pomocou rôznych knižníc, ako napríklad Selenium, Robot Framework a bUnit.

V praktickej časti sa čitateľ dočíta o vytvorenej webovej stránke, ktorá využíva frameworky Blazor a Entity Framework, ale najmä o automatizovaných testoch, ktoré využívajú knižnicu bUnit, a ktoré testujú správnosť chovania komponentov na stránke.

Kľúčové slová: .Net Core, Blazor, Entity Framework, automatizované testovanie, bUnit

## ABSTRACT

In theoretical part of this thesis, we will talk about frameworks .Net Core, briefly about Asp.Net, and later our focus will be at framework Blazor, Entity Framework, and automated web tests using different libraries such as Selenium, Robot Framework and bUnit.

In practical part, we will talk about the created website, which uses frameworks Blazor and Entity Framework, but also about automated tests, which use library bUnit and test the correct behavior of components on website.

Keywords: .Net Core, Blazor, Entity Framework, automated testing, bUnit

Týmto by som sa chcel poďakovať vedúcemu tejto diplomovej práce Ing. Erikovi Královi Ph.D. za jeho odbornú pomoc, cenné rady a pripomienky.

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

# OBSAH

**ÚVOD** ..... **8**
**TEORETICKÁ ČÁST** ..... **9**
**1 TESTOVANIE SOFTVÉRU** ..... **10**
1.1 MANUÁLNE TESTOVANIE SOFTVÉRU ..... 10
1.1.1 Exploratory testing ..... 10
1.1.2 Vizuálne regresné testy ..... 10
1.2 AUTOMATIZOVANÉ TESTOVANIE SOFTVÉRU ..... 10
1.2.1 End-to-End (E2E) testy ..... 11
1.2.2 Integračné testy ..... 11
1.2.3 Jednotkové testy ..... 11
**2 NÁSTROJE TESTOVANIA SOFTVÉRU** ..... **13**
2.1 SELENIUM ..... 13
2.1.1 Selenium IDE ..... 14
2.1.2 Selenium RC ..... 17
2.1.3 Selenium WebDriver ..... 18
2.1.4 Selenium Grid ..... 20
2.2 ROBOT FRAMEWORK ..... 21
2.3 BUNIT ..... 23
**3 .NET** ..... **25**
3.1 .NET CORE ..... 25
3.1.1 .Net 5 ..... 25
3.1.2 .Net 6 ..... 26
3.2 ASP.NET ..... 27
3.2.1 Asp.Net Core ..... 27
3.3 BLAZOR ..... 27
3.3.1 Komponenty ..... 28
3.3.2 Modely hostingu ..... 28
**4 ENTITY FRAMEWORK** ..... **31**
4.1 ENTITY FRAMEWORK 6 A ENTITY FRAMEWORK CORE ..... 32
4.2 ENTITY DATA MODEL ..... 33
4.3 VYTVORENIE MODELU ..... 34
**PRAKTICKÁ ČÁST** ..... **35**
**5 WEBOVÁ STRÁNKA** ..... **36**
5.1 STRÁNKY ..... 36
5.1.1 Vyhľadávanie udalostí ..... 36
5.1.2 Vytvorenie udalosti ..... 37
5.1.3 Detail udalosti ..... 38
5.2 KOMPONENTY ..... 39
5.2.1 Atómy ..... 39
5.2.2 Molekuly ..... 42
5.2.3 Organizmy ..... 43

5.3 DATABÁZA ... 44
5.4 POUŽITÉ POSTUPY ... 46
5.4.1 Inversion of Control ... 46
5.4.2 Repository-Service Pattern ... 47
5.4.3 Atomický dizajn ... 49
5.5 KLONOVANIE REPOZITÁRA ... 50
**6 AUTOMATIZOVANÉ TESTY ... 51**
6.1 TESTY ... 51
6.1.1 EventButtonTests ... 51
6.1.1.1 JoinEventTest ... 51
6.1.1.2 LeaveEventTest ... 52
6.1.1.3 CannotJoinFinishedEventTest ... 53
6.1.1.4 OwnerCantLeaveTest ... 53
6.1.1.5 NotLoggedInTest ... 54
6.1.2 EventCardTests ... 54
6.1.2.1 MapIsSimpleMap ... 55
6.1.2.2 PeopleListExpandsOnClick ... 55
6.1.2.3 EventDetailsAreCorrect ... 55
6.1.3 EventFormTests ... 56
6.1.3.1 MapIsSetToPositionalMap ... 56
6.1.3.2 MinimumDateIsSetToToday ... 57
6.1.3.3 EventPropertiesMatchEventData ... 57
6.1.4 LocationFieldsTests ... 58
6.1.4.1 FieldsAreReadOnly ... 58
6.1.4.2 LatitudeValueChanges ... 58
6.1.4.3 LongitudeValueChanges ... 59
6.1.4.4 CityValueChanges ... 59
6.1.5 PlayerDialogTests ... 60
6.1.5.1 IsNotExpandedAfterCloseButtonClick ... 60
6.1.5.2 AllPlayersAreListed ... 60
6.2 POUŽITÉ POSTUPY ... 60
6.2.1 Používanie bUnit ... 61
6.2.2 Popis kódu testov ... 63
6.2.2.1 ContextBase ... 63
6.2.2.2 Fake Data ... 65
6.2.2.3 Testovacie sady ... 65
**ZÁVĚR ... 68**
**SEZNAM POUŽITÉ LITERATURY ... 69**
**SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK ... 71**
**SEZNAM OBRÁZKŮ ... 72**
**SEZNAM TABULEK ... 74**

# ÚVOD

Technologický pokrok v IT priemysle sa vyvíja veľmi rýchlo. Technológie vznikajú a zanikajú, a pri vývoji softvéru to platí možno aj o niečo viac. Webové stránky tiež už vystriedali nejeden spôsob ich tvorby. Čím je softvér komplexnejší, tým väčšia potreba vzniká overovaním, či je problém, ktorý softvér rieši správne riešený, a či sa program správa tak, ako by sa mal.

Kedysi bývali celé tímy testerov, ktorých veľkou časťou náplne práce bolo klikať myšou a ťukať do klávesnice v snahe objaviť nejakú chybu v softvéri. Popritom ako vývojári nasadzovali nové zmeny do tejto softvérovej aplikácie, mohli zároveň nevedomky zaniesť nové bugy do starých častí, ktoré pred tým fungovali správne. Testery teda netestovali iba nové funkcionality vyvíjaného softvéru, ale aj tie, ktoré pred tým boli funkčné.

Manuálne testovanie väčšej a komplexnejšej aplikácie bývalo veľmi časovo náročné, preto vznikla potreba automatizácie testovania aplikácii. Dnes už si môžeme vybrať spomedzi niekoľkých automatizačných knihovní, ktoré by bolo možné použiť pri tvorbe aplikácie, nie len iba tej webovej.

Táto záverečná práca je zameraná na tvorbu webovej aplikácie pomocou webového frameworku Blazor, a na relatívne novú testovaciu knižnicu webových stránok vytvorených pomocou technológie Blazor – frameworku bUnit.

Zároveň si popíšeme, aké postupy sme pri tvorbe webovej aplikácie a samotných testov použili, a ukážeme si aj riadky zdrojových kódov, ktoré sme pri riešeniach použili.

# I. TEORETICKÁ ČÁST

# 1 TESTOVANIE SOFTVÉRU

## 1.1 Manuálne testovanie softvéru

Hovorí sa o tom, že každý test, ktorý sa dá zautomatizovať by sa mal automatizovať. Prinesie to rast produktivity za menej ľudského času. Sú však prípady, kedy návratnosť investície nie je dostatočne vysoká, aby as oplatilo testovaciu sadu automatizovať. Takéto druhy testov je lepšie vykonávať manuálne. [1]

### 1.1.1 Exploratory testing

Automatizované testy sledujú presnú sadu inštrukcii pri vyhodnocovaní správania. Exploratory testing je oproti tomu viac náhodné testovanie, preto sa mu kedysi zvyklo hovoriť „ad-hoc testing“. Jedná sa o druh testovania, kedy tester skúsi rôzne náhodné alebo improvizované scenáre pri snahe objaviť chyby alebo nečakané správanie. [1]

### 1.1.2 Vizuálne regresné testy

Pri testovaní softvéru sa nemusia objaviť iba chyby v správaní. Môže sa jednať aj o zle umiestnené prvky užívateľského rozhrania, nevhodne zvolené písmo, ťažko čitateľné farby v texte a podobne. Existujú nástroje, ktoré testujú takéto chyby v softvérovom produkte. K tomuto ukladajú snímky obrazovky, a využívajú OCR k porovnávaniu s očakávanými výsledkami. Takéto testy bývajú nákladné pri ich tvorbe. Oveľa lacnejšie pritom môže byť, ak sa na softvérový produkt pozrie ľudské oko, v snahe nájsť vizuálne chyby. [1]

## 1.2 Automatizované testovanie softvéru

Automatizované testovanie softvéru je využitie softvérových nástrojov pre automatizovanie ľudského manuálneho procesu preskúšania a validácie softvérového produktu. Väčšina moderných agilných softvérových projektov využíva automatizované testovanie softvéru už od jeho začiatku. [1]

Pred tým, než sa začalo k testovaniu softvéru využívať automatizovaného testovania, softvérové spoločnosti najímali celý tím QA dohliadajúci na kvalitu produktu. Tento tím vytvoril kolekciu testovacích plánov, alebo zoznam postupov krok po kroku, v zmysle otestovania funkcionality vyvíjaného softvéru. Tím následne manuálne prešiel týmito postupmi zakaždým, keď vyšiel nový update, alebo bola vydaná zmena v softvérovom

projekte. QA tím následne tieto výsledky predstavil vývojárskemu tímu spolu s prípadnými problémami, ktoré našli. [1]

Tento proces bol pomalý, drahý, a náchylný na robenie chýb. Automatizované testovanie prináša tímovú efektivitu a návratnosť investície QA tímov. [1]

Automatizované by v prvom rade mali byť End-To-End testy, jednotkové testy a integračné testy. [1]

### 1.2.1 End-to-End (E2E) testy

Diskutabilne najviac hodnoty prinášajú End-to-End testy. End-to-End testy simulujú užívateľské zážitky naprieč celému full stack softvérového produktu. Tieto testy všeobecne pokrývajú user stories, ako napríklad „užívateľ sa môže prihlásiť“, alebo „užívateľ si môže zmeniť nastavenia emailu“. Tieto testy sú vysoko hodnotné lebo zabezpečujú, že skutočný užívateľ má hladký zážitok bez chýb, aj keď sú v softvéri nové zmeny. [1]

### 1.2.2 Integračné testy

Často sa stáva, že softvérový kód zavolá aj službu tretej strany. Testovaný kód tak nebude mať prístup ku kódu službe tretej strany. Integračné testy nahradzujú výsledky vrátené z tejto služby za falošné výsledky (mock), a porovnávajú, či kód, ktorý je s nimi spojený sa správa tak, ako by sa mal. Integračné testy môžu byť nedrahou alternatívou jednotkových testov, ku ktorým sú podobné spôsobom, akým sú napísané. Avšak vrátená investícia je diskutabilná v prípade, že už existuje kombinácia jednotkových a E2E testov. [1]

### 1.2.3 Jednotkové testy

Ako meno napovedá, jednotkové testy pokrývajú jednotlivé jednotky kódu. Jednotky kódu sú najlepšie zmerateľné v definíciách funkcii. Jednotkové testy tak pokryjú individuálnu funkciu. Tieto testy zisťujú, či so vstupom do funkcie nastane aj očakávaný výstup z funkcie. Kód, v ktorom prebiehajú citlivé výpočty, ako napríklad vo financiách, je najlepšie pokrytý jednotkovými testami. Jednotkové testy nie sú drahé, a naopak, sú rýchle na implementáciu a poskytujú veľkú hodnotu návratu investície. [1]

#### Odporúčané postupy

Existuje niekoľko dôvodov, prečo by sme mali písať jednotkové testy. Pomáhajú s regresiou, poskytujú dokumentáciu a pomáhajú pri dobrom dizajne. Avšak ťažko čitateľné

alebo krehké testy nám môžu veľmi znepríjemniť život. Microsoft na stránkach svojej dokumentácie uviedol charakteristiky dobrého jednotkového testu: [2]

- Dobre napísaný jednotkový test by mal byť rýchly. Dnes už nie je nezvyčajné, že väčšie projekty majú tisíce jednotkových testov. Tieto testy by preto malo bežať veľmi krátku dobu – v radoch milisekúnd.
- Izolovanosť jednotkových testov je ďalšou charakteristikou. Testy by nemali byť závislé na vonkajších faktoroch akými sú napríklad súborový systém alebo databáza.
- Mali by byť opakovateľné a konzistentné s výsledkami, ktoré poskytujú. Znamená to, že by mali vracať rovnaké výsledky aj keď ich spustíme viackrát, pokiaľ nič nezmeníme medzi jednotlivými spusteniami.
- Dobrý jednotkový test taktiež nepotrebuje kontrolu od človeka, ale sám spozná, či prešiel alebo zlyhal.
- Jeho tvorba by nemala príliš dlho trvať v porovnaní s dĺžkou kódu ktorý testuje. Ak by napísanie jednotkového testu trvalo príliš dlho, mali by sme sa zamyslieť nad niečím, čo je lepšie testovateľné.

Ďalším bodom je aranžovanie testov. Arrange, act, assert je bežný postup pri písaní jednotkových testov: [2]

- V prvom kroku arrange si vytvoríme a nastavíme objekty, s ktorými budeme v testoch pracovať
- V druhom kroku interagujeme s objektami – vpisujeme text do formulárov, klikáme na tlačidlá, a podobne
- V poslednom kroku porovnávame, či daný test splnil požiadavky – či výsledky z druhého kroku sú také, aké očakávame

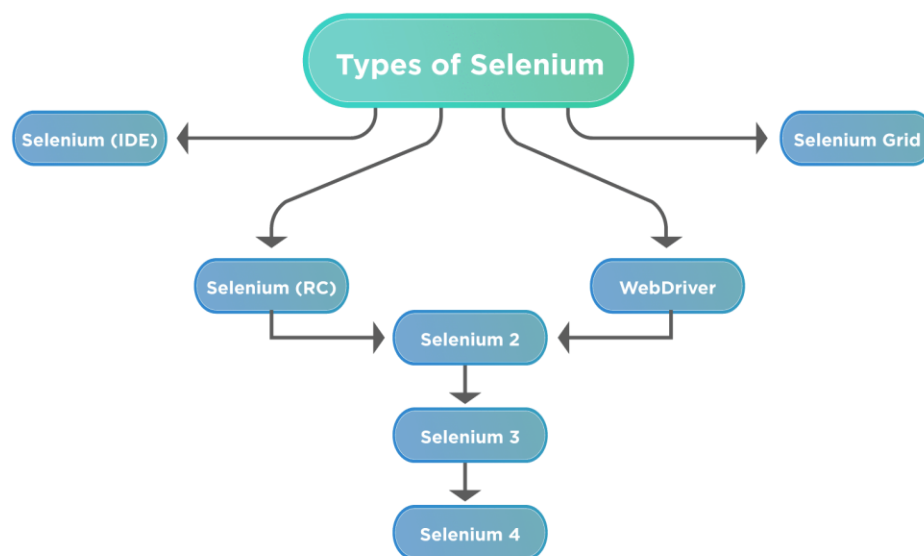
Rovnako by sme nemali do testov písať akúkoľvek logiku – mali by sme sa vyhnúť písaniu *if, while, for*, *switch* a tak ďalej. Bude tak menšia šanca, že sa v testoch nájde bug, a vďaka tomuto prístupu sa viac bude prizerať na koncový výsledok než na implementačné detaily. [2]

# 2 NÁSTROJE TESTOVANIA SOFTVÉRU

## 2.1 Selenium

Selenium nie je iba jeden nástroj, je to skupina nástrojov [3][4]. Každý nástroj iným spôsobom podporuje automatizované testovanie. Pozostáva zo 4 hlavných častí: [3][5]

1. Selenium Integrated Development Environment (IDE)
2. Selenium Remote Control (RC)
3. WebDriver
4. Selenium Grid




Obrázok 1 Nástroje Selenium [5]

Jedným z princípov tohto projektu je podporovať spoločné rozhranie pre všetky hlavné technológie prehliadačov. Webové prehliadače sú neuveriteľne komplexné aplikácie, ktoré vykonávajú operácie úplne inými spôsobmi, aj keď to často vyzerá rovnako. Aj keď sa v týchto prehliadačoch text renderuje rovnakým písmom, obrázky sú zobrazené na rovnakých miestach, a odkazy nás presmerujú na rovnaké miesto, rozdiely medzi tým čo sa deje vo vnútri sú veľmi rozdielne. Selenium abstrahuje tieto rozdiely, skrýva detaily a zložitosti osobe, ktorá píše kód. Toto umožňuje riadky komplexnejšieho kódu, ktoré pobežia úplne rovnako v prehliadači Firefox, Internet Explorer, Chrome, a všetkých ostatných podporovaných prehliadačoch. [4]

### 2.1.1 Selenium IDE

Selenium IDE je plugin, ktorý ponúka jedno rozhranie, ktoré nám umožňuje písať testovacie skripty v programovacích jazykoch napríklad C#, Java, Python, NodeJS a ďalšie. Ovládač prehliadača (driver) potom spúšťa tieto skripty v inštancii prehliadača na zariadení. Dokáže tak simulovať správanie užívateľa na webovej stránke. [6]

Tento nástroj ponúka výbornú cestu, ako sa naučiť syntax Selenium. Je dostupný pre prehliadače Google Chrome, Mozilla Firefox a Microsoft Edge. [4]

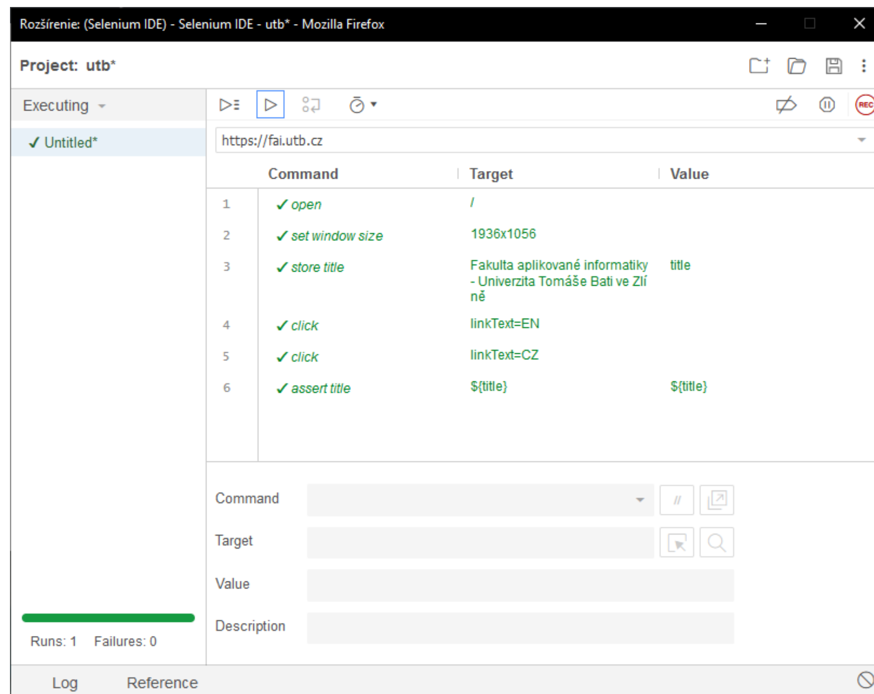
Je používaný na písanie testovacích prípadov. Ponúka grafické rozhranie pre nahrávanie akcii užívateľa, ktorý sa chce naučiť a používať Selenium. Zaznamenané akcie môžu byť prekonvertované do programovacích jazykov podporovaných Selenium, a tieto skripty môžu byť použité aj v iných prehliadačoch. [7]

Funguje to tak, že užívateľ navštívi webovú stránku, ktorú chce zaznamenávať, a spustí nahrávanie v Selenium IDE. Následne sú jeho akcie na stránke zaznamenávané. Po tom ako užívateľ dokončí svoj zamýšľaný test, môže si kroky uložiť. Tieto kroky je následne možné spustiť už zautomatizované. [7]

Ladenie je iba ďalšou funkcionalitou, ktorú nám tento nástroj ponúka. Môžeme určovať rýchlosť krokov, pozastaviť vykonávanie testu, alebo krokovať. [7]

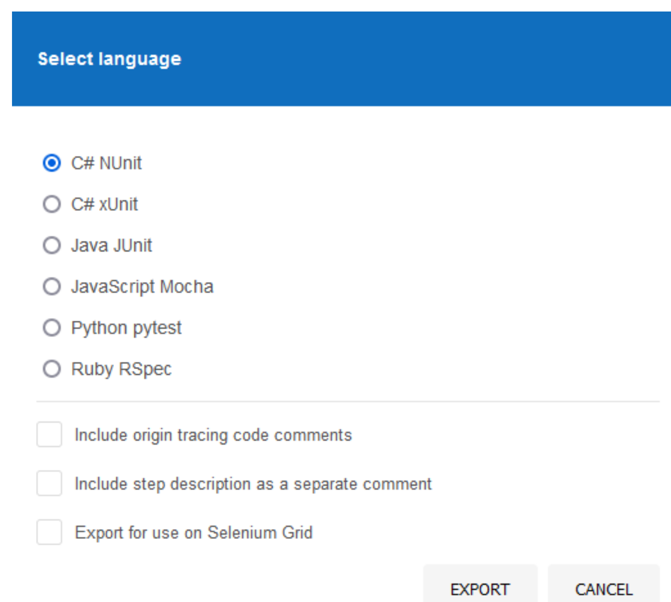
V rámci tohto nástroja je tiež možné vytvárať nielen jeden test, ale celú skupinu testov v testovacej sade. [7]

V rámci práce bol vytvorený test, ktorý navštívi stránku https://www.fai.utb.cz/. Nadpis stránky (tag title) sa mení v závislosti na zvolenom jazyku, takže v češtine a v angličtine je iný. Test si teda najprv v jazyku čeština uloží nadpis na stránke, zmení jazyk na angličtinu, následne zmení jazyk späť na češtinu a skontroluje, či uložený nadpis zodpovedá aktuálnemu nadpisu. Zaznamenané testovacie kroky pomocou pluginu Selenium IDE sa nachádzajú na obrázku 2 nižšie.




Obrázok 2 Selenium IDE

Ako môžeme vidieť, kroky sa uložili správne. Po otvorení sa okno spustilo v malom rozlíšení, preto bolo maximalizované. Aj táto akcia bola zaznamenaná v tomto nástroji (viď krok 2 v obrázku 2). Tieto kroky boli následne exportované do jazyku C# (obrázok 3 nižšie) pre testovací framework NUnit.




Obrázok 3 Výber jazyku a frameworku pri exporte zo Selenium IDE

Vygenerovaný kód bol aj vyskúšaný. Po vytvorení nového projektu s použitím frameworku NUnit, a po importovaní potrebných knižníc Selenium a GeckoDriver (WebDriver pre Firefox), bol vygenerovaný kód vložený do nového súboru projektu. Vygenerovaný kód mal však na riadku 44 v metóde Is.EqualTo parameter celý v reťazci znakov.

Namiesto toho, aby sa nadpis stránky porovnával s uloženým nadpisom získaného zo stránky, porovnával sa s textovým reťazcom „vars[“title”].ToString()”, a nie ako s kódom rovnakého znenia. Po odstránení nadbytočných úvodzoviek (obrázok 4 nižšie) sme boli schopný spustiť test. Spustil sa prehliadač Firefox, a vykonávalo sa presne to, čo bolo počas tvorenia testu zaznamenávané – na stránke sa preklikol jazyk a kontroloval sa tag title. Test po tejto kontrole úspešne prešiel.

```
1   // Generated by Selenium IDE
2   using System;
3   using System.Collections;
4   using System.Collections.Generic;
5   using System.Linq;
6   using System.Threading;
7   using OpenQA.Selenium;
8   using OpenQA.Selenium.Chrome;
9   using OpenQA.Selenium.Firefox;
10  using OpenQA.Selenium.Remote;
11  using OpenQA.Selenium.Support.UI;
12  using OpenQA.Selenium.Interactions;
13  using NUnit.Framework;
14
15  [TestFixture]
16  public class DefaultSuiteTest
17  {
18      private IWebDriver driver;
19      public IDictionary<string, object> vars { get; private set; }
20      private IJavaScriptExecutor js;
21
22      [SetUp]
23      public void SetUp()
24      {
25          driver = new FirefoxDriver();
26          js = (IJavaScriptExecutor)driver;
27          vars = new Dictionary<string, object>();
28      }
29
30      [TearDown]
31      protected void TearDown()
32      {
33          driver.Quit();
34      }
35
36      [Test]
37      public void untitled()
38      {
39          driver.Navigate().GoToUrl("https://fai.utb.cz/");
40          driver.Manage().Window.Size = new System.Drawing.Size(1936, 1056);
41          vars["title"] = driver.Title;
42          driver.FindElement(By.LinkText("EN")).Click();
43          driver.FindElement(By.LinkText("CZ")).Click();
44          Assert.That(driver.Title, Is.EqualTo(vars["title"].ToString()));
45      }
46  }
47
```

Obrázok 4 Exportované kroky do jazyku C# s frameworkom NUnit

Ako vidíme, Selenium IDE nám nielen umožní zaznamenávať testovacie kroky bez nutnosti manuálneho písania, ale dokážeme s ním aj tieto kroky exportovať do kódu v rôznych programovacích jazykoch, a využiť ich v nástrojoch Selenium RC alebo WebDriver. [3]

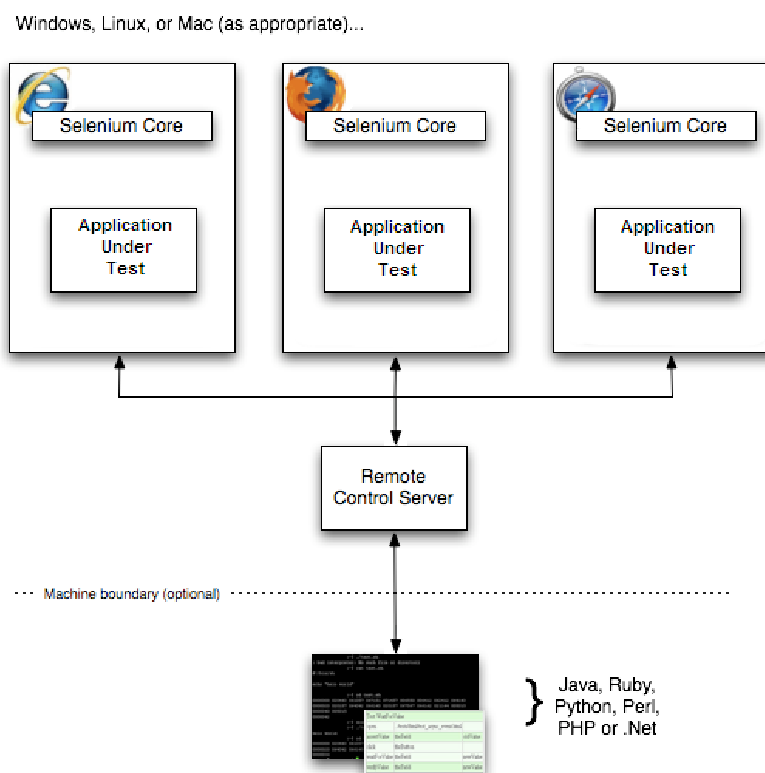
### 2.1.2 Selenium RC

Selenium RC je pôvodnou verziou (tiež známou ako Selenium 1). Selenium RC bol dlhú dobu hlavným Selenium projektom, pred tým než spojením WebDriver a Selenium RC vznikla novšia verzia – Selenium 2 [5][7]. Selenium RC však už nie je podporovaný [4], ale užívatelia stále môžu pomocou tohto nástroja vyvíjať skripty [7].

Selenium Server spúšťa a ukončuje prehliadače, interpretuje a spúšťa príkazy, ktoré k nemu odosiela testovací program. Zároveň sa správa ako HTTP proxy, zachytáva a overuje HTTP správy, ktoré si prehliadač a testovaná aplikácia (AUT) medzi sebou posielajú. [4]

Pre každý jazyk bola vytvorená nová klientska knižnica. Tieto knižnice nám v programovacom jazyku našej voľby poskytujú rozhranie so sadou funkcii medzi každým jazykom a serverom Selenium RC. Pomocou týchto funkcii knižnica posiela príkazy na Selenium Server, ktoré sú na ňom následne spracované. O výsledku spracovania nás knižnica informuje, tým, že nám ho pošle späť do programového kódu. [4]

Diagram nižšie na obrázku 5 ukazuje, ako klientske knižnice komunikujú so serverom.

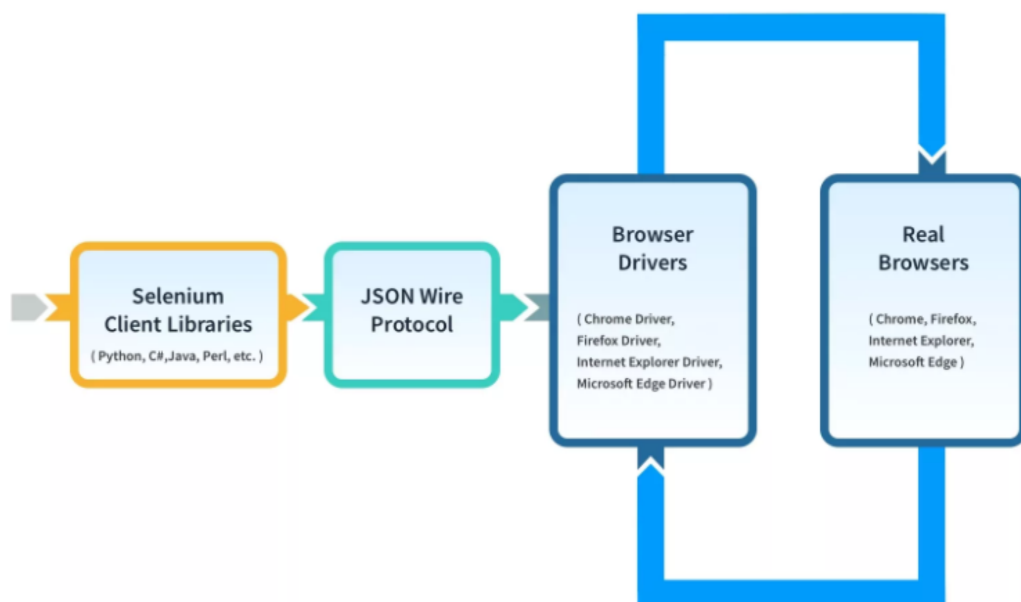



Obrázok 5 Architektúra Selenium RC [4]

### 2.1.3 Selenium WebDriver

Selenium WebDriver je najčastejšie používaný komponent Selenium. WebDriver používateľom umožňuje písať kód v jazyku, ktorý si využívatelia vyberú, a umožní im interagovať s prehliadačom podľa ich voľby pomocou ovládačov špecifických pre prehliadač. WebDriver pracuje na úrovni operačného systému. [5]

Klientske knižnice komunikujú s ovládačmi prehliadača pomocou protokolu Json Wire Protocol [5][8]. Selenium poskytuje ovládače špecifické zvlášť pre každý podporovaný prehliadač. Tieto ovládače interagujú s ich príslušným prehliadačom vytvorením zabezpečeného spojenia. Ovládače prehliadačov sú tiež špecifické pre jazyk, ktorý je použitý pre automatizovanie testového prípadu [8].




Obrázok 6 Architektúra WebDriveru [8]

Selenium 3 bolo vylepšenie viacerými spôsobmi. Verzia Selenium 3 sa stala W3C štandardom. Selenium RC bol presunutý do legacy balíčku, a bolo implementovaných viacero vylepšení spolu s novými funkciami a vlastnosťami. [5]

Selenium 4 je najnovšou verziou a pre koncových užívateľov je stále v beta verzii. [5]

**Selenium Locators**

Na to aby WebDriver dokázal interagovať s elementami na stránke tieto elementy musí nájsť. Parametrom do funkcie *findElement* povieme ovládači, ako element nájde. Existuje veľa spôsobov, podľa čoho ho nájsť: [7]

1. Podľa ID (By.id)
2. Podľa mena (By.name)
3. Podľa mena triedy (By.className)
4. Podľa mena (HTML) tagu (By.tagName)
5. Podľa textu v odkaze (By.linkText)
6. Podľa čiastočného textu v odkaze (By.partialLinkText)
7. Podľa CSS (By.cssSelector)
8. Podľa XPath (By.xpath)

Majme *input* element s menom *cdensity*. Riadok nižšie WebDriveru povie, že má nájsť element, ktorého atribút *name* má hodnotu *cdensity.* Keď ho nájde, môžeme mu posielať príkazy. V riadku nižšie bol v prvej časti element nájdený pomocou funkcie *findElement* [7], a následne doňho bolo napísané číslo 10:

```
driver.findElement(By.name("cdensity")).sendKeys("10");
```

Takýmto spôsobov môžeme vyhľadať elementy na stránke a interagovať s nimi pomocou rôznych príkazov.

**Actions API**

Actions API je nízko-úrovňové rozhranie, ktoré poskytuje virtualizovaný vstup zariadenia do webového prehliadača. Narozdiel od vysoko-úrovňových interakcii s elementami, v ktorých prebiehajú dodatočné validácie, Actions API poskytuje určitú kontrolu nad vstupnými zariadeniami. Selenium poskytuje prístup vstupných zdrojov: vstupy kláves pre klávesnice, ukazovateľ pre myš, pero alebo dotykové zariadenie a vstup skrolovacieho kolieska (scroll wheel). [4]

Do Actions API patria napríklad akcie: [4]

- Klávesnice
    - Key down – simuluje stlačenie klávesy
    - Key up – simuluje uvoľnenie klávesy

    - Send keys – konvenčná metóda Actions API, ktorá kombinuje key down a key up príkazy v jednej akcii
- Myši
    - Click and hold – presunie kurzor do stredu elementu a klikne bez uvoľnenia
    - Context click – presunie kurzor na element a vykoná stisk pravého tlačidla myši
    - Drag and drop – klikne s držaním na zdrojoví element, presunie sa na lokáciu cieľového elementu a následne uvoľní myš
- Skrolovacieho kolieska

**Interakcie s elementami**

Jedná sa o vysoko-úrovňovú inštrukčnú sadu pre manipuláciu ovládacích prvkov. Tieto metódy sú navrhnuté tak, aby bližšie napodobňovali užívateľský zážitok. Oproti Actions API sa pokúsi o 2 kroky pred tým, než vykoná danú akciu: [4]

1. Ak je vyhodnotené, že daný element sa nachádza mimo viditeľnej časti stránky, tak scrolluje element do viditeľnej časti.
2. V druhom kroku sa uistí, že s daným elementom sa dá interagovať pred tým, než vykoná akciu. Toto môže znamenať, že rolovanie bolo neúspešné, alebo že element nie je na stránke zobrazený

Do tejto inštrukčnej sady patria napríklad príkazy ako *click*, ktorý klikne uprostred elementu, alebo príkaz *send keys*, ktorý pošle vstup do textových polí alebo iných ovládacích prvkov, v ktorých je možné upravovať ich obsah. [4]

### 2.1.4 Selenium Grid

Selenium Grid umožňuje spustenie skriptov WebDriveru na vzdialených zariadeniach, či už virtuálnych alebo skutočných, a to smerovaním príkazov odoslaných klientom na vzdialené inštancie prehliadača. [4]

Zároveň dokáže spravovať verzie prehliadačov a ich konfigurácie centrálne, namiesto toho, aby sme ich museli spravovať pre každý test zvlášť [4].

Za cieľ si dáva jednoduchý spôsob spúšťania testov paralelne na viacerých strojoch [4][7]. Toto značne urýchľuje testovací proces naprieč prehliadačmi a platformami, a dáva nám to rýchlu a presnú spätnú väzbu. [7]

## 2.2 Robot Framework

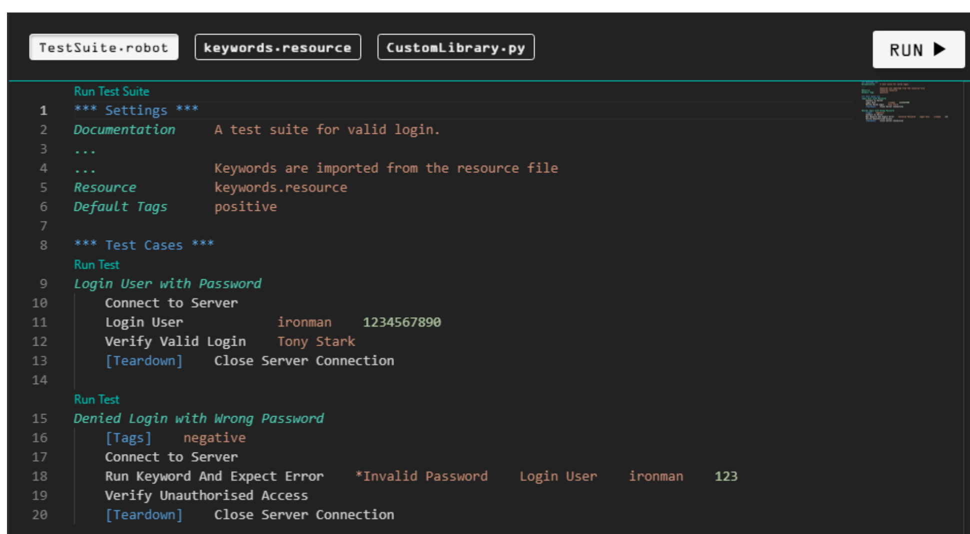
Robot Framework je generický open-sourcový automatizovaný framework. Môže byť použitý pre testovú automatizáciu alebo robotickú automatizáciu procesov, a to úplne zadarmo bez licenčných poplatkov. [9]

Robot Framework má jednoduchú syntax, keďže využíva ľuďmi známe slová. Jeho schopnosti môžu byť rozšírené knižnicami implementovanými v programovacom jazyku Python, Java, alebo mnoho ďalších. Pozostáva z knihovní a nástrojov, ktoré sú vyvíjané ako samostatné projekty. [9]

Je otvorený a rozšíriteľný, a môže byť integrovaný prakticky skoro s každým iným nástrojom pre vytvorenie silných a flexibilných automatizačných riešení. Podporuje ho obrovská komunita ľudí, ktorý prispievajú k jeho rozširovaniu. [9]

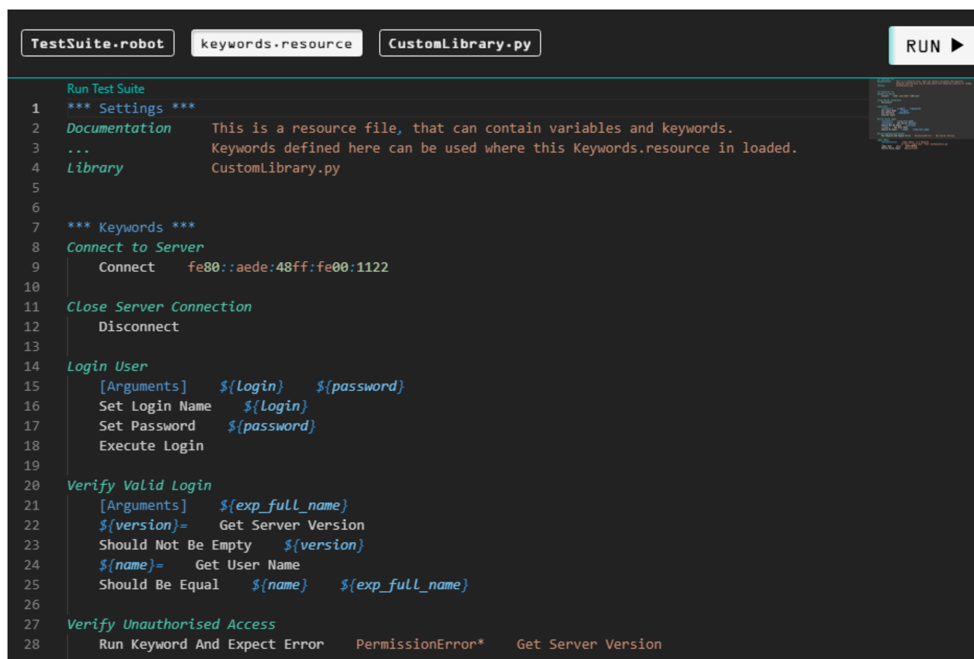
V závislosti od potrieb užívateľa je tu možnosť nainštalovať si externé knižnice. Mimo iné tu nájdeme knižnicu SeleniumLibrary, pre testovanie webových aplikácii s využitím populárnej knižnice Selenium. [9]

V kóde nižšie si všimneme skutočne jednoduchej syntaxe. Na obrázku je testovacia sada, ktorá pozostáva z 2 testovacích prípadov. Všimneme si tiež, že príkazy v testoch obsahujú kľúčové slová, ktoré sú priamou interpretáciou toho, čo človek od stroja vyžaduje. Za príkazmi sú medzerou oddelené argumenty do funkcie.




Obrázok 7 Ukážkový kód v Robot Framework [9]

Na obrázku 8 nižšie sú definované kľúčové slová, ktoré boli použité v obrázku 7, avšak na nižšej úrovni. Ak by sme išli ešte o úroveň nižšie, a pozreli by sme sa na funkcie zo súboru *CustomLibrary.py*, ktoré sú tu použité, našli by sme už programoví kód v jazyku Python. [9]




Obrázok 8 Definícia kľúčových slov [9]

Výstupom z vykonaných testov je správa o prehľade testov v súborovom formáte HTML v podobe reportu. Tieto súbory obsahujú štatistiky, ktoré vznikli na základe tagov a spustených testovacích sadách a testovacích prípadoch samotných. Pre detailnejšie informácie sú generované aj logy, ktoré sú priamo prepojené odkazom vygenerovaným v reporte. Keď všetky spustené testy sa úspešne dokončia, pozadie reportov bude zelené. Ak by niektorý neprešiel, respektíve prešiel neúspešne, pozadie reportu bude červené. Pozadie však môže byť aj žlté – čo by znamenalo, že testy boli preskočené. O tom, ktorý test prešiel, a ktorý neprešiel, si môžeme po dokončení testu overiť aj v príkazovom riadku. Výpis v príkazovom riadku však nie je taký detailný, a obsahuje iba informácie o tom, či testovacia sada a testovací prípad prebehli úspešne alebo neúspešne. [9]

## 2.3 bUnit

BUnit je testovacia knižnica pre testovanie Blazor komponenty. Za cieľ si dáva, aby bolo jednoduché písať zrozumiteľné a stabilné jednotkové testy. Pomocou bUnit užívateľ dokáže: [10]

- Nastaviť a definovať testovacie komponenty pomocou syntaxe jazyka C# alebo syntaxe Razor
- Overiť výsledky pomocou sémantického HTML komparátora
- Interagovať a pozorovať komponenty, rovnako ako aj spúšťať udalosti
- Injectovať služby do testovanej komponenty
- Mockovať autentizáciu a autorizáciu Blazoru, a podobne

BUnit je založený na už existujúcich frameworkoch, ktoré sú zamerané na jednotkové testy, ako napríklad xUnit, NUnit a MSTest, ktoré spúšťajú testy Blazor komponenty úplne rovnako, ako aj iné bežné jednotkové testy. Dĺžka testu za použitia bUnit je v rámci milisekúnd v porovnaní s UI testami, ktoré bežia v prehliadači, ktorých dĺžka testu býva meraná v sekundách. [10]

Testovanie Blazor komponent je trochu odlišné od testovania regulárnych tried jazyka C#. Blazor komponenty sú vykreslené (*rendered)*, a majú životný cyklus Blazor komponenty, do ktorej môžeme poslať vstup, a ktorá nám zároveň môže vrátiť výstup. [10]

Vykresľovanie sa vykonáva skrz testovací kontext frameworku bUnit. Výsledkom vykreslenia je *IRenderedComponent*, čo je tiež označované ako vykreslený komponent, ktorý poskytuje prístup k inštancii komponenty a k značeniu (markup) vo vnútri komponenty. [10]

Majme komponent *Counter*, ktorý zvyšuje hodnotu čísla o 1 s každým stlačením tlačidla. Kód komponenta je na obrázku 9 nižšie.

```
<h1>Counter</h1>

<p>
    Current count: @currentCount
</p>

<button class="btn btn-primary" @onclick="IncrementCount">Click me</button>

@code {
  int currentCount = 0;

  void IncrementCount()
  {
    currentCount++;
  }
}
```

Obrázok 9 Komponent Counter [10]

Spočiatku je hodnota zvyšovaného čísla 0 a po kliknutí na tlačidlo sa hodnota zvýši o 1. To znamená, že po prvom kliknutí bude hodnota 1. Test, pokrývajúci túto funkcionalitu by mohol vyzerať nasledovne: [10]

```
[Fact]
public void CounterShouldIncrementWhenClicked()
{
  // Arrange: render the Counter.razor component
  using var ctx = new TestContext();
  var cut = ctx.RenderComponent<Counter>();

  // Act: find and click the <button> element to increment
  // the counter in the <p> element
  cut.Find("button").Click();

  // Assert: first find the <p> element, then verify its content
  cut.Find("p").MarkupMatches("<p>Current count: 1</p>");
}
```

Obrázok 10 Test komponent Counter [10]

Tento test využíva TestContext frameworku bUnit pre vykreslenie komponentu *Counter* pomocou metódy *RenderComponent*. Následne pomocou metódy *Find* nájde tlačidlo, a klikne pomocou *Click.* Na záver si test nájde element *paragraph*, ktorý vypisuje mimo iné aj hodnotu čísla, a v poslednom kroku porovná, či sa hodnota zvýšila na 1. [10]

# 3 .NET

.Net (čítame Dot Net) je platforma pre vývojárov, v ktorej sú obsiahnuté nástroje, knižnice a programovacie jazyky pre tvorbu rôznych typov aplikácii. Existujú rozličné implementácie, a každá implementácia umožňuje spustiť kód na rôznych platformách – Windows, Linux, macOS, iOS, Android atď. [11]

## 3.1 .Net Core

Dnes už je bežné, že vyvíjané aplikácie sa používajú z rôznych typov zariadení. Vzniká teda potreba frameworku, ktorý by dokázal bežať vo všetkých využívaných zariadeniach. Na toto Microsoft myslel, keď vytváral framework .Net Core. [12]

.Net Core je novšia verzia .Net Framework. Je to voľne dostupný a otvorený (open-source) softvérový crossplatformový framework vyvíjaný spoločnosťou Microsoft, ktorý beží na operačných systémoch Windows, Linux a macOs. Je využiteľný pre rôzne typy aplikácii, akými sú napríklad desktopové, mobilné, webové aplikácie, ale aj cloudové, IoT, herné aplikácie atď. [12]

Kód napísaný v .Net Core sa správa rovnako v odlišných architektúrach – nezáleží, či architektúra inštrukčnej sady zariadenia je x64, x86 alebo ARM.[12]

Podporuje viacero programovacích jazykov – vývojári si môžu vybrať medzi programovacími jazykmi C#, F# alebo Visual Basic. [12]

### 3.1.1 .Net 5

.Net 5 je nástupnou veľkou verziou .Net Core 3.1. Táto verzia preskočila verziu .Net Core 4, aby sa vyhlo zameneniu tohto názvu s verziou .Net Framework 4.x. Takisto sa už v názve nepoužíva „Core“, aby bolo zrejmé, že sa jedná o hlavnú implementáciu .Net, ktorá sa bude ďalej rozširovať. [2]

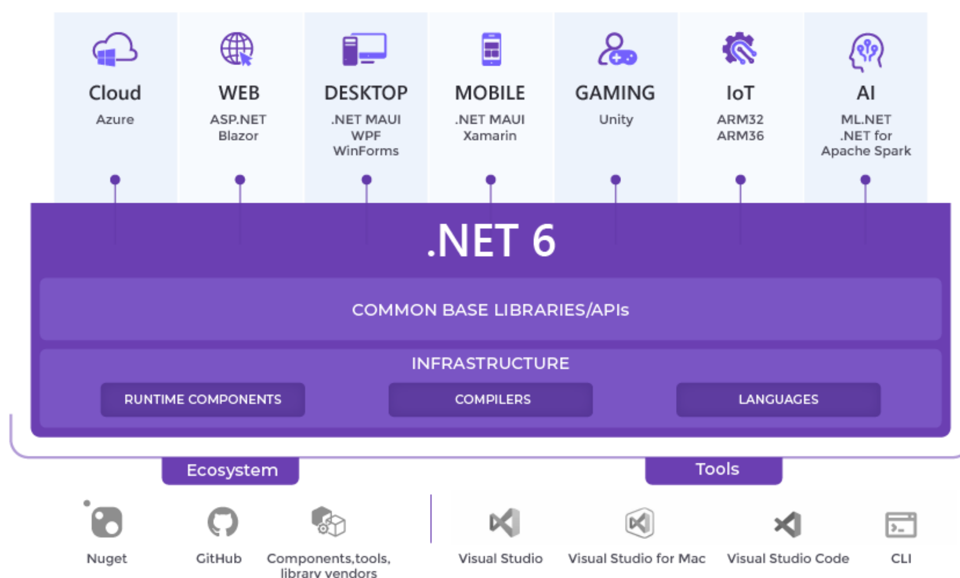
.Net 5 podporuje viac typov aplikácii a platforiem než .Net Core alebo .Net Framework. [2]

Obrázok 11 Jednotná platforma .Net 5 [13]

### 3.1.2 .Net 6

.Net 6 spája časti .Net zjednocovacieho plánu, ktorý začal vo verzii .Net 5. Verzia .Net 6 unifikuje SDK, Base Class Library (BCL) a beh programu naprieč desktopovými, mobilnými, IoT a cloudovými aplikáciami. [2]

.Net 6 so sebou prináša aj nové vlastnosti jazyka C# 10, ktoré redukujú kód potrebný pri vývoji, ale prináša aj celkový lepší výkon behu programu. [2][14]




Obrázok 12 Jednotná platforma .Net 6 [14]

## 3.2 Asp.Net

Asp.Net je predchodca Asp.net Core. Pre všetky nové aplikácie by sa mal využívať novší Asp.Net Core. Asp.Net totiž podporuje iba aplikácie pre operačný systém Windows. [11]

Platforma .Net poskytuje komponenty, ktoré sú znovu-využiteľné v rôznych typoch aplikáciách. Frameworky, ako napríklad Asp.Net rozširujú tieto komponenty. Asp.Net teda rozširuje .Net o nástroje a knižnice, špecifické pre tvorbu webových aplikácii, napríklad: [11]

- Základný framework pre spracovanie webových požiadaviek
- Šablónu pre tvorenie dynamických webových stránok za použitia jazyku C#, známu tiež ako Razor
- Knižnice pre bežné webové vzory, ako napríklad Model View Controller (MVC)
- Autentizačný systém, ktorý zahŕňa knižnice, databáze a vzorové stránky pre používanie prihlásení, vrátane viac faktorového prihlásenia a externej autentizácie pomocou služby Google, Twitter a tak ďalej

**Razor**

Razor poskytuje syntax pre vytváranie dynamických webových stránok za použitia HTML a C#. Kód napísaný v jazyku C# je na serveri vyhodnotený a výsledný obsah v HTML je odoslaný späť k používateľovi. Kód, ktorý sa spúšťa na strane klienta je napísaný v jazyku JavaScript. Asp.Net integruje JavaScriptové frameworky a zahŕňa predkonfigurované šablóny pre Single Page Application (SPA) frameworky akými sú napríklad React a Angular. [11]

### 3.2.1 Asp.Net Core

Asp.net Core je open-sourcový framework pre tvorbu webových aplikácii a služieb s Dot Netom od spoločnosti Microsoft. Je cross-platformový a beží na operačných systémoch Windows, Linux, macOs a službe Docker. [11]

## 3.3 Blazor

Blazor je framework pre tvorbu interaktívnych webových aplikácii za použitia frameworku .Net, ktorý umožňuje: [2]

- Zdieľať aplikačnú logiku medzi stranami servera a klienta, ktorá je písaná v jazyku C#

- Vykresľovať užívateľské rozhranie ako HTML a CSS pre podporu širokého množstva prehliadačov, a to aj vrátane mobilných prehliadačov
- Využívať existujúci ekosystém .Net knižníc
- Tvoriť hybridné desktopové a mobilné aplikácie

### 3.3.1 Komponenty

Aplikácie tvorené frameworkom Blazor sú založené na komponentoch. Komponent je určitý element užívateľského rozhrania, ako je napríklad webová stránka, dialóg, alebo vstupný formulár. Komponenty sú .Net triedy v jazyku C#, ktoré: [2]

- Definujú flexibilnú logiku užívateľského rozhrania
- Dokážu reagovať na užívateľské udalosti
- Môžu byť vnorené do seba a znovu použiteľné na viacerých miestach
- Dokážeme zdieľať a distribuovať ako Razor knižnice, alebo ako NuGet balíček

Trieda komponentu je zvyčajne písaná vo forme Razor stránky so súborovou .razor príponou. Komponenty frameworku Blazor nazývame formálne ako Razor komponenty a neformálne ako Blazor komponenty. [2]

Razor syntax nám umožňuje kombinovať HTML značenie s kódom v jazyku C#, a prepínať sa medzi týmito jazykmi v jednom súbore. Razor Pages a MVC tiež používajú Razor. Avšak na rozdiel od Razor Pages a MVC, ktoré sú budované okolo modelu požiadavky a odpovede, komponenty sú použité konkrétne pre užívateľské rozhranie a kompozíciu na strane klienta. [2]

### 3.3.2 Modely hostingu

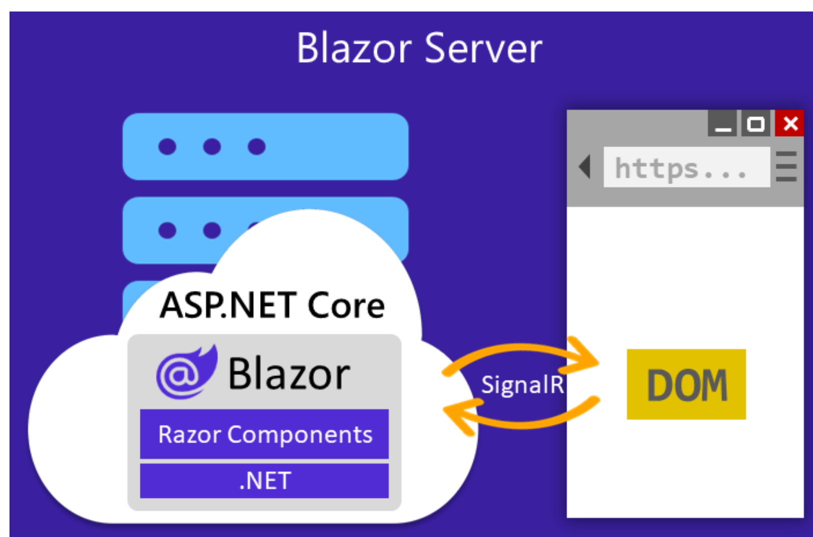
Komponenty Razor môžu bežať na strane servera v ASP.NET Core (Blazor Server), a na strane klienta v prehliadači vo WebAssembly založenom na .NET Runtime. Možný je tiež hosting Razor komponent v natívnych mobilných a desktopových aplikácii, ktoré sú vykresľované do Embedded Web View Control – Blazor Hybrid. Tvorenie Razor komponent je vo všetkých modeloch rovnaké. Rovnaké komponenty môžu byť zároveň použité vo viacerých modeloch bez zmeny. [2]

**Blazor Server**

V tomto modeli je aplikácia spustená na servery zvnútra aplikácie ASP.NET Core. Obnovy užívateľského rozhrania, spracovanie udalostí a volania z jazyka JavaScript sú spracovávané cez spojenie SignalR. [2]

Na strane klienta Blazor skript nadviaže SignalR pripojenie so serverom. Klientska strana aplikácie je potom zodpovedná za udržovanie a obnovovanie stavu aplikácie podľa požiadaviek. [2]

Keď je Razor Page alebo view renderovaný, každý riadok Razor kódu vydáva HTML kód v textovej podobe. Po renderovaní sa server zbaví inštancie stránky alebo view, vrátane stavu, aký bol produkovaný. Keď je následne odoslaná ďalšia požiadavka na stránke, celá stránka je znovu renderovaná do HTML a odoslaná klientovi. [2]




Obrázok 13 Blazor Server [2]

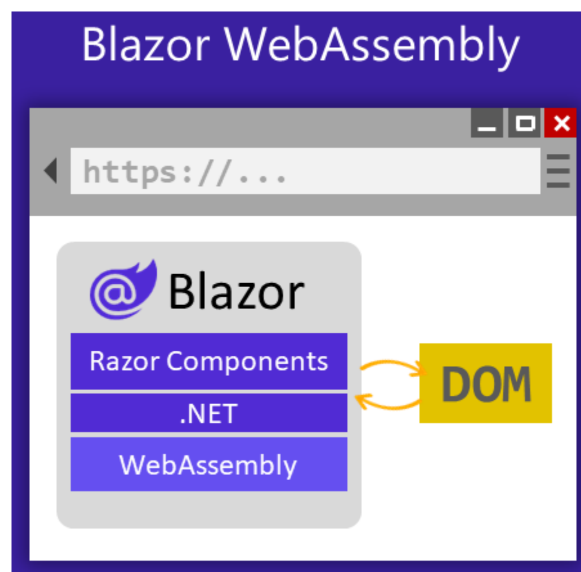
**Blazor WebAssemby**

Aplikácia Blazor WebAssembly (WASM) beží na strane klienta vo WebAssembly prehliadača, založenom na .Net runtime. Blazor aplikácia a jej závislosti, spolu s .Net runtime sú stiahnuté do prehliadača. Aplikácia je spustená priamo v prehliadačovom UI vlákne. [2]

Obnovy užívateľského rozhrania a spracovanie udalostí sa vykonáva v rámci toho istého procesu. Aplikačné assets sú nasadené ako statické súbory na webový server, alebo do služby schopnej podávania statického obsahu svojim klientom. [2]

Veľkosť aplikácie – *payload size*, je kritickým faktorom pre jej výkon a využiteľnosť. Príliš veľká aplikácia zaberie relatívne veľa času, aby sa stiahla do prehliadača, čo zhoršuje užívateľský zážitok. Blazor WebAssembly optimalizuje *payload size* aby znížila dobu sťahovania tak, že: [2]

- Nepoužitý kód je vyhodený z aplikácie po tom, čo je aplikácia uverejnená cez Intermediate Language Trimmer
- Odpovede HTTP sú kompresované
- .Net Runtime a assemblies sú cachované v prehliadači




Obrázok 14 Blazor WebAssembly [2]

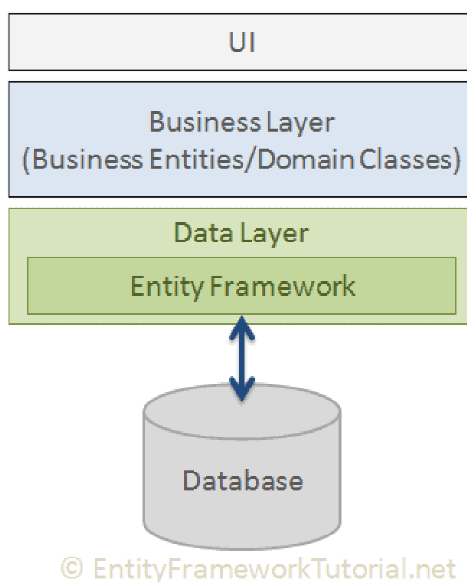
# 4 ENTITY FRAMEWORK

Pred .Net 3.5 vývojári často zvykli písať ADO.NET kód aby uložili alebo získali dáta z databáze. Zvykom bolo nadviazať spojenie s databázou, a vytvoriť dataset na stiahnutie alebo potvrdenie dát, a následne dáta v tomto datasete konvertovať na objekty .NET (alebo naopak) a aplikovať pravidlá na úrovni biznisu. Toto bol veľmi zaťažujúci proces náchylný na chyby. Microsoft preto poskytol framework nazvaný Entity Framework, aby zautomatizoval aktivity vzťahujúce sa na databázu pre našu vyvíjanú aplikáciu. [15]

Entity Framework je open-sourcový ORM framework pre .NET aplikácie podporované Microsoftom. Vývojárom umožňuje pracovať s dátami za použitia tried doménovej špecifikácie bez nutnosti sústredenia sa na databázové tabuľky a stĺpce, kde sú uložené dáta. S Entity Frameworkom dokážu vývojári pracovať na vyššom leveli abstrakcie keď sa zaoberajú s dátami, a môžu vytvárať a udržiavať dátami-orientované aplikácie s menším počtom riadkov kódu v porovnaní s tradičnými aplikáciami. [2] [15]

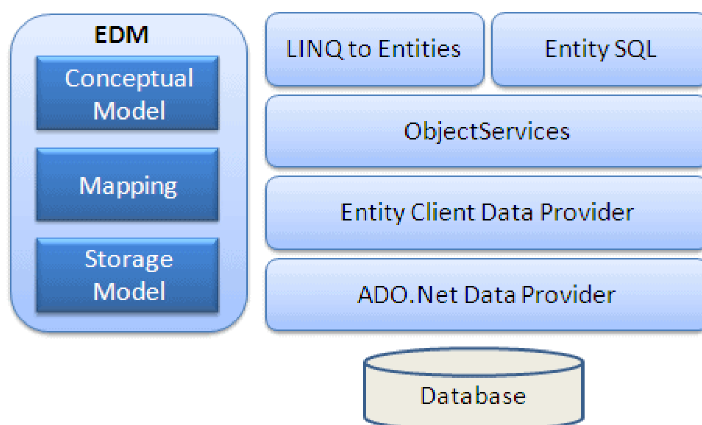
Oficiálna definícia znie [2] [15]: Entity Framework je objektovo-relačný mapper, ktorý umožňuje .NET vývojárom pracovať s databázou a používať pri tom objekty .NET. Eliminuje potrebu väčšiny kódu, ktorý pristupuje k priamo k dátam, ktorý vývojári zvyknú písať.

Na obrázku nižšie si môžeme všimnúť, na ktorej vrstve Entity Framework operuje popri komunikovaní s databázou a biznisovými modelmi aplikácie.




Obrázok 15 Vrstvy modelov tried [15]

Na obrázku 16 nižšie je zobrazená všeobecná architektúra Entity Frameworku.




Obrázok 16 Architektúra Entity Frameworku [15]

## 4.1 Entity Framework 6 a Entity Framework Core

Microsoft predstavil Entity Framework v roku 2008 spolu s .NET Frameworkom 3.5. Odvtedy bolo uvoľnených veľa verzií Entity Frameworku. Momentálne sú tu dve najnovšie verzie: Entity Framework 6 a Entity Framework Core. [15]

Entity Framework 6 je navrhnutý pre .Net Framework s podporou pre .Net Core. EF6 je stabilný a podporovaný produkt, už sa však ale aktívne nevyvíja. [2]

Tabuľka č.1 nižšie ukazuje dôležité rozdiely medzi týmito verziami.

| Entity Framework | Entity Framework Core |
| --- | --- |
| Prvý release v roku 2008 spolu s .NET Framework 3.5 SP1 | Prvý release v júny v roku 2016 spolu s .NET Core 1.0 |
| Stabilný s veľa vlastnosťami | Nový a stále vyvíjajúci sa |
| Podpora iba pre Windows | Podpora pre Windows, Linux a OSX |
| Funguje na .NET Frameworku 3.5+ | Funguje na .Net Frameworku 4.5+ a .NET Core |
| Open-source | Open-source |

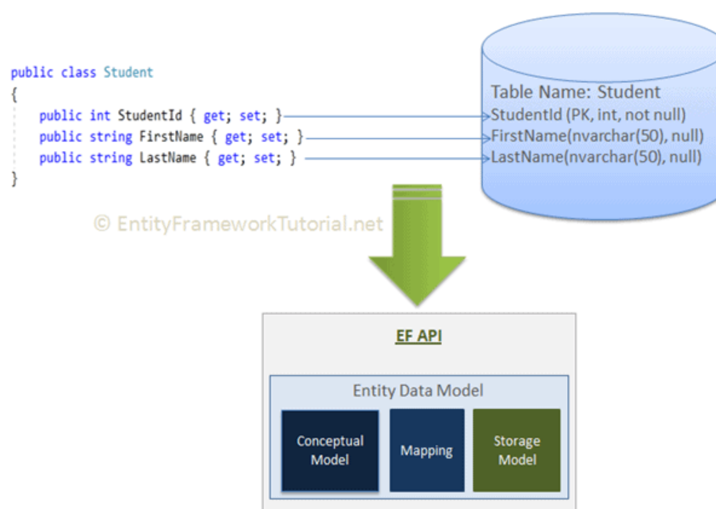
Tabuľka 1 Rozdiely medzi Entity Framework 6 a Entity Framework Core [2][15]

## 4.2 Entity Data Model

Prvou úlohou Entity Framework API je zostaviť Entity Data Model (EDM). Tento model je reprezentáciou všetkých metadát v pamäti conceptual modelu, storage modelu a mapovaním medzi nimi. [2] [15]

Model EDM teda popisuje entity a vzťahy medzi nimi. Entitou sa myslí trieda alebo objekt, ktorý reprezentuje aplikačné dáta, akými sú napríklad zákazníci, produkty, alebo objednávky. [2]

EF vykonáva CRUD operácie za použitia EDM, ktorý využíva k zostaveniu SQL dotazov z LINQ dotazov, príkazov INSERT, UPDATE, a DELETE, a transformáciu databázového výsledku na objekty entity. [15]




Obrázok 17 Dátový model entity [15]

**Conceptual model**

Entity Framework tvorí tento model z doménových tried, kontextovej triedy, predvolených konvencií v doménových triedach a z konfigurácií. [15]

**Storage model**

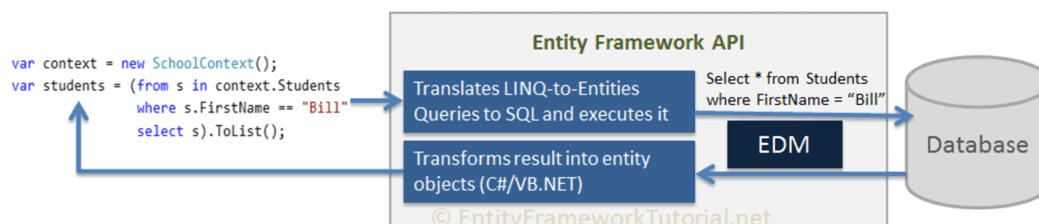
Storage model je zostavený Entity Frameworkom pre databázové schéma. V prípade code-first dizajnu bude posudzovaný z konceptuálneho modelu. V prípade database-first dizajnu bude pamäťový model posúdený z cieľovej databáze. [15]

**Mapovanie**

Entity Framework zahŕňa mapovaciu informáciu o tom, ako sa má conceptual model mapovať na databázové schéma (storage model). [15]

**Querying**

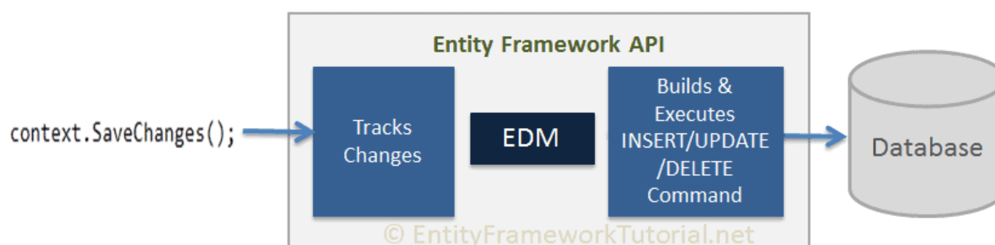
EF API prekladá LINQ-na-Entity dotazy na SQL dotazy pre relačné databáze a výsledky konvertuje späť na objekty entity, a to za použitia EDM. [2]




Obrázok 18 LINQ na Entity dotaz a spätná transformácia výsledku [15]

**Ukladanie**

EF API rozumie príkazom INSERT, UPDATE a DELETE na základe stavu entít v čase, keď bola zavolaná metóda *SaveChanges()*. ChangeTrack uchováva a sleduje stavy každej entity pred a po každej akcii. [2] [15]




Obrázok 19 Ukladanie zmien kontextu v EF API [15]

## 4.3 Vytvorenie modelu

Existujú dve možnosti ako vytvoriť model EF: [2]

1. Code-First – vývojár napíše kód, aby špecifikoval aký model chce vytvoriť. EF následne generuje modely na základe toho, ako boli napísané triedy entít a dodatočný konfiguračný model poskytnutý vývojárom
2. Database-First – vývojár špecifikuje model tak, že ho nakreslí pomocou rozšírenia EF Designer. Výsledný model je uložený ako XML v súbore s príponou EDMX. Jedná sa teda o postup, kedy je databáza vytvorená ako prvá, a Entity Framework tak už cieli na predom vytvorenú databázu, ku ktorej sa následne pripája.

# II. PRAKTICKÁ ČÁST

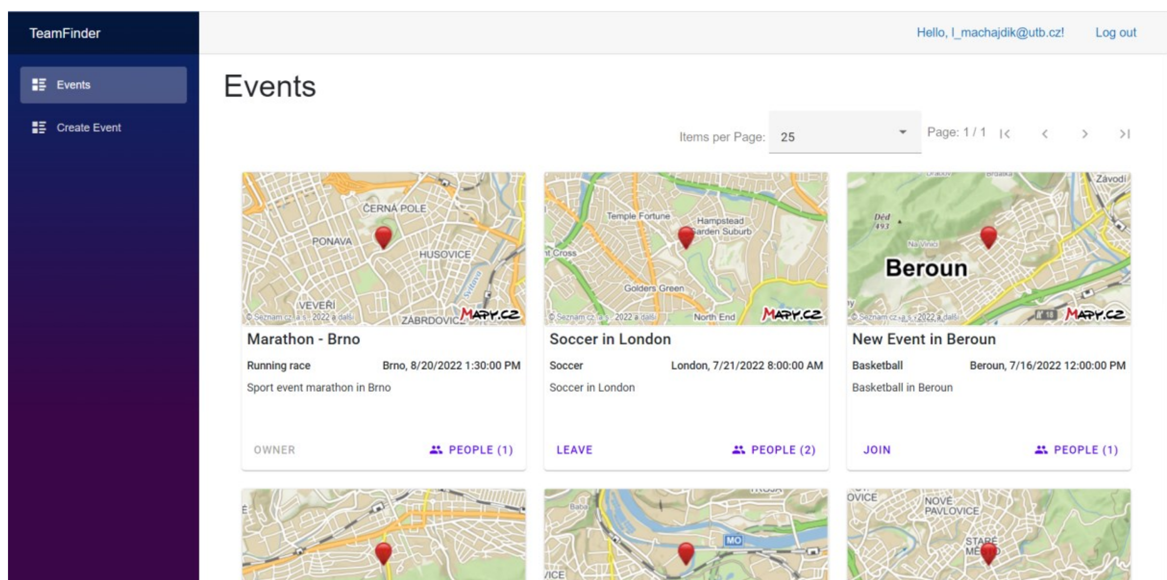
# 5 WEBOVÁ STRÁNKA

Vytvorená webová stránka je zameraná predovšetkým na ľudí, ktorý si chcú zašportovať, no nemajú s kým. Užívateľ sa môže zaregistrovať na tejto webovej stránke, a po prihlásení môže vytvoriť udalosť. Ostatný užívatelia sa k tejto udalosti môžu pripojiť a následne sa fyzicky stretnúť v čase, ktorý organizátor zadal, na mieste výberu konania udalosti. Stránka tiež umožňuje vyhľadávanie takýchto udalostí.

## 5.1 Stránky

### 5.1.1 Vyhľadávanie udalostí

Hlavnou stránkou aplikácie, je stránka so zobrazením všetkých udalostí. Užívatelia tu uvidia výpis všetkých vytvorených udalostí. Z tejto stránky je možné sa tiež k udalosti pridať, alebo z nej odísť. Užívateľ tu nájde stručný prehľad každej udalosti, ako napríklad názov udalosti, dátum a čas, mesto konania udalosti, ale aj tlačidlo pre zobrazenie pripojených užívateľov, a malý náhľad lokality udalosti na mape.

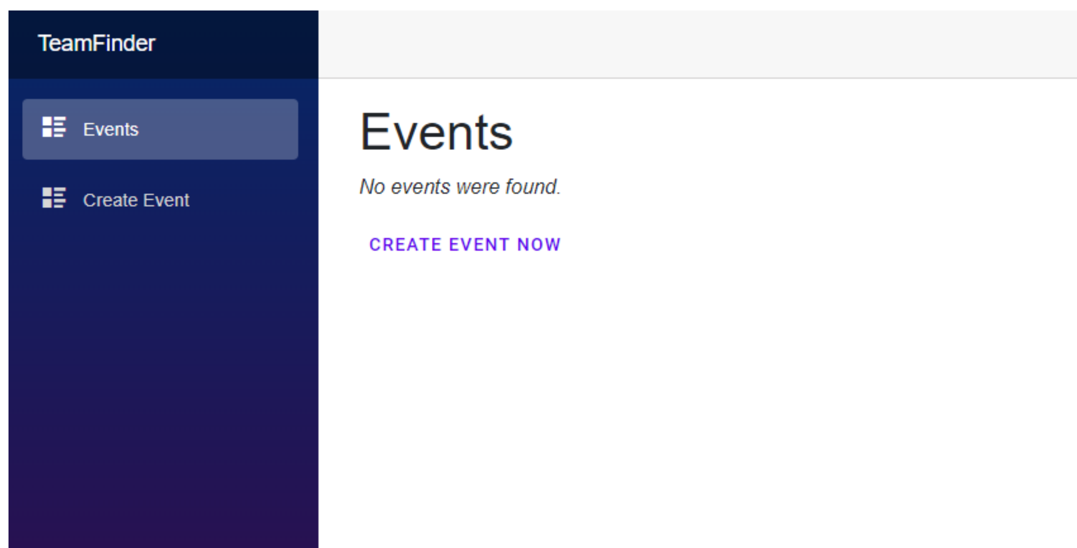



Obrázok 20 Stránka vyhľadávania udalostí

Udalosti na tejto stránke sú zoradené podľa dátumu, to znamená: medzi prvými užívateľ uvidí plánované udalosti, ktoré ešte len budú prebiehať, na konci v zozname budú udalosti, ktoré už prebehli.

Užívateľ si môže vybrať, koľko udalosti chce na stránku zobraziť. Zároveň sa môže prepínať medzi týmito stránkami – stránkovanie.

V prípade, že by neboli vytvorené žiadne udalosti sa na stránke zobrazí oznam o tejto skutočnosti, a užívateľovi aplikácia ponúkne založiť novú udalosť (viď Obrázok 21 nižšie).



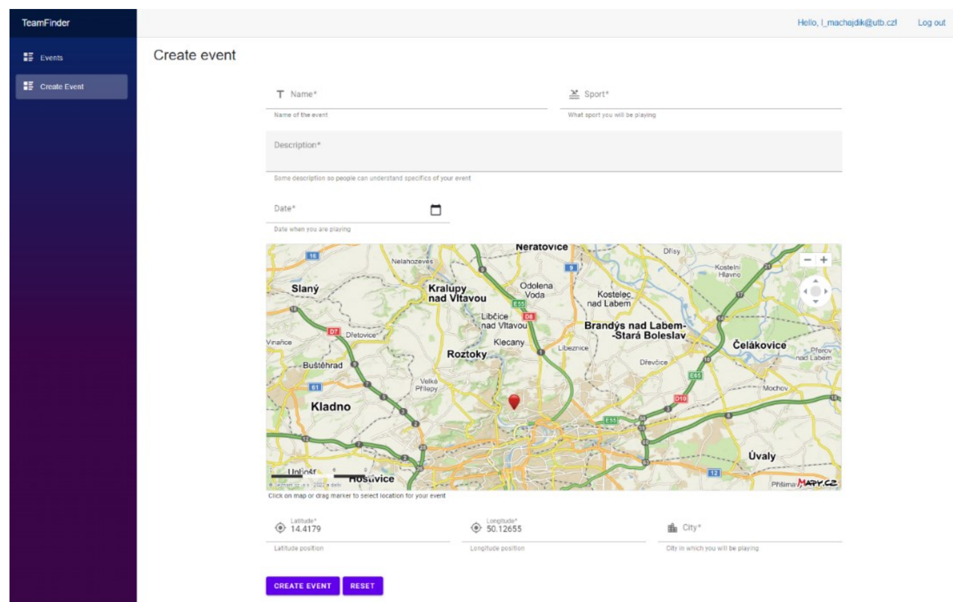

Obrázok 21 Prípad, kedy neboli vytvorené žiadne udalosti

### 5.1.2 Vytvorenie udalosti

Po úspešnom vytvorení udalosti je užívateľ uvedený ako vlastník tejto udalosti. Zároveň, aby sa predišlo vzniku udalosti kde nebude žiadny organizátor, zakladateľ udalosti nemôže z tejto udalosti odísť.

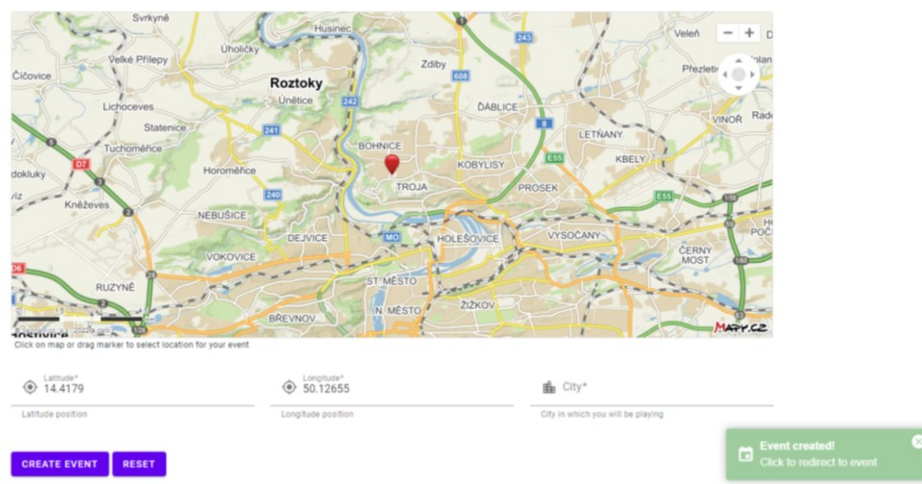
Lokalitu užívateľ zvolí prostredníctvom interaktívnej mapy. Kliknutím na mape užívateľ vytvorí značku na mape, vďaka čomu vieme, kde užívateľ bude chcieť udalosť organizovať.

Po vytvorení značky sa prepíšu údaje o lokalite do príslušných polí, ako je zemepisná šírka a zemepisná výška, ale aj mesto, v ktorom užívateľ zvolil, že sa bude udalosť konať. O samotnom komponentu mapa si povieme neskôr.




Obrázok 22 Stránka vytvorenia udalosti

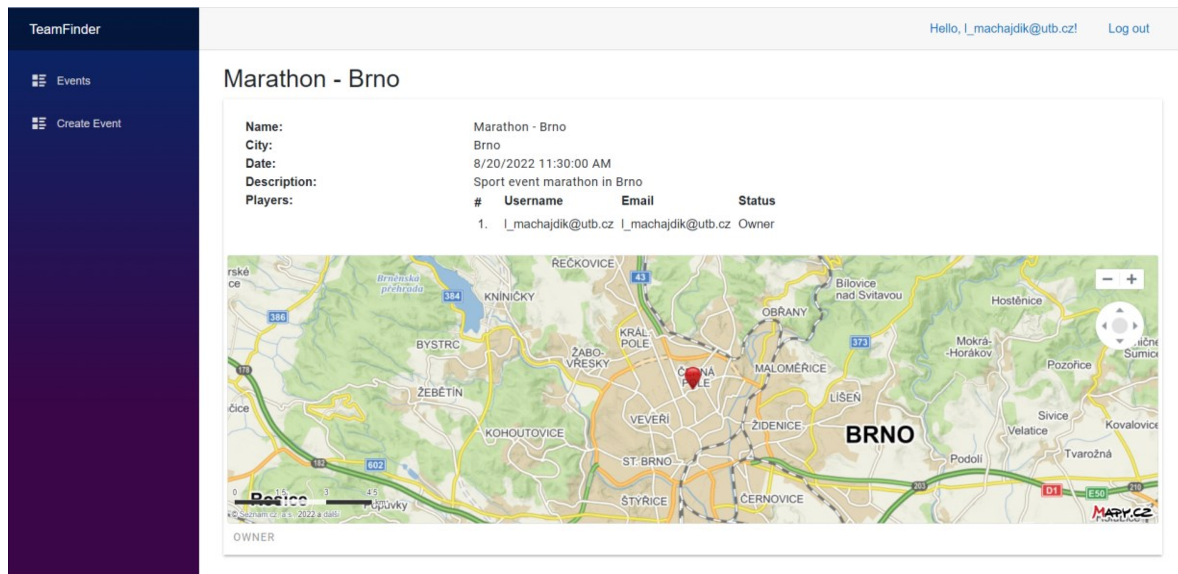
Po kliknutím na tlačidlo vytvoriť udalosť bude udalosť na stránke vytvorená, a užívatelia ju môžu nájsť na stránke vyhľadávania udalostí. Zároveň sa v pravom dolnom rohu užívateľovi zobrazí notifikácia o vytvorení tejto udalosti, a kliknutím na notifikáciu sa môže presmerovať na stránku detailu tejto udalosti.




Obrázok 23 Notifikácia o vytvorení udalosti v pravom dolnom rohu

### 5.1.3 Detail udalosti

Na stránke detailu udalosti sa nachádzajú všetky informácie o zvolenej udalosti, vrátane výpisu užívateľov, ktorý sa pripojili k tejto udalosti. Nachádza sa tu aj interaktívna mapa s označeným miestom konania udalosti. Užívateľ sa zároveň na tejto stránke môže k udalosti pripojiť alebo z nej odísť.




Obrázok 24 Stránka detailu udalosti

## 5.2 Komponenty

Navrhnutá webová aplikácia využíva metodológie atomického dizajnu, ktorý rozdeľuje komponenty. Od najmenších po najväčšie sú tu atómy, molekuly, organizmy, šablóny a stránky. Pre lepšie pochopenie, o čom webová aplikácia je, sme si už o jednotlivých stránkach povedali v kapitole 5.1 Stránky, preto popis stránok z tejto kapitoly vynecháme. Bližšie si o Atomickom Dizajne povieme v kapitole 5.4 Použité postupy.

Väčšina nižšie vymenovaných komponentov využíva knižnicu MatBlazor, ktorá zjednodušuje prácu pri tvorbe grafického rozhrania tým, že predstavuje sadu použiteľných komponentov.

### 5.2.1 Atómy

**CityField**

CityField je textové pole s výpisom výberu mesta udalosti. Tento komponent je použitý na stránke vytvárania udalosti. Vzhľadom na to, že mesto sa do tohto poľa zapíše po výbere lokality na mape, tento komponent slúži iba na čítanie, ale nie na zápis. Toto platí z rovnakého dôvodu aj pre atómy *LatitudeField* a *LongitudeField*.

**LatituteField**

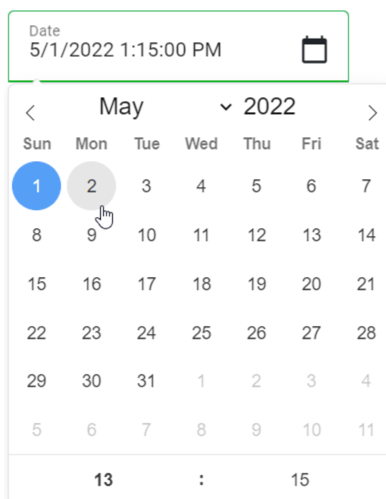
LatitudeField je textové pole, do ktorého sa po výberu miesta konania vytváranej udalosti prepíše hodnota príslušnej zemepisnej šírky.

**LongitudeField**

LatitudeField je textové pole, do ktorého sa po výberu miesta konania vytváranej udalosti prepíše hodnota príslušnej zemepisnej výšky.

**EventDatePicker**

EventDatePicker využíva MatBlazor komponentu pre výber dátumu a času. Je ošetrený, aby nebolo možné zadať neplatný dátum v prípade, že užívateľ hodnotu v textovom poli prepíše, a zároveň aby nebolo možné zadať dátum udalosti v minulosti. Zelený rám okolo textového poľa značí, že vstup je správny. V prípade, že by užívateľ prepísal hodnotu textového poľa, a vstup by nebol správny, zobrazí sa červený rám okolo textového poľa. Podobne sú ošetrené aj ostatné vstupy.




Obrázok 25 Komponent EventDatePicker

**EventButton**

EventButton je komponent, ktorý sa zobrazuje keď je popisovaná nejaká udalosť. Užívateľ sa vďaka tomuto tlačidlu dokáže k udalosti pripojiť, alebo sa z nej odpojiť. Pokiaľ užívateľ udalosť založil, zobrazí sa iba text „Owner“, ktorý značí, že užívateľ udalosť založil, a preto z nej nemôže odísť, aj keď je k nej pripojený. V prípade, že by udalosť už skončila, a teda zobrazovala by sa udalosť, ktorá prebehla v minulosti, zobrazí sa text „Finished“. Kliknutím na tlačidlo, keď je komponent v stave *Owner* alebo *Finished* sa nevykoná žiadna akcia, pretože žiadna nie je potrebná.

**PlayerTable**

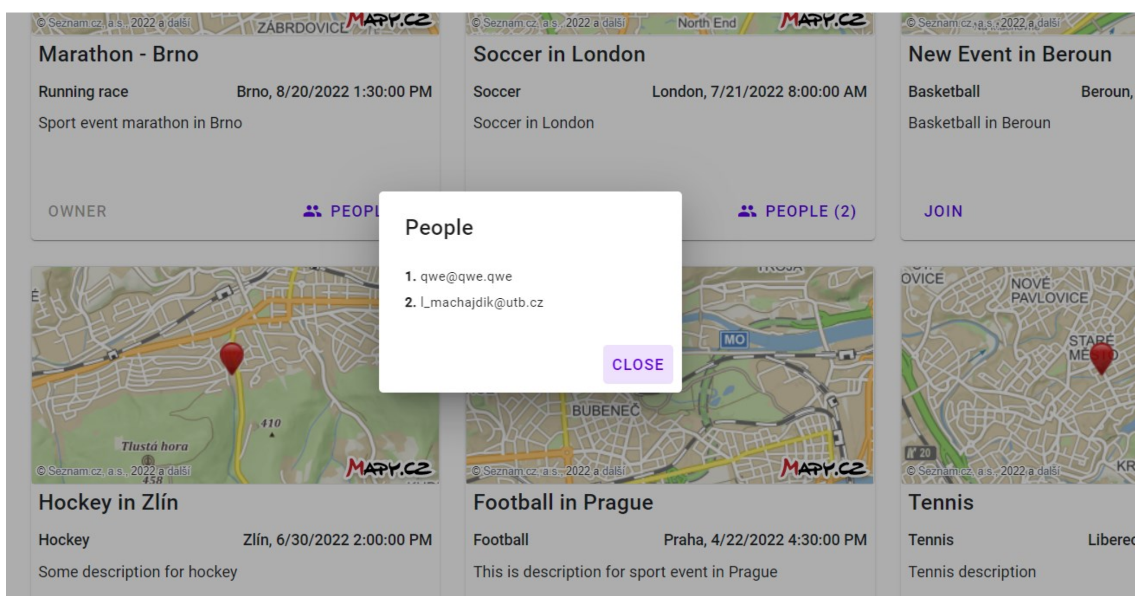
Tento komponent slúži pre vypísanie účastníkov udalosti. Zakladateľ udalosti má vždy pridelené číslo 1, avšak nemusí byť vždy na prvý v zozname. Je to preto, lebo tabuľka sa dá zoradiť podľa stĺpcov v tejto tabuľke. K tomuto bol využitý komponent knižnice MatBlazor.

| # | Username | Email | Status |
|---|---|---|---|
| 1. | testUser | qwe@qwe.qwe | Owner |
| 2. | l_machajdik | l_machajdik@utb.cz | Joined |

Obrázok 26 Komponent PlayerTable

**PlayerListDialog**

Tento komponent zobrazuje pripojených užívateľov k udalosti. Je využitý v komponente EventCard. Po kliknutí na tlačidlo People sa dialóg otvorí a užívateľ uvidí pripojených ľudí k tejto udalosti. Kliknutím na tlačidlo zavrieť, alebo kliknutím mimo dialógu sa dialóg zatvorí.




Obrázok 27 Komponent PlayerListDialog

**Map**

Map je komponent využívajúci mapu od Mapy.cz. Má dostupne API pre prispôsobenie a nastavenie mapy podľa potrieb užívateľa. Vo webovej aplikácii toto API využívame pre zobrazenie miesta, kde sa udalosť bude konať. Užívateľovi tak ponúkame vizuálny prehľad,

kde sa udalosť bude konať. Pre tieto účely využívame 3 rôzne implementácie tejto mapy, ktoré odlišujeme stavmi *SimpleMap*, *InteractiveMap* a *PositionalMap*.

SimpleMap je implementácia, ktorá zobrazuje iba náhľad mapy. Nie je interaktívna, a na mape je zobrazená iba poloha udalosti pomocou značky, ktorá je vycentrovaná v strede.

InteractiveMap využíva Mapy.cz API pre mapu v interaktívnej podobe. Užívateľ tak môže mapu odďaľovať, približovať, posúvať sa na iné miesta na mape. Mapa je spočiatku vycentrovaná na značku, ktorá značí miesto konania udalosti. V prípade, že sa užívateľ posunie na mape tak, že by značka udalosti už nebola vidieť, zobrazí sa na okraji značka v smere udalosti. Táto značka zostáva pri okraji, a po kliknutí na ňu sa mapa presunie na miesto pôvodnej značky, teda presunie sa miesto konania udalosti.

PositionalMap je stav, ktorý oproti ostatným nevyužívame pre zobrazenie miesta konania vytvorenej udalosti, ale využívame túto komponentu pri tvorbe udalosti. Užívateľ zvolí miesto konania udalosti tak, že na interaktívnej mape zvolí miesto a kliknutím na mape, kde plánuje jeho udalosť organizovať. Na tomto mieste sa zobrazí značka udalosti. Opätovným kliknutím na iné miesto sa značka presunie na iné miesto, ale užívateľ so značkou dokáže pohnúť aj jednoduchým pretiahnutím značky na inú pozíciu. Následne sa zistí zemepisná šírka a výška miesta kde užívateľ klikol, spolu s názvom mesta na daných súradniciach. Tieto informácie sa potom prepíšu do príslušných komponentov a zobrazia sa užívateľovi. Viac-menej sa jedná o rovnakú implementáciu ako v predchádzajúcej mape, rozdielom je to, že užívateľ si dokáže sám zvoliť, kde sa bude značka nachádzať.

### 5.2.2 Molekuly

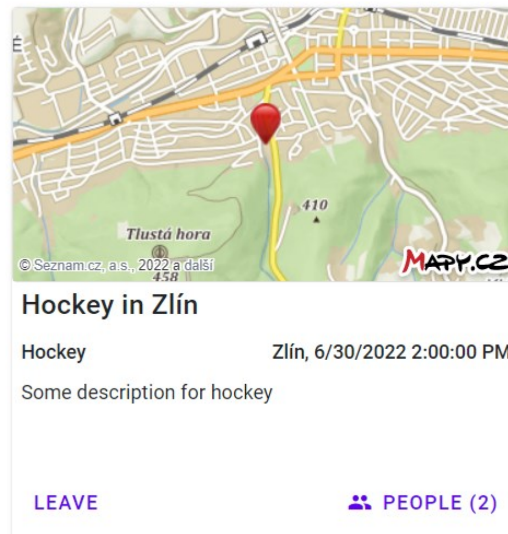
#### EventCard

Tento komponent sa nachádza na stránke vyhľadávania udalostí. Zobrazuje prehľad o prezeranej udalosti. Využíva atómy:

- Map v stave SimpleMap
- EventButton
- PlayerListDialog

Komponent EventCard slúži ako stručný náhľad udalosti. Užívateľ si tak môže pozrieť všetky potrebné informácie aby zistil, či má záujem pripojiť sa k udalosti. Medzi tieto informácie patrí náhľad miesta konania udalosti, názov udalosti, šport, mesto,

dátum a čas konania udalosti, popis udalosti, a po kliknutí na tlačidlo *People* aj pripojených užívateľov v udalosti.




Obrázok 28 EventCard komponent

**LocationFields**

LocationFields je komponent, ktorý združuje polia zemepisnej šírky, zemepisnej výšky, a mesta konania udalosti. Využíva teda komponenty:

- LatitudeField
- LongitudeField
- CityField

### 5.2.3 Organizmy

**EventForm**

EventForm je komponent na stránke vytvárania udalosti, ktorý slúži ako formulár pre zadanie údajov potrebných pre vytvorenie udalosti. Užívateľ na stránke vytvorenia udalosti musí vyplniť potrebné údaje:

- Názov udalosti – textové pole
- Popis udalosti – textové pole
- Šport – textové pole
- Dátum a čas – komponent EventDatePicker
- Lokalita – komponenty LocationFields a Map v stave PositionalMap

Všetky vstupy sú ošetrené. Užívateľovi nebude umožnené udalosť vytvoriť, pokiaľ vynechal nejaký údaj, alebo ho zadal nesprávne. Pri textových vstupoch sa kontroluje, či užívateľ vôbec nejaký vstup zadal, a či tento vstup spĺňa minimálnu a maximálnu dĺžku. Pri dátume a čase sa navyše porovnáva, či je zvolený platný dátum a čas, a či užívateľ nevybral dátum v minulosti.

V prípade, že by si udalosť užívateľ rozmyslel, a chcel by vytvoriť inú, zíde sa mu tlačidlo reset. Kliknutím na toto tlačidlo sa všetky polia zresetujú, a užívateľ môže vyplňovať od znovu.

Zresetovanie vstupov tiež nastane vždy, keď užívateľ vytvorí udalosť. Týmto zároveň ošetríme prípad, kedy by užívateľ nechtiac klikol na tlačidlo viackrát. Namiesto toho, aby sa vytvorilo viac udalostí, vytvorí sa jedna udalosť, a vzhľadom na to, že sa vstupy zresetovali, aplikácia užívateľa nepustí vytvoriť udalosť s prázdnymi – neplatnými vstupmi.

**EventView**

V tomto komponente sa načítajú vytvorené udalosti, a vykreslia sa komponenty EventCard s popismi o jednotlivých udalostiach. Jedná sa teda o pole udalosti, ktoré sa vykreslia v komponentoch EventCard. Zároveň je možné cez parameter komponentu zadať rozsah udalostí, ktoré sa majú zobraziť. Toto je využité na stránke vyhľadávania udalostí pre stránkovanie udalostí.

## 5.3 Databáza

Webová aplikácia využíva databázu, s ktorou komunikuje prostredníctvom frameworku Entity Framework. Nakoľko prenájom vzdialenej databáze by bol spoplatnený, databázu v tejto práci využívame iba lokálne.

Entity Framework poskytuje viacero druhov databáz. pre účely našej aplikácie je využitá databáza Microsoft SQL Server.

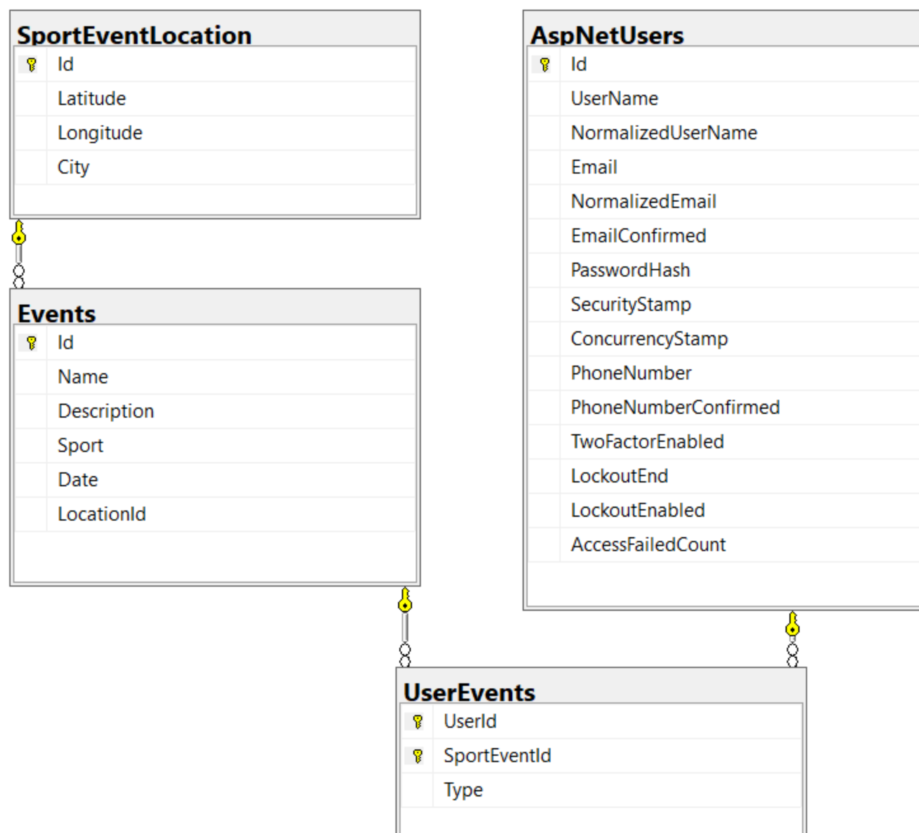
Využívame aj ASP.NET Core Identity, ktorý podporuje prihlasovaciu funkcionalitu v užívateľskom rozhraní, a spravuje používateľov, heslá, role, atď. [2]

Vďaka Identity nemusíme v aplikácii riešiť napríklad registráciu alebo prihlasovanie, ale Identity to vyrieši za nás.

Databázu využívame prístupom Code-First, kedy je z napísaného zdrojového kódu aplikácie vygenerovaná databáza – podľa zadaných vzťahov, ktoré vyplývajú z kódu.

Pre svoj správny chod Identity vytvorí niektoré tabuľky v databáze sám, ako napríklad tabuľku *AspNetRoles*. Túto tabuľku síce v databáze máme, ale v momentálnom riešení práce ju nevyužívame. Využili by sme ju, keby chceme definovať role užívateľom, napríklad rolu Administrátor.

Vygenerované tabuľky od ASP.Net Core Identity sme z ER diagramu na obrázku 29 nižšie vynechali, samozrejme až na tabuľku *AspNetUsers*.




Obrázok 29 Entity Relationship Diagram

Tabuľka *AspNetUsers* eviduje zaregistrovaných užívateľov, a ukladá o nich informácie ako je prihlasovacie meno, email, heslo a podobne.

V tabuľke *Events* ukladáme informácie o vytvorených udalostiach, akými sú názov udalosti, popis, šport, dátum a čas, a lokalita.

Tabuľka *UserEvents* definuje vzťah viac-ku-viac medzi užívateľmi a udalosťami. Jedná sa o tabuľku, ktorá uchováva informácie o tom, aký má užívateľ vzťah k akej udalosti. To o aký vzťah sa jedná vyplýva z vlastnosti Type. Užívateľov vzťah k udalosti je konštanta, ktorá môže byť buď vlastník v prípade, že užívateľ udalosť vytvoril, alebo pripojený v prípade, že užívateľ nie je vlastníkom a pripojil sa k udalosti. Znamená to, že keď sa užívateľ pridá

k udalosti, nachádza sa s touto udalosťou vo vzťahu pripojený, ale keď sa následne odpojí, tak sa vymaže celý jeho záznam z tabuľky, pretože užívateľ k udalosti už nemá žiadny vzťah.

Poslednou tabuľkou v ER Diagrame je tabuľka *SportEventLocation*. Táto tabuľka obsahuje informácie o lokalite udalosti, a teda má informácie o zemepisnej šírke, zemepisnej výške a meste tejto udalosti.

## 5.4 Použité postupy

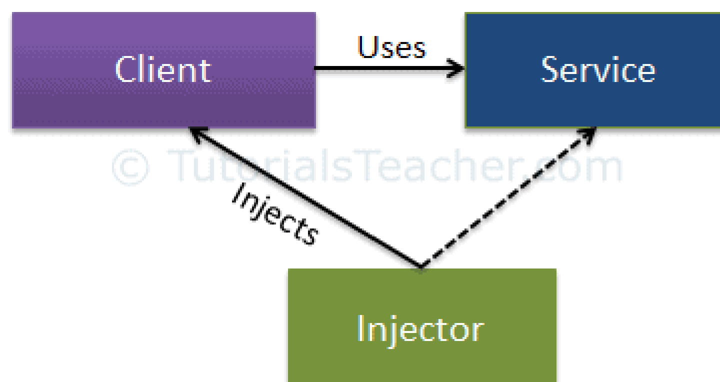
### 5.4.1 Inversion of Control

Inversion of Control (IoC) je návrhový princíp, ktorý odporúča inverziu rozdielnych prvkov v objektovo-orientovanom návrhu, aby sa dosiahlo voľných spojení medzi aplikačnými triedami. Dependency Injection je návrhový vzor, ktorý je používaný na implementáciu IoC. [16]

Dependency Injection zahŕňa 3 typy tried: [16]

- Klientska trieda je trieda, ktorá má závislosť na službe
- Služba je trieda, ktorá poskytuje závislosť klientskej triede
- Injector je trieda, ktorá injectuje objekt služby to triedy klienta

Na obrázku 30 sú zobrazené vzťahy medzi týmito triedami.




Obrázok 30 Vzťah medzi typmi tried v Dependency Injection [16]

Závislosť je objekt, na ktorý je iný objekt závislí. Závislosti sa zbavíme vytvorením rozhrania, ktorý bude objekt implementovať, alebo rodičovskej triedy, z ktorej objekt bude dediť, a toto následne zaregistrujeme v service kontajneri. V poslednom kroku injectujeme službu v konštruktore triedy, v ktorej túto službu chceme použiť. [2]

Dependency Injection umožňuje vytvorenie závislých objektov mimo triedy, a poskytuje tieto objekty triede rôznymi spôsobmi: [16]

- Constructor Injection – v takomto prípade je závislosť injectovaná do triedy cez konštruktor danej triedy
- Property Injection – závislosť je poskytnutá pomocou verejnej property v klientskej triede
- Method Injection – v tomto prípade klientska trieda implementuje rozhranie, ktoré deklaruje metódy, a injector používa toto rozhranie, ktoré poskytuje klientskej triede

Kód nižšie ukazuje príklad využitia constructor injection vo vytvorenej webovej aplikácii:

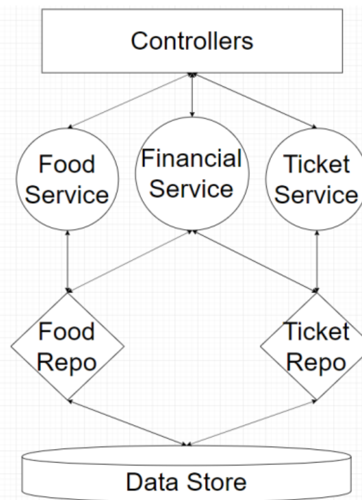
```
[ApiController]
public class EventsController : ControllerBase
{
   private readonly IMapper _mapper;
   private readonly IEventService _eventService;
   private readonly IUserEventsService _userEventsService;

   public EventsController(IMapper mapper, IEventService eventService,
IUserEventsService userEventsService)
   {
      _mapper = mapper;
      _eventService = eventService;
      _userEventsService = userEventsService;
   }
}
```

### 5.4.2 Repository-Service Pattern

Táto architektúra sa používa už nejakú dobu, ale niekto o nej ešte nemusel počuť. Repository-Service pattern delí biznisovú logiku aplikácie do dvoch odlišných vrstiev: [17]

- **Repozitáre** sú nižšou vrstvou. Úlohou týchto tried je dostať dáta do nášho úložiska dát, prípadne ich z neho získať. Dôležité je dodať, že pre každý repozitár musí pracovať iba s jedným modelom.
- **Služby** sú vyššou vrstvou. Tieto triedy budú mať v sebe injectované repozitáre, aby mohli spracovávať dáta z jedného alebo z viacerých repozitárov. Môžu tiež kombinovať tieto dáta, aby vytvorili zložitejšie biznisové objekty. Navyše predstavujú ďalšiu vrstvu abstrakcie medzi webovou aplikáciou a repozitármi, aby mohli byť menené nezávislejšie.

Obrázok 31 nižšie predstavuje ukážkový diagram architektúry podľa repository-service pattern.

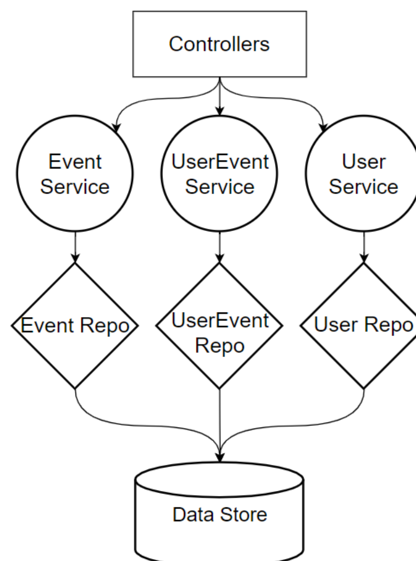



Obrázok 31 Repository-service pattern [17]

Repository-Service pattern využíva Dependency Injection aby mohol fungovať. Triedy na jednotlivých vrstvách budú injectovať triedy z nižších vrstiev. [17]

V takto navrhnutej architektúre nie je problém zameniť závislosti podľa potreby. Bolo by možné napríklad cachovať výsledky, a v prípade, že by sa našli už nacachované dáta, mohli by sme použiť tie, inak by sa použili dáta, čo by sa našli z databáze. Server by sme tak odľahčili o jednu či dve požiadavky.

Obrázok 32 nižšie zobrazuje ako bol využitý repository-service pattern vo vytvorenej webovej aplikácii.




Obrázok 32 Repository-service pattern v navrhnutej webovej aplikácii

Nasledujúcim kódom nižšie v aplikácii registrujeme repozitáre a služby:

```
//repositories
builder.Services.AddScoped<IEventRepository, EventRepository>();
builder.Services.AddScoped<IUserRepository, UserRepository>();
builder.Services.AddScoped<IUserEventsRepository, UserEventsRepository>();

//services
builder.Services.AddScoped<IEventService, EventService>();
builder.Services.AddScoped<IUserService, UserService>();
builder.Services.AddScoped<IUserEventsService, UserEventsService>();
```

### 5.4.3 Atomický dizajn

Atomický dizajn je metodológia, ktorej myšlienka pochádza z chémie a zložení hmoty. V chémii majú atómy pevne dané vlastnosti, ktorými sú definované. Samostatný kyslík alebo vodík sú atómy s nezávislými vlastnosťami. Keď sa ale atómy spoja, vytvoria molekuly, ktoré nadobúdajú svoje vlastné charakteristiky. V prípade spojenia kyslíku s 2 atómami vodíka vytvoríme molekulu vody. [18]

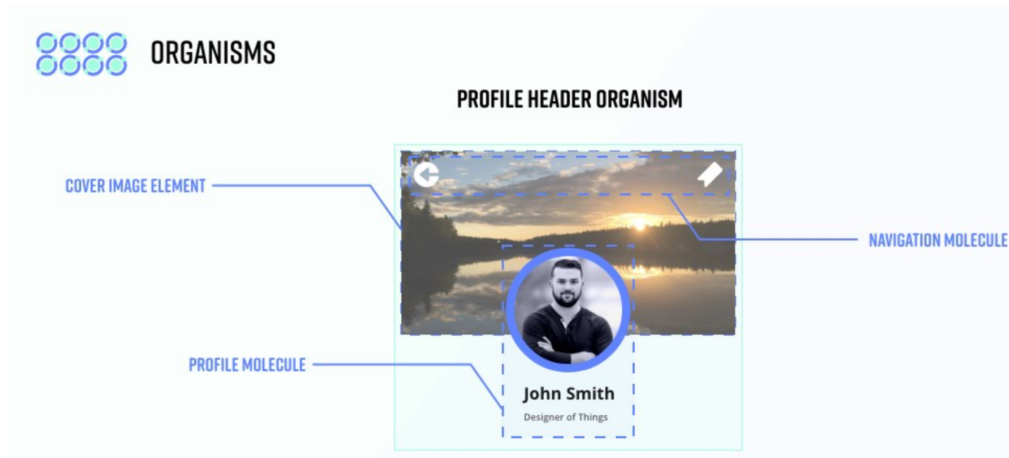
Dokážeme skombinovať viaceré menšie elementy, aby sme získali väčšie objekty pri budovaní našich návrhov. Atomický dizajn nám ponúka nielen štruktúru pre tvorenie našich návrhov, ale vytvára stavebné bloky pre budovanie systémov alebo knižníc. [18]

Podľa atomického dizajnu existuje 5 odlišných prvkov, od toho najmenšieho po najväčší sú to: [18]

1. Atómy
2. Molekuly
3. Organizmy
4. Šablóny
5. Stránky

Rovnako ako v chémii je atóm najmenším stavebným blokom v systéme. V softvérovom poňatí sa jedná o tlačidlá, textové polia, ikony, atď. [18]

Molekuly sú stavebným blokom, ktorý je tvorený atómami. Napríklad spojením ikony profilu a labelu s textom užívateľského mena sme schopný vytvoriť molekulu, zobrazujúcu náhľad profilu užívateľa, ktorý je možné použiť pri vyhľadávaní užívateľov. Ak by sme k našej molekule pridali ešte obrázok pozadia a navigačnú lištu, dokázali by sme vytvoriť organizmus, ktorý predstavuje úvodnú hlavičku na stránke profilu užívateľa, podobne ako to je na obrázku 33 nižšie. [18]




Obrázok 33 Organizmus hlavička na stránke profilu [18]

O jednotlivých prvkoch atomického dizajnu v našej webovej aplikácii sme si už povedali v kapitole komponenty.

## 5.5 Klonovanie repozitára

Keďže vo webovej aplikácii používame lokálnu databázu, po naklonovaní projektu z repozitára musíme nastaviť databázu. Je potrebné si nainštalovať SQL Server a vykonať migráciu databáze pomocou príkazu „*dotnet ef database update migration*“ v zložke servera. Iba potom bude webová aplikácia funkčná.

Pokiaľ je dostupná vzdialená databáza, je možné namiesto toho prepísať *ConnectionString* v *AppSettings* projektu na vzdialenú databázu, a migráciu vykonať na vzdialenej databáze.

# 6 AUTOMATIZOVANÉ TESTY

Automatizované testy boli napísané s použitím testovacej knižnice bUnit. Triedy boli rozdelené tak, aby každá trieda testovala jeden komponent. V nasledujúcej kapitole si o jednotlivých testovacích triedach povieme viac.

## 6.1 Testy

Vo všetkých testoch boli namockované odpovede zo servera, vďaka čomu bolo možné simulovať, ako sa aplikácia zachová v daných podmienkach.

Rovnako bola simulovaná aj autorizácia užívateľa, a teda či je užívateľ prihlásený alebo odhlásený. Viac si o tomto povieme v kapitole 6.2 Použité postupy.

Podobne boli simulované všetky ostatné počiatočné podmienky v testoch, ktoré sú potrebné pre správnosť testov – preconditions. Následne sa vykonajú kroky testu, a vyhodnotia sa podmienky po vykonaní testu – postconditions.

### 6.1.1 EventButtonTests

Tieto testy testujú komponentu EventButton, a či sa nachádza za konkrétnych podmienok v správnom stave ako pred testom, tak po aj teste.

V tejto triede sú tiež definované konštanty s hodnotou textu tlačidla, vďaka čomu môžeme zistiť, v akom stave sa tlačidlo nachádza:

- OwnerText – užívateľ je vlastník udalosti, nemalo by mu byť umožnené z udalosti odísť, ani sa k nej pripojiť
- JoinText – užívateľ nie je pripojený k udalosti, po kliknutí na tlačidlo sa k nej pripojí
- LeaveText – užívateľ je pripojený k udalosti, po kliknutí na tlačidlo z nej odíde
- FinishedText – udalosť už skončila. Užívateľ sa k nej nemôže ani pridať, ani z nej odísť
- LoginText – užívateľ nie je prihlásený, po kliknutí na tlačidlo užívateľa presmeruje na stránku, kde sa môže prihlásiť

#### *6.1.1.1 JoinEventTest*

Tento test porovnáva, či sa správne prepíšu vlastnosti komponentu po tom, čo sa užívateľ korektne pripojí k udalosti.

*Preconditions:*

- Užívateľ je prihlásený
- Užívateľ nie je pripojený k udalosti, ani nie je jej vlastníkom
- Požiadavka zo servera o pripojený k udalosti prebehne úspešne
- Udalosť ešte neskončila

*Testovacie kroky:*

1. Na začiatku sa nájde tlačidlo v komponente
2. Uloží sa text z tlačidla do premennej – text pred kliknutím
3. Na toto tlačidlo sa klikne, a čaká sa kým sa doňho prepíše konštanta *LeaveText.*
4. Uloží sa text z tlačidla do premennej – text po kliknutí

*Postconditions*:

- Text tlačidla pred kliknutím je zhodný s textom *JoinText*
- Text tlačidla po kliknutí je zhodný s textom *LeaveText*
- Prepísal sa vzťah užívateľa k udalosti na pripojený

#### *6.1.1.2 LeaveEventTest*

Tento test porovnáva, či sa správne prepíšu vlastnosti komponentu po tom, čo užívateľ korektne odíde z udalosti.

*Preconditions:*

- Užívateľ je prihlásený
- Užívateľ je pripojený k udalosti
- Požiadavka zo servera o odídení z udalosti prebehne úspešne
- Udalosť ešte neskončila

*Testovacie kroky:*

1. Uloží sa text z tlačidla do premennej – text pred kliknutím
2. Na toto tlačidlo sa klikne, a čaká sa kým sa doňho prepíše konštanta *JoinText.*
3. Uloží sa text z tlačidla do premennej – text po kliknutí

*Postconditions*:

- Text tlačidla pred kliknutím je zhodný s textom *LeaveText*
- Text tlačidla po kliknutí je zhodný s textom *JoinText*

- Prepísal sa vzťah užívateľa k udalosti – žiadny vzťah

#### *6.1.1.3 CannotJoinFinishedEventTest*

V tomto teste sa zameriame na udalosť, ktorá už skončila. V prípade, že užívateľ nie je pripojený k udalosti, by toto tlačidlo malo slúžiť na to, aby sa k udalosti pripojil. Ale ak zároveň už udalosť skončila, nemalo by mu byť umožnené v takýchto podmienkach kliknutím na tlačidlo sa k udalosti pripojiť.

*Preconditions:*

- Užívateľ je prihlásený
- Udalosť už skončila
- Užívateľ nie je k udalosti pripojený, ani ju nevytvoril – nemá k nej väzbu

*Testovacie kroky:*

1. Uloží sa text z tlačidla do premennej – text pred kliknutím
2. Na tlačidlo v tejto komponente sa klikne, a následne sa určitú dobu čaká, či sa text zmení na niečo rozdielne od textu *FinishedText*
3. Uloží sa text z tlačidla do premennej – text po kliknutí

*Postconditions*:

- Text tlačidla pred kliknutím je zhodný s textom *FinishedText*
- Text tlačidla po kliknutí je zhodný s textom *FinishedText*
- Väzba užívateľa k udalosti zostala nezmenená – žiadna väzba

#### *6.1.1.4 OwnerCantLeaveTest*

Ak užívateľ udalosť vytvoril, nemalo by mu byť umožnené z nej odísť. Práve týmto sa zaoberá tento test.

*Preconditions:*

- Užívateľ je prihlásený
- Udalosť ešte neskončila
- Užívateľ túto udalosť vytvoril

*Testovacie kroky:*

1. Uloží sa text z tlačidla do premennej – text pred kliknutím

2. Na toto tlačidlo sa klikne, a následne sa po určitú dobu čaká, či sa text zmení na niečo rozdielne od textu *OwnerText*
3. Uloží sa text z tlačidla do premennej – text po kliknutí

*Postconditions*:

- Text tlačidla pred kliknutím je zhodný s textom *OwnerText*
- Text tlačidla po kliknutí je zhodný s textom *OwnerText*
- Užívateľ je v udalosti stále evidovaný ako vlastník tejto udalosti

#### *6.1.1.5 NotLoggedInTest*

Ak užívateľ nie je prihlásený, mal by sa mu po kliknutí na tlačidlo v tomto komponente zobraziť text *LoginText*. Po kliknutí na tlačidlo by mal byť užívateľ presmerovaný na stránku, kde sa môže prihlásiť.

V tomto teste vyžadujeme službu NavigationManager, aby sme zistili aktuálnu Uri. Viac si o tomto povieme v kapitole 6.2 Použité postupy.

*Preconditions:*

- Užívateľ nie je prihlásený
- Udalosť ešte neskončila

*Testovacie kroky:*

1. Uloží sa text z tlačidla do premennej – text pred kliknutím
2. Klikne sa na tlačidlo a určitú dobu sa čaká, či sa text zmení na niečo rozdielne od textu *LoginText*
3. Uloží sa text z tlačidla do premennej – text po kliknutí

*Postconditions*:

- Uri z NavigationManagera je zhodná s Uri stránkou prihlásenia užívateľa
- Text tlačidla pred kliknutím je zhodný s textom *LoginText*
- Text tlačidla po kliknutí je zhodný s textom *LoginText*
- Aktuálna Uri z NavigationManagera nie je zhodná z Uri pred kliknutím

### 6.1.2 EventCardTests

Táto testovacia trieda testuje komponentu EventCard.

#### *6.1.2.1 MapIsSimpleMap*

Tento krátky test testuje, či je v komponente nastavená mapa v správnom stave *SimpleMap*, aby mapa slúžila iba na náhľad lokality.

*Testovacie kroky:*

1. Nájdi komponent Map, ktorý by sa mal nachádzať vo vnútri komponentu EventCard

*Postconditions*:

- Komponenta Map je nastavená v stave *SimpleMap*

#### *6.1.2.2 PeopleListExpandsOnClick*

V EventCard komponentu je tlačidlo People. Po kliknutí na toto tlačidlo sa zobrazí dialógové okno, ktoré vypisuje zoznam užívateľov k udalosti. Tento test testuje, či keď užívateľ klikne na tlačidlo, či sa správne otvorí toto dialógové okno.

*Testovacie kroky:*

1. Nájdi komponent PlayerListDialog, ktorý by sa mal nachádzať vo vnútri komponentu EventCard
2. Premennú PlayersExpanded komponentu PlayerListDialog, ktorá značí, či je dialógové okno otvorené ulož do premennej
3. Nájdi tlačidlo People a klikni naň
4. Do premennej ulož stav v premennej PlayersExpanded komponentu PlayerListDialog

*Postconditions*:

- Uložená premenná z kroku 2 značí, že dialógové okno nie je otvorené
- Hodnota premennej z kroku 4 značí, že dialógové okno je po kliknutí na tlačidlo otvorené.

#### *6.1.2.3 EventDetailsAreCorrect*

Vzhľadom na to, že v testovanej komponente je cez parameter tiež predávaná inštancia udalosti, a nachádza sa tu veľa informácii, ktoré popisujú udalosť, vzniká potreba kontrolovať, či sa tieto dáta prepisujú do komponenty správne.

*Testovacie kroky:*

1. Nájdi element, v ktorom je vypísaný názov udalosti a ulož hodnotu jeho textu do premennej
2. Nájdi element, v ktorom je vypísaný popis udalosti a ulož hodnotu jeho textu do premennej
3. Nájdi element, v ktorom je vypísaný šport udalosti a ulož hodnotu jeho textu do premennej
4. Nájdi element, v ktorom je vypísaný dátum a čas udalosti a ulož hodnotu jeho textu do premennej

*Postconditions*:

- Porovnaj, či sa názov udalosti z uloženej premennej zhoduje s názvom udalosti inštancie predanej ako parameter
- Porovnaj, či sa popis udalosti z uloženej premennej zhoduje s popisom udalosti inštancie predanej ako parameter
- Porovnaj, či sa šport udalosti z uloženej premennej zhoduje so športom udalosti inštancie predanej ako parameter
- Porovnaj, či sa dátum a čas udalosti z uloženej premennej zhoduje s dátumom a časom udalosti inštancie predanej ako parameter

### 6.1.3 EventFormTests

Táto testovacia sada obsahuje testy komponenty, ktorá slúži ako formulár pri vytváraní udalosti.

#### *6.1.3.1 MapIsSetToPositionalMap*

Vzhľadom na to, že v testovanej komponente využívame mapu pre zadávanie lokality, tento test zisťuje, či je mapa nastavená do stavu *PositionalMap*.

*Testovacie kroky:*

1. Nájdi komponent Map v komponente EventForm

*Postconditions*:

- Porovnaj, či je typ mapy nastavený na *PositionalMap*

#### *6.1.3.2 MinimumDateIsSetToToday*

V tejto komponente využívame komponent EventDatePicker, ktorý by mal mať ošetrené, že ako dátum udalosti sa nedá zvoliť dátum v minulosti.

*Testovacie kroky:*

1. Nájdi komponent EventDatePicker

*Postconditions*:

- Porovnaj, či je minimálny dátum a čas, ktorý užívateľ v komponente dokáže vybrať zhodný s aktuálnym dátumom a časom

#### *6.1.3.3 EventPropertiesMatchEventData*

Ako parameter do komponentu sa predáva inštancia udalosti. Test následne porovnáva, či sú jednotlivé informácie o udalosti v textových poliach komponentu zhodné s hodnotou z inštancie, ktorú sme predali ako parameter.

*Testovacie kroky:*

1. Nájdi textové pole s názvom udalosti a text v ňom ulož do premennej
2. Nájdi textové pole so športom udalosti a text v ňom ulož do premennej
3. Nájdi textové pole s popisom udalosti a text v ňom ulož do premennej
4. Nájdi textové pole s dátumom a časom udalosti a text v ňom ulož do premennej
5. Nájdi textové pole so zemepisnou šírkou udalosti a text v ňom ulož do premennej
6. Nájdi textové pole so zemepisnou výškou udalosti a text v ňom ulož do premennej
7. Nájdi textové pole s mestom konania udalosti a text v ňom ulož do premennej

*Postconditions*:

- Porovnaj, či hodnota z uloženej premennej s názvom udalosti je zhodná s názvom udalosti inštancie predanej cez parameter do komponentu
- Porovnaj, či hodnota z uloženej premennej so športom udalosti je zhodná so športom udalosti inštancie predanej cez parameter do komponentu
- Porovnaj, či hodnota z uloženej premennej s popisom udalosti je zhodná s popisom udalosti inštancie predanej cez parameter do komponentu
- Porovnaj, či hodnota z uloženej premennej s dátumom a časom udalosti je zhodná s dátumom a časom inštancie predanej cez parameter do komponentu

- Porovnaj, či hodnota z uloženej premennej so zemepisnou šírkou je zhodná so zemepisnou šírkou udalosti inštancie predanej cez parameter do komponentu
- Porovnaj, či hodnota z uloženej premennej so zemepisnou výškou udalosti je zhodná so zemepisnou výškou udalosti inštancie predanej cez parameter do komponentu
- Porovnaj, či hodnota z uloženej premennej s mestom konania udalosti je zhodná s mestom konania udalosti inštancie, predanej cez parameter do komponentu

### 6.1.4 LocationFieldsTests

Testy v tejto triede testujú komponent LocationFields.

#### *6.1.4.1 FieldsAreReadOnly*

Test overuje, že textové pole zemepisnej šírky, zemepisnej výšky a mestom udalosti sú nastavené iba na čítanie.

*Testovacie kroky:*

1. Nájdi komponent LatitudeField a v ňom textové pole
2. Nájdi komponent LongitudeField a v ňom textové pole
3. Nájdi komponent CityField a v ňom textové pole

*Postconditions*:

- Over, že textové pole v komponente LatitudeField je nastavené iba na čítanie
- Over, že textové pole v komponente LongitudeField je nastavené iba na čítanie
- Over, že textové pole v komponente CityField je nastavené iba na čítanie

#### *6.1.4.2 LatitudeValueChanges*

Test overuje, že zemepisná šírka sa zmení v textovom poli, pokiaľ bola zmenená zemepisná šírka v udalosti, ktorá je ku komponentu viazaná.

*Testovacie kroky:*

1. Ulož do premennej hodnotu zemepisnej šírky z inštancie udalosti predanej do komponentu
2. Nájdi komponent LatitudeField, a ulož hodnotu jeho zemepisnej šírky do premennej
3. Zmeň zemepisnú šírku v udalosti na inú

*Postconditions*:

- Over, že premenné z kroku 1 a z kroku 2 sú zhodné
- Over, že hodnota zemepisnej šírky v komponente LatitudeField sa zmenila na zmenenú hodnotu zadanú v kroku 3

#### *6.1.4.3 LongitudeValueChanges*

Test overuje, že zemepisná výška sa zmení v textovom poli, pokiaľ bola zmenená zemepisná výška v udalosti, ktorá je ku komponentu viazaná.

*Testovacie kroky:*

1. Ulož do premennej hodnotu zemepisnej výšky z inštancie udalosti predanej do komponentu
2. Nájdi komponent LongitudeField, a ulož hodnotu jeho zemepisnej výšky do premennej
3. Zmeň zemepisnú výšku v udalosti na inú

*Postconditions*:

- Over, že premenné z kroku 1 a z kroku 2 sú zhodné
- Over, že hodnota zemepisnej výšky v komponente LatitudeField sa zmenila na zmenenú hodnotu zadanú v kroku 3

#### *6.1.4.4 CityValueChanges*

Test overuje, že mesto konania udalosti sa zmení v textovom poli, pokiaľ bolo zmenené mesto v udalosti, ktoré je ku komponentu viazané.

*Testovacie kroky:*

1. Ulož do premennej hodnotu mesta z inštancie udalosti predanej do komponentu
2. Nájdi komponent CityField, a ulož jeho hodnotu mesta do premennej
3. Zmeň mesto v udalosti na iné

*Postconditions*:

- Over, že premenné z kroku 1 a z kroku 2 sú zhodné
- Over, že hodnota mesta v komponente CityField sa zmenila na zmenenú hodnotu zadanú v kroku 3

### 6.1.5 PlayerDialogTests

Testy v tejto sade testujú komponent PlayerDialog.

#### *6.1.5.1 IsNotExpandedAfterCloseButtonClick*

V tomto teste sa overuje, či sa komponent po tlačidlo zatvoriť skutočne zatvorí.

*Testovacie kroky:*

1. Do premennej ulož hodnotu premennej, ktorá ukazuje, či je komponent otvorený
2. Nájdi tlačidlo zatvoriť a klikni naň
3. Do premennej ulož hodnotu premennej, ktorá ukazuje, či je komponent otvorený

*Postconditions:*

- Premenná uložená v kroku 1 pred kliknutím na tlačidlo zatvoriť značí, že komponent je otvorený
- Premenná uložená v kroku 3 po kliknutí na tlačidlo zatvoriť značí, že komponent je zatvorený

#### *6.1.5.2 AllPlayersAreListed*

Tento test overuje, či sú v zozname účastníkov v dialógu vypísaný všetci účastníci udalosti

*Testovacie kroky:*

1. Nájdi zoznam účastníkov udalosti, a ulož jeho text do premennej
2. Zisti, či sú v zozname vypísaný všetci účastníci, a výsledok ulož do premennej

*Postconditions:*

- Z uloženej premennej z kroku 2 vyplýva, že všetci účastníci udalosti sú v zozname vypísaný

## 6.2 Použité postupy

Testovacia knižnica bUnit je vcelku jednoduchá na používanie. Pri práci s ňou sme sa riadili podľa odporúčaní písania dobrých jednotkových testov. V ďalšej podkapitole si povieme o používaní knižnice bUnit vo vytvorenom projekte k testovaniu webovej aplikácie.

### 6.2.1 Používanie bUnit

**TestContext**

Zakaždým je potreba vytvoriť inštanciu triedy TestContext. Z tejto triedy potom vždy vyrenderujeme komponent, ktorý chceme testovať.

V rámci jednej triedy potom voláme ten istý kód na začiatku testu:

```
using var ctx = new TestContext();
var cut = ctx.RenderComponent<MyTestComponent>();
```

Kde do generického parametru metódy RenderComponent napíšeme triedu, ktorú chceme vyrenderovať. Vykreslený komponent si potom ukladáme do premennej, V prípade, že chceme do komponenty predať parameter, ktorý má komponent používať pre svoju správnu funkčnosť, musíme metódu zavolať nasledovne:

```
ctx.RenderComponent<EventButton>(parameters => parameters
   .Add(component => component.SportEv,
                 new SportEvent("New event", DateTime.Now, "Event description")));
```

Kde do metódy Add za prvý parameter dosadíme premennú, ktorej chceme predať hodnotu, a v druhom parametri metódy predávame samotnú hodnotu. Kódom vyššie teda hovoríme, že chceme vyrenderovať komponent EventButton, a chceme, aby tomuto komponentu bola predaná inštancia udalosti. Renderovanie komponentu je potrebné pre prácu s ním, keďže až po vyrenderovaní môžeme nájsť jednotlivé elementy v komponente a interagovať s nimi – napríklad nájsť tlačidlo aby sme naň klikli.

**TestAuthorization**

Pokiaľ by sme zároveň teste vyžadovali autentizáciu užívateľa v komponente, môžeme ju nasimulovať nasledovne:

```
var testAuthorizationContext = this.AddTestAuthorization();
testAuthorizationContext.SetAuthorized("myTestUser", AuthorizationState.Unauthorized);
```

V prvom riadku kódu vyššie hovoríme, že chceme pridať testovaciu autorizáciu, ktorú ukladáme to premennej testAuthorizationContext. Následne pomocou metódy

SetAuthorized, nastavíme, že chceme simulovať autorizáciu užívateľa s užívateľským menom TestUser, a chceme, aby mal autorizačný stav Authorized – autorizovaný.

V prípade, že by sme mu chceli pri autorizácii nastaviť ďalšie vlastnosti užívateľa, môžeme použiť metódu SetClaims, ako vidíme v kóde nižšie:

```
testAuthorizationContext.SetClaims(
   new Claim(JwtClaimTypes.Subject, Guid.NewGuid().ToString())
);
```

V príklade vyššie mu pomocou tejto metódy priradzujeme náhodný identifikátor GUID, pretože tento je potrebné definovať v niektorých testoch, napríklad v testovacej sade EventButtonTests – v teste JoinEventTest. V tomto teste prihlásený užívateľ vytvára požiadavku na pripojenie k udalosti, v ktorej posiela mimo iné aj svoje užívateľské GUID. Aj keď je odpoveď mockovaná, pretože je súčasťou tejto požiadavky GUID identifikátor, je nutné nejaký poskytnúť. Nezáleží pritom či je vymyslený, náhodný, alebo či vôbec existuje.

**Registrácia služby**

Ďalej je potrebné zaregistrovať služby, ktoré testy využívajú. Toto dosiahneme nasledujúcim riadkom kódu:

```
Services.AddScoped<IEventService, EventService>();
```

Aj keď služby priamo nevyužívame v testoch, pokiaľ v týchto testoch využívame komponent, ktorý volá túto službu, musíme takúto službu zaregistrovať pred tým, než vyrenderujeme komponent, ktorý službu využíva.

**Mockovanie služby HttpClient**

K mockovaniu služby HttpClient sme využili knižnicu RichardSzalaj.MockHttp, ako sa spomína na stránke bUnit. [19]

Nasledujúcim riadkom kódu vytvoríme testovacieho http klienta:

```
var mockHttpMessageHandler = Services.AddMockHttpClient();
```

Ďalej mu musíme povedať, ako má reagovať na požiadavky na jednotlivé koncové body:

```
mockHttpMessageHandler.When("/myEndpoint").RespondJson(new List<int> { 1,2,3,4});
```

Riadok vyššie hovorí, že ak komponent v teste zavolá koncový bod /myEndpoint, mockovaný HttpClient mu odpovie kolekciou List, v ktorej vráti mockovaný výsledok – čísla od 1 po 4.

Vďaka tomuto môžeme v testoch simulovať odpoveď zo strany servera bez toho, aby sme museli koncový bod priamo volať. Takýmto spôsobom si testy dokážu zachovať svoju izolovanosť.

**NavigationManager**

NavigationManager v Blazoru zodpovedá za querying a správu navigácie URI. Je možné vytvoriť si falošnú testovaciu verziu NavigationManagera v bUnit:

```
var navigationManager = ctx.Services.GetRequiredService<NavigationManager>();
```

Túto službu využívame v teste NotLoggedInTest, kde zisťujeme, či nás po kliknutí na tlačidlo navigovalo na správnu stránku. Akonáhle komponent využije službu NavigationManager pre presmerovanie na inú stránku, v našej testovaciej verzii si môžeme zistiť, na akú Uri užívateľa služba presmerovala.

### 6.2.2 Popis kódu testov

#### *6.2.2.1 ContextBase*

Trieda ContextBase slúži ako základná trieda pre vytvorenej webovej aplikácie. Rozširuje triedu TestContext, aby bolo možné využívať funkcionalitu triedy TestContext. Alternatívou by bolo vytvoriť v tejto triede inštanciu triedy TestContext, a jednotlivé metódy využívať cez túto inštanciu. Obe tieto možnosti sú spomenuté v oficiálnej dokumentácii bUnit.

```
public abstract class ContextBase<TComponent> : TestContext
where TComponent : ComponentBase
```

Môžeme si všimnúť, že definícia triedy ContextBase obsahuje generický parameter TComponent, ktorý musí byť typu ComponentBase – a teda musí sa jednať o Blazor komponent.

**SetupComponent**

Keďže bUnit je knižnica pre testovanie Blazor komponentov, vždy budeme chcieť testovať aspoň jeden komponent. Znamená to, že si testovacie triedy môžeme rozdeliť tak, že každá trieda bude testovať práve jeden komponent.

Testovacie triedy sa stávajú testovacími sadami pre daný komponent, a testovacie metódy samotnými testami v testovacej sade. V prípade, že by bolo potreba v teste vyrenderovať

ešte iný komponent, stále to bude možné vďaka tomu, že táto trieda ContextBase rozširuje triedu TestContext, a nadobúda tak všetkých vlastností, ktoré táto trieda poskytuje.

Pre účely renderovania komponentu sme vytvorili metódu SetupComponent:

```
protected abstract IRenderedComponent<TComponent> SetupComponent(params object[]
args);
```

Táto metóda komponent iba vykreslí. Dôležité však je, že táto metóda je abstraktná. Takto je umožnené definovať renderovanie komponentu až v triede potomka. Táto metóda je volaná vo vnútri metódy ComponentSetup, čo je metóda, ktorá ukladá komponent po jeho renderovaní do premennej. Túto premennú potom môžeme používať v testoch. Stačí, keď raz v triede definujeme, ako sa má komponent vykresliť. Keďže testujeme v celej triede ten istý komponent, ktorý má rovnaké požiadavky pre každý test v nej, nemusíme takýmto spôsobom písať kód pre vykreslenie komponentu v každom teste zvlášť, stačí raz v testovacej sade. Pokiaľ by sme komponentu pri renderovaní chceli predať v iných testoch v rámci triedy iné parametre, môžeme ich predať cez parameter componentArgs.

```
protected void ComponentSetup(params object[] componentArgs)
{
   Component = SetupComponent(componentArgs);
}
```

**SetupAuthorization**

Pomocou metódy SetupAuthorization vytvorenej v tejto triede obsluhujeme autorizáciu, v ktorej udávame aj užívateľský GUID. Vzhľadom na to, že vo väčšine komponentov, ktoré testy overujú je potrebné, aby mal autentizovaný užívateľ aj pridelený identifikátor GUID:

```
private readonly TestAuthorizationContext _testAuthorizationContext;
protected string TestUserName => "test-user";
...
protected void SetupAuthorization(AuthorizationState state)
{
   if (state is not AuthorizationState.Unauthorized)
   {
      _testAuthorizationContext.SetAuthorized(TestUserName, state);
      _testAuthorizationContext.SetClaims(
         new Claim(JwtClaimTypes.Subject, Guid.NewGuid().ToString())
      );
   }
}
```

**MockHttpResponse**

Vytvorili sme tiež metódu MockHttpResponses, ktorá nám jednoduchším zápisom namockuje požadované koncové body a vráti výsledok, aký požadujeme. Prvým parametrom do tejto metódy je koncový bod, aký chceme mockovať. Druhým parametrom je návratová hodnota, ktorá sa má z endpointu vrátiť.

```
protected void MockHttpResponse(string endpoint, object returnValue)
{
   _mockHttpMessageHandler.When(Localhost + endpoint).RespondJson(returnValue);
}
```

#### 6.2.2.2 Fake Data

Vzhľadom na to, že testy by mali byť izolované od databáze, ktorú vo webovej aplikácii využívame, vytvorili sme si služby, ktoré nám vrátia vymyslené dáta. V týchto triedach sa nachádza niekoľko metód, ktoré nám vrátia falošné výsledky. Jedným z týchto metód je aj GetEvent, ktorá nám vráti inštanciu udalosti, ktorú potom predávame do komponentov, ktoré ju potrebujú pre svoju funkčnosť.

```
 public async Task<SportEvent> GetEvent(RelationshipType type) =>
await Task.FromResult(new SportEvent()
{
   Sport = "SportName",
   Date = DateTime.Now.AddDays(5),
   Description = "Some description",
   Name = "Event name",
   Type = type,
   Location = await GetLocation(),
   Id = Guid.NewGuid()
});
```

#### 6.2.2.3 Testovacie sady

V triede konkrétnej testovacej sady sa už nachádza samotná implementácia testov pre daný komponent. Ukážka hlavičky triedy z testovacej sady EventButtonTests:

```
public class EventButtonTests : ContextBase<EventButton>
```

Z riadku vyššie si všimneme, že táto trieda dedí z triedy ContextBase, kde ako generický parameter predávame komponent EventButton. Ako sme si už povedali, týmto trieda ContextBase vie, aký komponent má vykresliť.

Pre vykreslenie komponentu definujeme metódu SetupComponent, kde pridáme komponente EventButton parameter inštancie udalosti.

```
protected override IRenderedComponent<EventButton> SetupComponent(params object[]
args)
{
   var ev = args[0] as SportEvent;
   return RenderComponent<EventButton>(parameters => parameters
      .Add(p => p.SportEv, ev)
   );
}
```

Na riadkoch nižšie je ukážka jedného z implementovaných testov – testu LeaveEventTest:

```
[Fact]
public async Task LeaveEventTest()
{
   var sportEvent = await FakeEvents.GetEvent(RelationshipType.Joined);
   SetupAuthorization(AuthorizationState.Authorized);
   ComponentSetup(sportEvent);
   MockHttpResponse(Endpoints.LeaveEvent, true);

   var btn = Component.Find("button");
   var beforeClickText = btn.GetInnerText();

   btn.Click();
   Component.WaitForElementTextToBe("button", JoinText);

   var afterClickText = btn.GetInnerText();

   Assert.Equal(beforeClickText,LeaveText);
   Assert.Equal(afterClickText,JoinText);
   Assert.Equal(RelationshipType.None,sportEvent.Type);
}
```

Všimneme si, že na začiatku testu voláme metódu GetEvent aby sme si získali inštanciu udalosti, kde ako parameter posielame, že užívateľ má byť k udalosti pripojený. Ďalej voláme SetupAuthorization s parametrom Authorized, aby sme simulovali prihláseného užívateľa.

Volaním funkcie MockHttpResponse mockujeme koncový bod Endpoints.LeaveEvent, a požadujeme, aby výsledok tejto požiadavky bol true – pravda, čo značí, že užívateľ sa úspešne pripojil k udalosti.

Ďalej v teste hľadáme tlačidlo pomocou metódy Find, a text tlačidla si ukladáme do premennej. Následne sa vykoná kliknutie na toto tlačidlo, a čakáme, či sa text prepíše na nami očakávaný text. Ak nie tak test zlyhá na tomto riadku.

Ak áno, test pokračuje ďalej a do premennej sa ukladá text tlačidla po kliknutí, a následne sa porovnáva, či:

- Text pred kliknutím zodpovedá textu 'odísť z udalosti'
- Text po kliknutí zodpovedá textu 'pripojiť sa k udalosti'
- Vzťah užívateľa k tejto udalosti sa prepísal na vzťah 'žiadny vzťah'

# ZÁVĚR

V tejto záverečnej práci sme vytvorili webovú aplikáciu pomocou webového frameworku Blazor, ktorá slúži jej užívateľom na nájdenie partnerov na športovanie. Užívateľ môže na stránke vytvoriť udalosť, kde zadá údaje ako sú napríklad názov udalosti, dátum a čas, aký šport sa bude hrať a lokalitu udalosti. Následne sa ostatný užívatelia k tejto udalosti môžu pripojiť. Popis v udalosti pomáha k tomu, aby zakladateľ udalosti povedal podrobnosti, ako napríklad čo si so sebou priniesť. Miesto udalosti v reálnom svete užívatelia nájdu vďaka interaktívnej mape so značkou miesta konania udalosti. Pri tvorbe tejto webovej aplikácie sme sa riadili doporučenými postupmi, ktoré sme v tejto práci vysvetlili.

Témou tejto práce bolo aj vypracovať automatizované testy na túto webovú aplikáciu. Automatizované testy boli vytvorené pomocou testovacej knižnice bUnit. Táto knižnica je určená pre testovanie komponentov vytvorených webovým frameworkom Blazor. V dokumentácii knižnice bUnit boli aj s príkladmi popísané riešenia problémov, na ktoré sme počas tvorenia testov prichádzali, akými sú napríklad autorizácia testového užívateľa, mockovanie HttpClienta, registrácia služieb a podobne.

Pri tvorení testov sme sa teda riadili návodmi v dokumentácii testovacej knižnice bUnit. Využili sme dedenia z triedy TestContext, a túto možnosť sme ďalej rozšírili ďalej podľa vlastného návrhu.

Výsledkom je niekoľko spoľahlivých testov, ktoré testujú komponenty za rôznych podmienok, či fungujú tak, ako by mali. Pri ďalších úpravách webovej aplikácie sa nám stalo, že keď sme spustili testy, nie všetky testy neprešli. Vďaka tomuto sme dokázali rýchlo a včas opraviť chybu, ktorú sme pri úprave webovej stránky nedopatrením zaniesli.

Testovacia knižnica bUnit však testuje iba komponenty na klientskej strane aplikácie. Zároveň nebolo možné niektoré testy jednoduchým spôsobom vytvoriť, kvôli obmedzeniam tejto knižnice – napríklad automatizované testy komponentu mapa, vzhľadom na to, že kód v komponente pre obsluhu mapy nie je napísaný v jazyku C#, ale v jazyku Javascript.

# SEZNAM POUŽITÉ LITERATURY

[1] REHKOPF, Max, c2022. Automated software testing. Atlassian | Software Development and Collaboration Tools [online]. Sydney: Atlassian [cit. 2022-05-13]. Dostupné z: https://www.atlassian.com/continuous-delivery/software-testing/automated-testing

[2] Developer tools, technical documentation and coding examples | Microsoft Docs [online], c2022. Redmond: Microsoft Corporation [cit. 2022-05-13]. Dostupné z: https://docs.microsoft.com/

[3] Selenium Tool Suite, c2011-2021. Tutorials List - Javatpoint [online]. Noida: Sonoo Jaiswal [cit. 2022-05-13]. Dostupné z: https://www.javatpoint.com/selenium-tool-suite

[4] Selenium [online], c2022. Chicago [cit. 2022-05-13]. Dostupné z: https://www.selenium.dev

[5] SINGH, Ravinder, c2013-2022. Selenium WebDriver Achitecture. Tools QA [online]. ToolsQA [cit. 2022-05-13]. Dostupné z: https://www.toolsqa.com/selenium-webdriver/selenium-webdriver-architecture/

[6] Selenium: Definition, How it works and Why you need it | BrowserStack, c2011-2022. Most Reliable App & Cross Browser Testing Platform | BrowserStack [online]. Mumbai: BrowserStack [cit. 2022-05-13]. Dostupné z: https://www.browserstack.com/selenium

[7] Online Tutorials Library [online], c2022. Hyderabad: Tutorials Point [cit. 2022-05-13]. Dostupné z: https://www.tutorialspoint.com/

[8] UNADKAT, Jash, c2011-2022. Selenium WebDriver Tutorial : Getting Started with Test Automation. Most Reliable App & Cross Browser Testing Platform | BrowserStack [online]. Mumbai: BrowserStack [cit. 2022-05-13]. Dostupné z: https://www.browserstack.com/guide/selenium-webdriver-tutorial

[9] Robot Framework [online]. Helsinki: Robot Framework [cit. 2022-05-13]. Dostupné z: https://robotframework.org

[10] BUnit: a testing library for Blazor components [online], c2022. Egil Hansen [cit. 2022-05-13]. Dostupné z: https://bunit.dev

[11] .NET | Free. Cross-platform. Open Source. [online], c2022. Redmond: Microsoft Corporation [cit. 2022-05-13]. Dostupné z: https://dotnet.microsoft.com/

[12] .NET Core Overview, c2022. TutorialsTeacher - Learn Technologies [online]. TutorialsTeacher [cit. 2022-05-13]. Dostupné z: https://www.tutorialsteacher.com/core/dotnet-core

[13] L, Richard, c2022. Introducing .NET 5. .NET Blog [online]. Redmond: Microsoft [cit. 2022-05-13]. Dostupné z: https://devblogs.microsoft.com/dotnet/introducing-net-5/

[14] What's New in .NET 6 – A Complete Overview Of Features In .NET 6.0, 24 Nov 2021n. l. Custom Software Development Company | Rishabh Software [online]. Vadodara: Rishabh Software [cit. 2022-05-13]. Dostupné z: https://www.rishabhsoft.com/blog/dotnet-6-features

[15] Entity Framework Tutorial [online], c2020. [cit. 2022-05-13]. Dostupné z: https://www.entityframeworktutorial.net

[16] Inversion of Control, c2022. TutorialsTeacher - Learn Technologies [online]. TutorialsTeacher [cit. 2022-05-13]. Dostupné z: https://www.tutorialsteacher.com/ioc/inversion-of-control

[17] JONES, Matthew, c2022. The Repository-Service Pattern with DI and ASP.NET 5.0. Exception Not Found [online]. Ghost & Maido [cit. 2022-05-13]. Dostupné z: https://exceptionnotfound.net/the-repository-service-pattern-with-dependency-injection-and-asp-net-core/

[18] RAE, Matt, c2019-2021. Atomic Design Principles & Methodology 101. Adobe: Creative, marketing and document management solutions [online]. California: Adobe [cit. 2022-05-13]. Dostupné z: https://xd.adobe.com/ideas/process/ui-design/atomic-design-principles-methodology-101/

[19] Mocking HttpClient, c2022. BUnit: a testing library for Blazor components [online]. Egil Hansen [cit. 2022-05-13]. Dostupné z: https://bunit.dev/docs/test-doubles/mocking-httpclient.html

# SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

| | |
|---|---|
| AUT | Application Under Test |
| CRUD | Create, Read, Update, Delete |
| EDM | Entity Data Model |
| ER | Entity Relationship |
| HTML | HyperText Markup Language |
| IoC | Inversion of Control |
| IoT | Internet of Things |
| OCR | Optical Character Recognition |
| ORM | Object-Relational mapping |
| QA | Quality Assurance |
| RC | Remote Control |
| W3C | World Wide Web Consortium |
| XML | Extensible Markup Language |

# SEZNAM OBRÁZKŮ

Obrázok 1 Nástroje Selenium [5] ....13
Obrázok 2 Selenium IDE ....15
Obrázok 3 Výber jazyku a frameworku pri exporte zo Selenium IDE ....15
Obrázok 4 Exportované kroky do jazyku C# s frameworkom NUnit ....16
Obrázok 5 Architektúra Selenium RC [4] ....17
Obrázok 6 Architektúra WebDriveru [8] ....18
Obrázok 7 Ukážkový kód v Robot Framework [9] ....21
Obrázok 8 Definícia kľúčových slov [9] ....22
Obrázok 9 Komponent Counter [10] ....24
Obrázok 10 Test komponent Counter [10] ....24
Obrázok 11 Jednotná platforma .Net 5 [13] ....26
Obrázok 12 Jednotná platforma .Net 6 [14] ....26
Obrázok 13 Blazor Server [2] ....29
Obrázok 14 Blazor WebAssembly [2] ....30
Obrázok 15 Vrstvy modelov tried [15] ....31
Obrázok 16 Architektúra Entity Frameworku [15] ....32
Obrázok 17 Dátový model entity [15] ....33
Obrázok 18 LINQ na Entity dotaz a spätná transformácia výsledku [15] ....34
Obrázok 19 Ukladanie zmien kontextu v EF API [15] ....34
Obrázok 20 Stránka vyhľadávania udalostí ....36
Obrázok 21 Prípad, kedy neboli vytvorené žiadne udalosti ....37
Obrázok 22 Stránka vytvorenia udalosti ....38
Obrázok 23 Notifikácia o vytvorení udalosti v pravom dolnom rohu ....38
Obrázok 24 Stránka detailu udalosti ....39
Obrázok 25 Komponent EventDatePicker ....40
Obrázok 26 Komponent PlayerTable ....41
Obrázok 27 Komponent PlayerListDialog ....41
Obrázok 28 EventCard komponent ....43
Obrázok 29 Entity Relationship Diagram ....45
Obrázok 30 Vzťah medzi typmi tried v Dependency Injection [16] ....46
Obrázok 31 Repository-service pattern [17] ....48
Obrázok 32 Repository-service pattern v navrhnutej webovej aplikácii ....48

Obrázok 33 Organizmus hlavička na stránke profilu [18]..........................................50

# SEZNAM TABULEK

Tabuľka 1 Rozdiely medzi Entity Framework 6 a Entity Framework Core [2][15] ..32