Srovnání frameworků MAUI a Flutter

Damián Romančík

Bakalářská práce
2023

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

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

Akademický rok: 2022/2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Damián Romančík**
Osobní číslo:	**A20017**
Studijní program:	**B0613A140020 Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Srovnání frameworků MAUI a Flutter**
Téma práce anglicky:	**Comparison of MAUI and Flutter Frameworks**

Zásady pro vypracování

1. Popište současný stav technologií pro vývoj mobilních aplikací.
2. Zaměřte se na frameworky Microsoft Maui a Flutter.
3. Navrhněte ukázkové aplikace vhodné pro srovnání frameworků Maui a Flutter.
4. Vytvořte navržené ukázkové aplikace a popište jejich klíčové části.
5. Srovnejte řešení v obou frameworcích.
6. Zhodnoťte dosažené výsledky včetně doporučení pro tvorbu nových aplikací.

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

Seznam doporučené literatury:

1. Flutter. Flutter documentation [online]. Mountain View: Google, 2015 [cit. 2022-11-23]. Dostupné z: https://docs.flutter.dev/.
2. Dart. Dart documentation [online]. Mountain View: Lars Bak and Kasper Lund, 2011 [cit. 2022-11-23]. Dostupné z: https://dart.dev/guides.
3. Microsoft Maui. .NET Multi-platform App UI documentation [online]. Redmond: Microsoft, 2022 [cit. 2022-11-23]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/maui/?view=net-maui-7.0.
4. C#. C# documentation [online]. Redmond: Microsoft, 2000 [cit. 2022-11-23]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/csharp/.
5. BISHOP, John. C# : návrhové vzory. Brno: Zoner Pres, 2010. ISBN 9788074130762.

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

Datum zadání bakalářské práce: **2. prosince 2022**
Termín odevzdání bakalářské práce: **26. května 2023**

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

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

Ve Zlíně dne 7. prosince 2022

Prohlašuji, že

- beru na vědomí, že odevzdáním 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 bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji 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 – 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 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 bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem 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 bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 17.5.2023

Damián Romančík, v.r.
podpis studenta

ABSTRAKT

Táto bakalárska práca sa zaoberá porovnaním frameworkov Microsoft Maui a Flutter. V teoretickej časti práce je popísaný súčasný stav technológií pre vývoj mobilných aplikácií a je zameraná na frameworky Microsoft Maui a Flutter. V praktickej časti práce sú vytvorené ukážkové aplikácie: TODO aplikácia, Google Maps aplikácia, Navigation menu aplikácia a Recipe aplikácia. Tieto aplikácie sú realizované a popísané ich kľúčové časti. Následne sú porovnané jednotlivé aplikácie v oboch frameworkoch. Na záver je sformulované celkové zhodnotenie oboch frameworku vrátane doporučení pre tvorbu nových aplikácií.

Kľúčové slová:

.NET MAUI, Flutter, C#, XAML, Dart, multiplatformové aplikácie, natívne aplikácie, vývoj mobilných aplikácií

ABSTRACT

This bachelor thesis compares Microsoft Maui and Flutter frameworks. The theoretical part of the work describes the current state of technology for mobile application development and focuses on the Microsoft Maui and Flutter frameworks. In the practical part of the work, sample applications are created: the TODO application, the Google Maps application, the Navigation menu application and the Recipe application. These applications are realized and their key parts are described. Subsequently, individual applications in both frameworks are compared. At the end, an overall evaluation of both frameworks is formulated, including recommendations for the creation of new applications.

Keywords:

.NET MAUI, Flutter, C#, XAML, Dart, multiplatform apps, native apps, mobile application development

Týmto by som chcel poďakovať vedúcemu práce Ing. et Ing. Erikovi Královi, Ph.D. za odborné vedenie, ochotu, cenné rady a konzultácie bakalárskej práce.

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

OBSAH

ÚVOD

V dnešnej dobe sú mobilné aplikácie neoddeliteľnou súčasťou každodenného života. S rozvojom technológií pre vývoj mobilných aplikácií sa zvyšujú požiadavky na rýchlosť a efektívnosť ich vývoja aplikácii. V rámci tejto problematiky sa rozvíjajú rôzne spôsoby vývoja aplikácií a aj rôzne technológie, ktoré uľahčujú vývoj mobilných aplikácií.

Cieľom práce je porovnať dva relatívne nové frameworky pre vývoj mobilných aplikácií a to framework .NET MAUI a framework Flutter. Teoretická časť práce poskytuje prehľad o vývoji mobilných aplikácií, pričom sú rozobrané natívne a multiplatformové spôsoby vývoja mobilných aplikácií. Ďalej je venovaná pozornosť inovatívnym technológiám pre mobilné aplikácie, ako je cloud computing, umelá inteligencia a progresívne webové aplikácie (PWA). V rámci frameworkov sú podrobnejšie analyzované frameworky .NET MAUI a Flutter, vrátane ich fungovania, kľúčových vlastností a bezpečnosti. Práca tiež obsahuje krátky popis vývojových prostredí, ako je Visual Studio a Android Studio, ktoré sú vhodné na použitie pri vývoji mobilných aplikácií pomocou frameworku .NET MAUI a Flutter. Praktická časť práce je zameraná na implementáciu vzorových aplikácii, ktoré slúžia na porovnanie frameworku .NET MAUI a Flutter. V rámci jednotlivých ukážkových aplikácii v obidvoch frameworkoch sú popísané kľúčové časti kódu, vykonané ich porovnanie a následne zhodnotenie. V závere praktickej časti práce je popísané celkové zhodnotenie frameworku a odporúčania pre tvorbu nových mobilných aplikácií na základe získaných poznatkov.

I. TEORETICKÁ ČÁST

1 VÝVOJ MOBILNÝCH APLIKÁCIÍ

Za začiatok vývoja mobilných aplikácií mnohí ľudia považujú uvedenie prvého Iphone a Android zariadenia. História vývoja mobilných aplikácií však siaha do histórie trochu ďalej. V roku 1994 bol na trh uvedený prvý smartfón IBM Simon so zabudovanými vlastnými aplikáciami. Kúpou smartfónu získali ľudia prístup k mobilným aplikáciám, ako je napríklad kalkulačka, zoznam kontaktov, hodiny a dokonca bola prítomná aj aplikácia predchádzajúca Google mapy. [1]

Ďalší skok v histórii mobilných aplikácii nastal v ďalšom desaťročí, keď bol v roku 2002 uvedený na trh smartfón BlackBerry s inovatívnym konceptom e-mailu. [2]

Predstavenie prvého iPhonu od spoločnosti Apple v roku 2007 a prvých smartfónov so systémom Android od spoločnosti HTC, v októbri 2008, bolo dôležitým míľnikom, ktorý odštartoval revolúciu vývoja mobilných aplikácií. [3][4]

Tieto prelomové udalosti priniesli nový prístup k mobilným zariadeniam, kde sa zameriavali na dotykové ovládanie a prijemné používateľské rozhrania, čo otvorilo dvere pre vznik nových a inovatívnych aplikácií. Tieto aplikácie sa viac zameriavali na užívateľskú prívetivosť a všeobecne na užívateľské rozhranie (UI). Od vydania úplne prvého smartfónu sa počet aplikácií neustále zvyšuje a zlepšuje ich kvalita a prepracovanosť. Dnes nám aplikácie uľahčujú mnoho činností a sú ľahko dostupné. Výrobcovia smartfónov začali uvádzať rôzne platformy a operačné systémy na trh, ktoré umožnili vývojárom vytvárať a distribuovať aplikácie pre rôzne zariadenia. Obchody s aplikáciami, ako App Store od Apple a Google Play Store pre Android, sa stali obľúbeným miestom, kde používatelia mohli objavovať a ľahko sťahovať rôzne aplikácie na svoje zariadenia. [5]

V posledných rokoch sa vývoj mobilných aplikácii rýchlo rozvíja. Dôvodom je stále rastúci počet používateľov mobilných zariadení a ich využívanie na rôzne účely. Veľkým míľnikom bol rozvoj internetu ktorý silne ovplyvnil vývoj mobilných aplikácií. Mobilné aplikácie sa dnes stali neoddeliteľnou súčasťou každodenného života väčšiny z nás a poskytujú rôzne služby nevyhnutné pre každodenný život, od správy účtov a bankovníctva až po zábavu a sociálne siete. Na základe rýchlo sa meniacich trendov vývojári prichádzajú s novými a inovatívnymi riešeniami, ktoré prispievajú k rozšíreniu možností používania mobilných aplikácií. Vývoj mobilných aplikácii sa neustále posúva dopredu vďaka rýchlemu rozvoju informačných a komunikačných technológií. Používajú sa na rôzne účely, od komunikácie cez sociálne siete, po sledovanie zdravia a fitness, až po prácu s údajmi a analytika.

V budúcnosti sa tiež očakáva, že vývoj mobilných aplikácií sa bude posúvať k väčšej integrácii rozšírenou realitou. To by mohlo prispieť k vytvoreniu nových možností pre mobilné aplikácie v oblastiach ako napríklad logistika, zdravotníctvo a automobilový priemysel. [6]

V súčasnosti existuje veľké množstvo rôznych typov mobilných aplikácií, od zábavných ako napríklad mobilné hry až po aplikácie na vzdelávanie. Mobilné aplikácie sú využívané nie len jednotlivcami, ale aj spoločnosťami na účel, ako napríklad aplikácie pre cestovanie, bankovníctvo a sociálne siete. [7]

Rozvoj mobilných aplikácií sa tiež stáva čoraz dôležitejším pre podniky, pretože tieto aplikácie im umožňujú zlepšiť komunikáciu so svojimi zákazníkmi a poskytovať im personalizované služby. Množstvo spoločností tiež využíva mobilné aplikácie na zlepšenie svojho marketingu a zvýšenie predaja. [8]

Jedným z hlavným trendov v oblasti mobilných aplikácií je využívanie umelej inteligencie a chatbotov, ktorí umožňujú aplikáciám lepšie sa prispôsobiť potrebám a preferenciám používateľov. Ďalším trendom je využívanie cloudových technológií umožňujúcich aplikáciám prácu s veľkými objemami dát a ďalšie pokročilé funkcie, ako napríklad synchronizáciu údajov medzi zariadeniami. V budúcnosti sa očakáva, že mobilné aplikácie budú ďalej rásť a rozvíjať sa vďaka pokračujúcemu vývoju nových technológií, ako napríklad 5G sietí a Internet of Things (IoT). Tieto technológie umožňujú mobilným aplikáciám vykonávať viac úloh a poskytovať viac funkcií, ako napríklad vysokorýchlostné prenosy dát, viac možností pre pokročilú analýzu a integráciu s rôznymi zariadeniami. Kombinácia týchto nových technológií s umelou inteligenciou a chatbotmi tiež poskytuje nové možnosti pre automatizáciu a personalizáciu služieb pre používateľov. V budúcnosti môžeme očakávať väčšiu inteligenciu a personalizáciu v mobilných aplikáciách, čo prispeje k ich ďalšiemu rozširovaniu a využitiu. [6]

V súčasnosti je kladený väčší dôraz na bezpečnosť a ochranu osobných údajov používateľov. Ochrana osobných údajov používateľov mobilných aplikácií je kľúčovou témou pre vývojárov i používateľov. Je dôležité aby vývojári tvorili aplikácie s bezpečným kódom, ktorý chráni osobné údaje pred neoprávneným prístupom, čo zahŕňa silné heslá, šifrovanie dát a používanie bezpečnostných protokolov. [9]

Vývoj mobilných aplikácii je proces tvorby softvéru pre smartfóny a digitálne zariadenia, najčastejšie pre Android a iOS. Mobilné aplikácie slúžia na poskytovanie podobných funkcií používateľom ako na desktop zariadení. [10]

Rozlišujeme dva základné prístupy k vývoju mobilných aplikácií: natívne a multiplatformové. Pri výbere akým spôsobom budeme vyvíjať mobilnú aplikáciu musíme myslieť na rôzne faktory. Pri voľbe prístupu vývoja aplikácie nie je možné jednoznačne určiť najlepší spôsob, lebo vývoj aplikácie je jedinečný v závislosti od konkrétneho projektu aplikácie.[11]

Medzi najdôležitejšie faktory, ktoré je dôležité zohľadniť pri výbere prístupu vývoja patrí:

- Funkcionalita
- Výkon
- Cenová politika
- Čas vývoja
- Bezpečnosť [11]

Okrem spomínaných faktorov závisí výber medzi natívnym a multiplatformovým vývojom od cieľov a požiadaviek konkrétneho projektu. Oba prístupy majú svoje výhody a nevýhody a výber závisí od konkrétnych potrieb a požiadaviek. [11]

Vývoj mobilných aplikácií je neustále sa meniaci proces, kde inovácie a technologické pokroky posúvajú hranice toho, čo je možné.

1.1 Natívne mobilné aplikácie

Natívne mobilné aplikácie sú aplikácie, ktoré sú vyvíjané pre konkrétnu platformu a sú optimalizované pre konkrétny operačný systém. Pri natívnom vývoji mobilných aplikácií je dôležitou informáciou to, pre aký operačný systém je mobilná aplikácia vyvíjaná. Najviac využívanými operačnými systémami sú práve Android alebo iOS. Aplikácie môžu byt vytvorené pomocou natívnych frameworkov ako je napríklad Swift, Java a Kotlin. [11]

Použitie natívneho vývoja mobilnej aplikácie má množstvo výhod. Medzi tie najdôležitejšie patrí:

- **Výkon** – pri zameraní na konkrétnu platformu je veľmi veľkou výhodou možnosť optimalizácie a možnosť kompilovať pomocou základného programovacieho jazyka. Tieto parametre nám umožňujú rýchlo reagovať na akcie užívateľa.
- **Zabezpečenie –** zvýšenie bezpečnosti údajov pomocou vstavaných bezpečnostných funkciám špecifických pre konkrétnu platformu.
- **Kvalitné a plynulé UX** – veľkou výhodou je dedenie rozhrania operačného systému zariadenia, ktoré nám zabezpečuje plynulý zážitok pre výstupy a vstupy od

užívateľa. Rovnako aj dojem a vzhľad pôsobia konzistentne. Taktiež sa riadia špecifickými návrhmi pre operačný systém, čo môže viesť k prirodzenejšiemu chodu aplikácie.

- **Úplný prístup k sade funkcií** - využívajú plný prístup k funkciám , čo prináša používateľom bohatšie a viac integrované používateľské prostredie. Zároveň získavajú prístup ku službám, ako sú push upozornenia, ktoré sú neoddeliteľnou súčasťou moderných aplikácii pre zlepšenie zapojenia používateľov.[12]

Pri uvažovaní o spôsobe vývoja aplikácie je nevyhnutné zvážiť aj nevýhody spojené s konkrétnym spôsobom. Medzi nevýhody natívneho vývoja patrí:

- **Náklady** - pre vývoj na viacerých platformách je nutné mať oddelené tímy vývojárov pre každú jednotlivú platformu.
- **Čas** – kvôli rozdielnym platformách pracujú samostatné tými na podobných aplikáciách, preto sa čas na vývoj natívnej aplikácie predlžuje.
- **Opakovateľnosť kódu** – pre účely vytvárania aplikácií pre rôzne mobilné operačné systémy je potrebné vytvárať a udržiavať kód v samostatných projektoch.[10]

1.2 Multiplatformové mobilné aplikácie

Pri multiplatformovom vývoji mobilných aplikácii nemusí byť určené pre aký operačný systém bude mobilná aplikácia vyvíjaná. Jedná sa o vytváranie mobilných aplikácií pre viaceré platformy z jedného zdrojového kódu. Aplikácie môžu byt vytvorené pomocou multiplatformových frameworkov ako je napríklad:

- React Native od spoločnosti Meta.
- Flutter od spoločnosti Google.
- Xamarin od spoločnosti Microsoft [13]

Použitie multiplatformového vývoja mobilnej aplikácie má množstvo výhod. Medzi tie najdôležitejšie patrí:

- **Nižšie náklady-** Vývoj mobilných aplikácií pre viac platforiem môže byt realizovaný len s jedným tímom vývojárov. Tento prístup zvyšuje dosah aplikácie.
- **Opakovateľnosť kódu-** Využitie rôznych vývojových nástroju pri vývoji mobilných aplikácií umožňuje opakované použitie kódu, čo zvyšuje produktivitu a umožňuje ľahšie udržiavanie aplikácie.

- **Rýchly vývoj-** Využitím medziplatformových nástrojov je možné dosiahnuť rýchlejšieho vývoja mobilných aplikácií pre viacero platforiem. Tieto nástroje taktiež optimalizujú proces testovania aplikácií počas vývoja.
- **Jednoduchšia údržba-** Aktualizácie aplikácie pre viacero platforiem je ľahšie vďaka jedinej kódovej základni. To umožňuje ľahšiu údržbu a aktualizáciu aplikácie na všetkých platformách. [13]

Pri použití multiplatformového prístupu je dôležité mať na pamäti aj nevýhody, ktoré sa s tým spájajú a ktoré môžu ovplyvniť celkovú kvalitu aplikácie a jej použiteľnosť. Medzi nevýhody multiplatformového vývoja:

- **Väčšia digitálna stopa -** Aplikácie písané pre viacero platforiem sú obvykle väčšie než natívne aplikácie pre jednu platformu. Vývojári by mali byť opatrní a venovať pozornosť dostupným možnostiam optimalizácie pre mobilné zariadenie.
- **Ťažká integrácia -** Multiplatformné frameworky nemusia obsahovať všetky funkcie špecifické pre danú platformu. Niektoré hardvérové integrácie, ako je používanie GPU, môžu vyžadovať zručnosti vo vývoji natívnych aplikácií.
- **Nižší výkon -** Frameworky pre viacero platforiem zvyčajne obsahujú vlastné prostredie, ktoré umožňuje spustenie aplikácie na rôznych platformách. To znamená, že komunikujú so službami špecifickými pre platformu, čo môže viesť k nižšiemu výkonu aplikácie.
- **Oneskorené funkcie platformy –** Pri vydaní nového vývojového balíku softvéru (SDK) je nutné počkať na aktualizáciu aby bolo možné využívať nové funkcie. [10]

1.3 Inovatívne technológie pre mobilné aplikácie

1.3.1 Cloud computing

Cloud computing sa stal neodmysliteľnou súčasťou nášho života a využíva ho väčšina populácie, často nevedomky. Mnoho webových a mobilných aplikácií, ako napríklad Gmail, je dnes postavených na cloudu. [14] Cloud computing umožňuje používateľom pristupovať k IT službám a zdrojom cez internet, čo im umožňuje flexibilnejšie a efektívnejšie využívať technológie. Tieto služby môžu zahŕňať úložisko dát, aplikácie a infraštruktúru. Používatelia platia len za to, čo skutočne využívajú, čo umožňuje organizáciám znížiť náklady na IT a zvýšiť flexibilitu. [15]

1.3.2 Umelá inteligencia

Umelá inteligencia predstavuje dôležitú súčasť mnohých moderných aplikácií. Ich využitie umožňuje automatizáciu procesov, zlepšovanie analýz a poskytovanie personalizovaných služieb používateľom. Používateľom umožňuje prispôsobenie sa ich potrebám a preferenciám, čo zvyšuje ich popularitu a význam. Všestranný a komplexný charakter umelej inteligencie otvoril svet inovácií a možností. [16]

Medzi najdôležitejšie technológie využívajúce umelú inteligenciu patrí:

- **Rozpoznávania reči** – Ide o technológiu umožňujúcu ovládať systém pomocou hlasových príkazov, ktorá zahŕňa schopnosť rozpoznávať rečový prejav. Najznámejším príkladom tohto systému je virtuálny asistent Siri.
- **Strojové učenie** – Technológia dokáže analyzovať preferencie používateľov a na základe získaných informácii prispôsobiť svoj obsah.
- **Biometria** – Pomocou tejto technológie dokáže identifikovať rôzne aspekty ľudského správania ako napríklad fyzické rysy, tvár a štruktúru tela. [17]

1.3.3 Progresívne webové aplikácie (PWA)

Veľké množstvo rozličných veľkostí displeja smartphonov a tabletov malo dopad na náročnosť tvorby a vývoja mobilných aplikácii. PWA je druh webovej aplikácie a jedná sa o jeden z najnovších trendov v oblasti internetových stránok. PWA sa snaží o to aby webová aplikácia vyzerala a dokonca sa aj správala ako mobilná aplikácia. Okrem toho, že PWA poskytujú používateľom lepší zážitok, tiež pomáhajú zlepšiť Optimalizáciu pre vyhľadávače (SEO) a znižujú náklady na vývoj a údržbu. [18]

Progresívne webové aplikácie sú novým typom webových aplikácií, ktoré využívajú špeciálne technológie a štandardy. Tieto technológie a štandardy umožňujú PWA využívať výhody webových aj natívnych funkcií, čo umožňuje vysokú interaktivitu a rýchlosť používania. PWA sú ľahko dostupné a objaviteľné prostredníctvom vyhľadávačov a môžu byť zdieľané jednoduchým odoslaním odkazu. Okrem toho je návšteva webovej stránky rýchlejšia a jednoduchšia ako inštalácia aplikácie. [19]

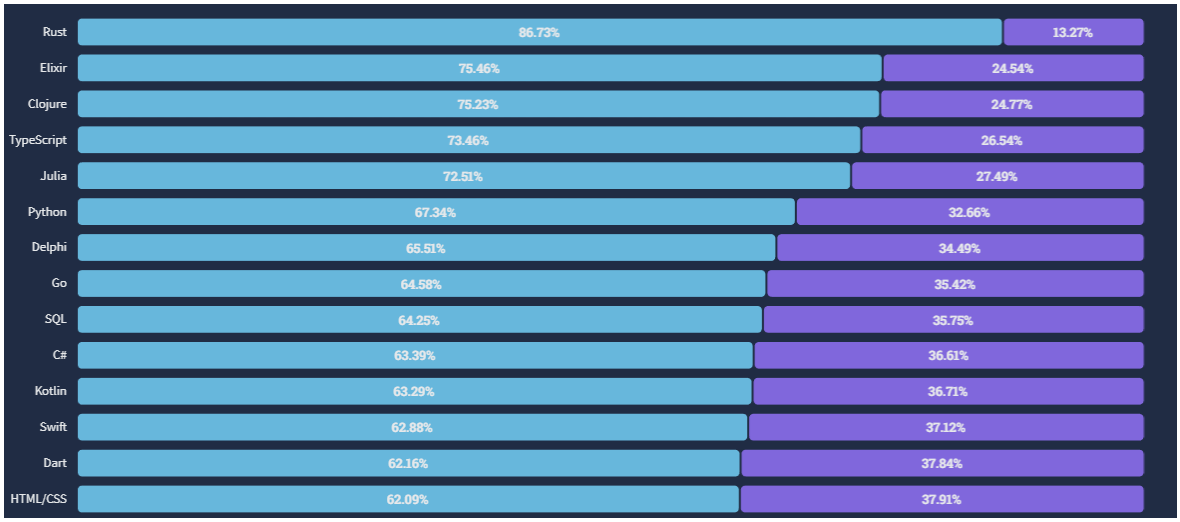
Existuje niekoľko kľúčových princípov, ktoré by sa webová aplikácia mala snažiť dodržiavať, aby bola identifikovaná ako PWA. Medzi tieto princípy patri:

- **Viditeľnosť** - obsah môže byť zahrnutý do vyhľadávania a zobrazený výsledkami vyhľadávania.

- **Prepojiteľnosť** – je možné aplikáciu zdieľať pomocou adresy URL
- **Offline** - dokáže fungovať aj bez pripojenia na internet alebo aj s slabým sieťovým pripojením.
- **Inovatívnosť** – aplikácia musí byť stále použiteľná s základnými funkcionalitami na starších prehliadačoch, ale plne funkčná na najnovších prehliadačoch.
- **Responzivita** – aplikácia by mala byť použiteľná na akomkoľvek zariadení s obrazovkou a prehliadačom ako sú mobilné telefóny, tablety, notebooky a televízory.
- **Zabezpečené** - spojenie medzi používateľom, aplikáciou a serverom sú zabezpečené pred akýmikoľvek tretími stranami, ktoré sa snažia získať prístup k citlivým údajom. [19]

2 FRAMEWORKY

Framework predstavuje štruktúru, ktorá sa používa ako základ pre vývoj softvéru. Jeho cieľom je zjednodušenie vývoja a zefektívnenie práce vývojárov. Frameworky poskytujú predefinované riešenia pre časté úlohy a problémy, ktoré sa vyskytujú v danom jazyku. [20]



Obrázok 1 Obľúbenosť programovacích jazykov [21]

C# a Dart sú moderné objektovo orientované programovacie jazyky, ktoré sa stali populárnymi medzi vývojármi v posledných rokoch (viď Obrázok 1). Oba jazyky majú veľkú komunitu vývojárov, bohatú dokumentáciu a širokú podporu nástrojov. Sú populárne pre svoju efektívnosť a použitie vo viacerých aplikáciách, ako sú desktopové aplikácie, webové aplikácie, mobilné aplikácie a hry. Oba jazyky získali dôveru a obľúbenosť vývojárskej komunity pre ich výkonnosť, syntax a širokú škálu použitia v modernom softvérovom vývoji. [21]

2.1 .NET MAUI

.NET Multi-platform App UI (.NET MAUI) je multiplatformová architektúra pre vytváranie natívnych mobilných a desktopových aplikácií pomocou C# a XAML. Pomocou .NET MAUI je možné vyvíjať aplikácie, ktoré môžu byť spustené na Android, iOS, macOS a Windows zariadení z jedného zdieľaného zdrojového kódu. [22]

.NET MAUI je opensourcový framework, ktorý sa rozvíja z Xamarin.Forms a umožňuje vytváranie natívnych mobilných a desktopových aplikácií pomocou C# a XAML. Napriek tomu že má .NET MAUI určité podobnosti s pôvodným Xamarin.Forms frameworkom, výrazné rozdiely medzi nimi sú viditeľné. Vývojári môžu použiť jeden projekt a jeden kód pre všetky platformy, čo uľahčuje vývoj a udržiavanie aplikácie. Ale v niektorých prípadoch je

potrebné pridať špecifický kód alebo zdroj pre konkrétnu platformu, je možné pridať napríklad prispôsobenie rozloženia užívateľského rozhrania (UI) alebo prístup k platformovo špecifickým funkciám. .NET MAUI poskytuje možnosť pridať tieto platformovo špecifické komponenty do projektu, takže vývojári môžu optimalizovať svoju aplikáciu pre každú platformu. [22]

Veľkou výhodou .NET MAUI je to, že podporuje opätovného načítanie zdrojového kódu za prevádzky, to znamená že je možné upravovať zdrojový kód za prevádzky bez toho, aby sme museli ručne pozastaviť beh aplikácie. [22]

2.1.1 Ako funguje .NET MAUI

.NET MAUI je nový framework umožňujúci vývojárom vytvárať multiplatformové aplikácie pomocou jazyka C# a XAML. Aplikácie vytvorené s .NET MAUI môžu byť spustené na rôznych platformách ako iOS, Android, macOS, a Windows. To znamená, že vývojári môžu použiť jeden kód pre viacero platformných cieľov, čo uľahčuje proces vývoja aplikácie. [10]

.NET MAUI je založený na Xamarin.Forms, ktorý umožňuje vývojárom používať jednotné rozhranie pre viac platformných cieľov, ako napríklad iOS, Android a Universal Windows Platform (UWP) a macOS. Týmto umožňuje využívať jeden kód pre viacero platformových cieľov a uľahčuje vývoj aplikácie. [22]

.NET MAUI tiež poskytuje nástroje pre optimalizáciu aplikácie pre jednotlivé platformy. To znamená, že vývojári môžu prispôsobiť aplikáciu pre rôzne veľkosti obrazovky a rôzne funkcie pre jednotlivé platformy. Napríklad, je možné prispôsobiť rozhranie pre dotykové gestá na mobilných zariadeniach. [22]

Aplikácie pre Android vytvorené s použitím .NET MAUI sa kompilujú z jazyka C# do jazyka IL, ktorý sa pri spustení aplikácie znovu kompiluje do natívneho kódu. Aplikácie pre iOS vytvorené s použitím .NET MAUI sú predkompilované z C# do natívneho kódu pre architektúru Advanced RISC Machine (ARM). Aplikácie pre macOS vytvorené s použitím .NET MAUI využívajú Mac Catalyst od spoločnosti Apple, ktorý umožňuje previesť aplikáciu pre iOS vytvorenú pomocou UIKit na plochu a prípadne ju rozšíriť o ďalšie rozhrania AppKit a API platformy. Aplikácie pre Windows vytvorené s použitím .NET MAUI využívajú knižnicu Windows UI 3 (WinUI 3) na vytváranie natívnych aplikácií pre Windows. [22]

2.1.2 Data binding and MVVM

MVVM je návrhový vzor používaný v oblasti používateľského rozhrania, ktorý umožňuje oddelenie kódu používateľského rozhrania od ostatných častí aplikácie. Tento vzor umožňuje vytvárať deklaratívne používateľské rozhrania pomocou XAML a potom viazať tieto rozhrania na dáta a príkazy pomocou štandardných označení. Dátové väzby vytvárajú voľné prepojenia, ktoré udržiavajú synchronizáciu medzi používateľským rozhraním a dátami. Tým, že sa vytvoria voľné prepojenia, používanie dátových väzieb znižuje tvrdé závislosti medzi rôznymi časťami kódu. Tento prístup uľahčuje úpravy jednotlivých kusov kódu, ako napríklad metód, tried a ovládacích prvkov, bez toho, aby tieto úpravy mali nežiadúce dôsledky v iných častiach kódu. Oddelenie záujmov, ktoré je kľúčovým konceptom mnohých návrhových vzorov, je v tomto prípade dôležitým prvkom vzoru MVVM.[23]

Výhody oddelenia kódu:

- Podporuje iteratívny, preskúmavajúci štýl programovania. Zmeny, ktoré sú izolované, sú menej rizikové a ľahšie sa s nimi experimentuje.
- Zjednodušuje unit testing. Jednotky kódu, ktoré sú izolované od seba, je možné testovať samostatne.
- Podporuje spoluprácu tímu. Decentralizovaný kód, ktorý sa riadi dobre navrhnutými rozhraniami, môže byť vyvíjaný oddelenými jednotlivcami alebo tímy.
- Zlepšuje udržiavateľnosť, oprava chýb v decentralizovanom kóde má menšiu pravdepodobnosť, že spôsobí regresie v inom kóde. [24]

V porovnaní s MVVM používa aplikácia s konvenčnejšou štruktúrou "code-behind" väčšinou dátové viazanie iba na zobrazenie údajov a priamo spracováva vstup používateľa udalostiam vystavenými ovládacími prvkami. Obslužné rutiny týchto udalostí sú umiestnené v súboroch s kódom na pozadí. Ovládacie prvky zvyčajne obsahujú kód, ktorý priamo upravuje používateľské rozhranie, čo robí ťažšie alebo nemožné nahradenie ovládacieho prvku bez nutnosti aktualizovať kód spracovávajúci udalosti. Tento prístup často vedie k hromadeniu kódu v súboroch s kódom, ktorý nesúvisí priamo s používateľským rozhraním, napríklad kód pre prístup k databáze, ktorý sa neskôr duplikuje a upravuje pre použitie v iných oknách.[24]

2.1.3 Bezpečnosť frameworku .NET MAUI

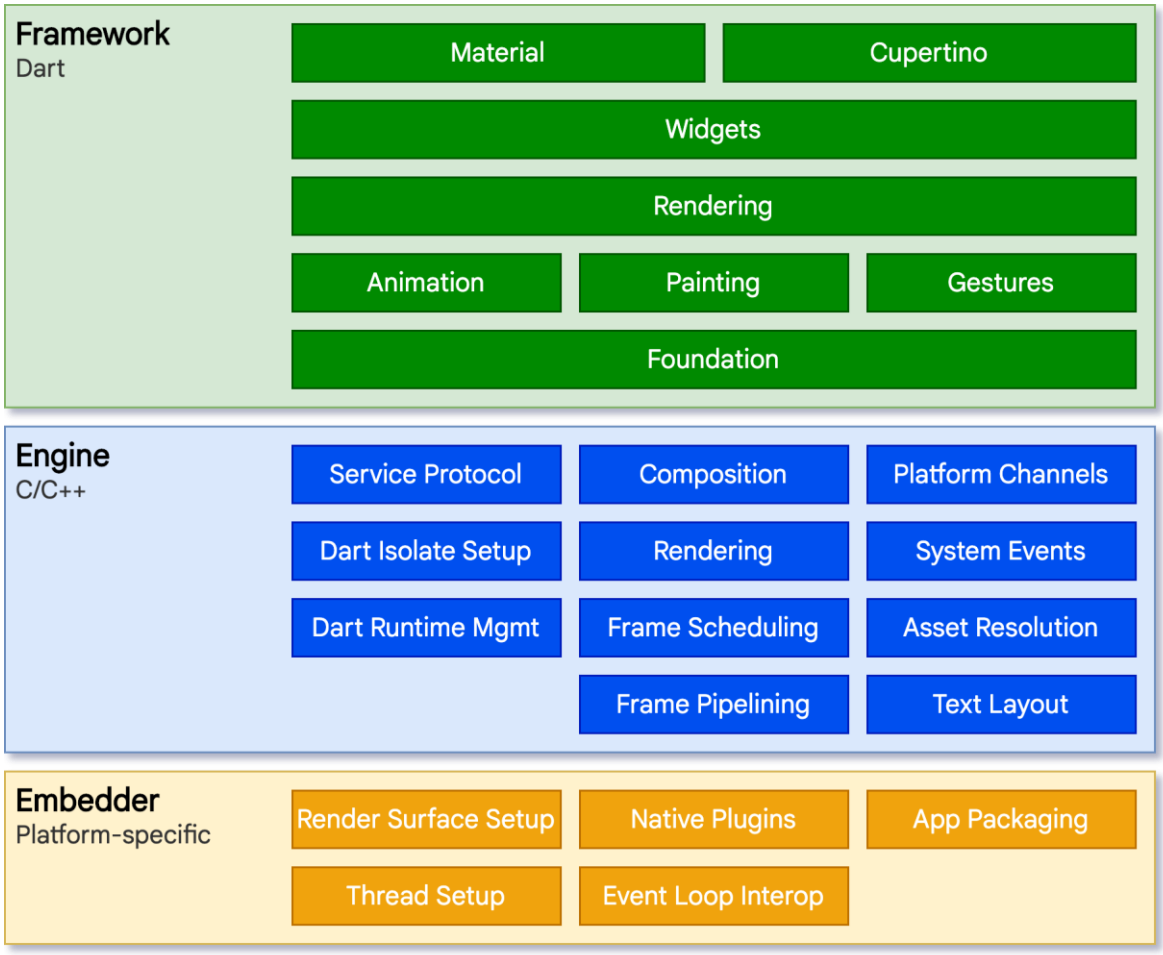
Framework .NET MAUI ponúka niekoľko možností na zabezpečenie aplikácií. Jednou z nich je rozhranie ISecureStorage, ktoré umožňuje bezpečné ukladanie párov kľúč/hodnota. Ďalšou možnosťou je rozhranie IWebAuthenticator, ktoré umožňuje spustenie prehliadačových autentifikačných tokov a počúvanie na spätné volanie na špecifickú URL zaregistrovanú pre aplikáciu. [25][26]

2.2 Flutter

Flutter je moderný framework, ktorý bol vyvinutý spoločnosťou Google a slúži na tvorbu multiplatformových mobilných aplikácií. Jeho hlavnou výhodou je to, že umožňuje vytvárať natívne aplikácie pre viaceré platformy z jediného zdrojového kódu. Tento open-source nástroj ponúka vývojárom intuitívne rozhranie, ktoré umožňuje rýchlejší a efektívnejší vývoj aplikácií. Vďaka svojej flexibilite a výkonnosti je Flutter ideálnym riešením pre tvorbu atraktívnych a výkonných mobilných aplikácií. Flutter sa líši od ostatných populárnych frameworku tým, že nie je len ďalším frameworkom alebo knižnicou. Miesto toho, ide o kompletnú sadu vývojového softvéru, známu ako SDK, ktorá poskytuje všetky nástroje a prostriedky potrebné pre vývoj plnohodnotných aplikácií. [27]

Flutter SDK je rozsiahla súprava nástrojov, ktorá zahŕňa viacero prvokov, ako sú knižnice, dokumentácia, rozhrania API a niekedy aj frameworky. Vďaka tomuto širokému spektru poskytuje všetky potrebné prostriedky pre vývoj softvéru a zjednodušuje tento proces pre vývojárov. Flutter používa programovací jazyk Dart a má bohatú knižnicu widgetov pre vytváranie užívateľského rozhrania. Knižnica predstavuje opakovane použiteľný blok kódu, ktorý je možné integrovať do aplikácie s cieľom vykonať špecifickú funkciu. Ide o spôsob, ako opätovne využiť existujúci kód bez nutnosti písania nového kódu od základu. [28]

2.2.1 Ako funguje Flutter



Obrázok 2 Architektonické vrstvy frameworku Flutter [29]

Hlavné architektonické vrstvy sú zobrazene na obrázku 2.

- **Embedder** - Jazyk, ktorý Flutter používa, je prispôsobený pre túto platformu a umožňuje vývojárom spúšťať aplikácie na akomkoľvek operačnom systéme.
- **An engine -** Flutter obsahuje engine napísaný v jazykoch C/C++, ktorý poskytuje nízkoúrovňovú implementáciu základných API pre túto platformu. Tieto API zahŕňajú grafiku, ktorá je zabezpečená pomocou Skia 2D grafickej knižnice, rozloženie textu, súborový a sieťový vstup/výstup, prístupnosť, architektúru zásuvných modulov a aj nástroje na runtime a kompiláciu jazyka Dart.
- **Framework –** Je postavený na jazyku Dart. Implementácia frameworku je dobrovoľná, ale ponúka bohatú kolekciu knižníc, ktoré sú rozdelené do vrstiev: základné triedy, vykresľovacia vrstva, widgetová vrstva a knižnice Material/Cupertino. [29]

Dart je základnou technológiou pre Flutter a je to objektovo orientovaný programovací jazyk, ktorý je optimalizovaný pre klientské prostredie a bol vyvinutý spoločnosťou Google. Dart môže byť skompilovaný do natívneho kódu pre mobilné zariadenia a počítače, ako to je aj u JavaScriptu. Vďaka tejto priamej kompilácii nie je potrebný ďalší most na komunikáciu s platformou, ako napríklad to je v prípade ReactNative. To znamená, že Flutter aplikácie majú rýchly čas spustenia a vysoký výkon. [28]

2.2.2 Widgety

Samotná obrazovka aplikácie vo Flutteri sa skladá z rôznych komponentov, ktoré sú nazývané widgety. Každý z týchto widgetov predstavuje jednu časť obrazovky a ich usporiadanie a výber ovplyvňuje celkový vzhľad a funkčnosť aplikácie. Kódu aplikácie sa hovorí "strom widgetov", nakoľko každý widget môže mať svoje vlastné podwidgety a celá štruktúra tvorí hierarchiu. [30]

Widgety predstavujú dôležitú hierarchiu tried. Widget predstavuje nemenný opis určitej časti používateľského rozhrania. Tieto widgety môžu byť vykreslené ako prvok na obrazovke, ktorý je súčasťou podkladového stromu renderovania. Widgety samotné nie sú schopné uchovávať zmenu. Na pridanie schopnosti widgetu uchovávať zmeny, je nutné použiť StatefulWidget [31].

Daný widget môže byť zahrnutý do stromu nula krát alebo viackrát. Pri každom umiestnení widgetu do stromu sa vykresľuje ako prvok na obrazovke. To znamená, že ak je komponenta viackrát vložená do stromu, bude vykreslená na prvok na obrazovke viackrát. [32]

Kľúčová vlastnosť riadi, ako jeden widget nahradí iný widget v strome. Ak sú vlastnosti runtimeType a kľúčové vlastnosti dvoch komponentov rovnaké, nový komponent nahradí starú komponentu aktualizáciou základného prvku. V opačnom prípade sa starý prvok zo stromu odstráni, nový widget sa vykreslí na prvok na obrazovek a nový prvok sa vloží do stromu. [32]

2.2.2.1 StatelessWidget

Stateless widget opisuje určitú časť používateľského rozhrania tým, že kombinuje iné widgety, ktoré sa zameriavajú na konkrétne aspekty tohto rozhrania. Tento proces sa môže opakovať rekurzívne, kým nie je opis rozhrania kompletný. Stateless widgety sú užitočné v prípadoch, keď časť používateľského rozhrania nezávisí na ničom inom, okrem konfiguračných informácií, ktoré sú uložené v samotnom objekte a v BuildContext. Ak sa však môžu

dynamicky meniť - napríklad v závislosti na stavoch riadených internými hodinami alebo stavoch systému, potom by bolo vhodnejšie použiť StatefulWidget. [33]

2.2.2.2 StatefulWidget

Stateful widget popisuje určitú časť používateľského rozhrania pomocou kombinácie iných widgetov, ktoré sa zameriavajú na konkrétne prvky tohto rozhrania. Proces tvorby pokračuje rekurzívne, až kým sa popis rozhrania nestane úplne konkrétnym. Stateful widgety sú užitočné, keď sa daná časť používateľského rozhrania môže dynamicky meniť, napríklad v závislosti na stavoch riadených vnútornými hodinami alebo stavoch systému. [31]

2.2.3 Bezpečnosť frameworku Flutter

Bezpečnostná stratégia frameworku Flutter je založená na kľúčových pilieroch, medzi tie najdôležitejšie patria:

- **Identifikovať** – Venovať pozornosť identifikácii základných aktív, kľúčových hrozieb a slabých miest.
- **Detekovať** - Efektívne odhaliť a identifikovať slabé miesta prostredníctvom použitia rôznych techník a nástrojov ako fuzzing, statické testovanie a skenovanie zraniteľnosti.
- **Chrániť** - Zabezpečenie kritických aktivít, eliminovať riziká, ktoré vznikajú z dôvodu existencie známych slabých miest. [34]

Osvedčené postupy pre minimalizovanie rizík bezpečnosti je udržanie si aktuálnych verzii Flutter SDK a udržiavanie aktuálnych závislostí v aplikáciách. [34] Okrem toho framework Flutter má balíček flutter_secure_storage, ktorý umožňuje ukladať dáta do bezpečného úložiska. Na platforme Android sa šifrujú kľúče a hodnoty použitím AES šifrovania na generovanie kľúča ktorý je zašifrovaný pomocou RSA šifry. Na platforme iOS sa na ukladanie a prístup pomocou kryptografických kľúčov používa KeyChain, ktorý je špecifický pre iOS. [35]

3 VÝVOJOVÉ PROSTREDIE

Pre vývoj Flutter aplikácií je možné použiť rôzne vývojové prostredia, ako napríklad Android Studio alebo Visual Studio Code. [36] Pre vývoj .MAUI aplikácií je nutné použiť Visual Studio 2022 vývojové prostredie.[37]

Tieto integrované vývojové prostredia (IDE) poskytujú vývojárom rôzne nástroje na zjednodušenie práce, ako napríklad nástroje na refactorovanie kódu, sledovanie závislostí, odladťovanie a ďalšie. Okrem toho sa vývojári môžu spoliehať aj na množstvo dostupných balíčkov a knižníc, ktoré umožňujú rýchly a jednoduchý vývoj aplikácií. [38]

3.1 Visual Studio

Visual Studio je jedným z nástrojov, ktoré sú dôležité pre vývoj softvéru. Tento program disponuje mnohými funkciami, ktoré umožňujú programátorom ľahko upravovať, ladiť a zostavovať kód a nakoniec publikovať aplikáciu. Medzi tieto funkcie patrí štandardný editor a debugger, ale aj kompilátory, dokončovanie kódu, grafické návrháre a ďalšie nástroje, ktoré môžu zlepšiť proces vývoja softvéru. Visual Studio poskytuje širokú paletu funkcií a nástrojov pre vývojárov. [39]

Visual Studio je integrované vývojové prostredie vyvinuté spoločnosťou Microsoft pro tvorbu grafického užívateľského rozhrania (GUI), konzolových aplikácií, webových aplikácií, mobilných aplikácií, cloudových a webových služieb a ďalších softwarových projektov. Toto IDE podporuje ako spravovaný kód, tak natívny kód, a využíva rôzne platformy Microsoftu pro vývoj softwaru, vrátane Windows Store, Microsoft Silverlight a Windows API. Visual Studio nie je zamerané na konkrétny programovací jazyk, a preto je možné s ním písať kód v C#, C++, Pythone, JavaScriptu a mnoho ďalších jazykoch. IDE je k dispozícii pre operačné systémy Windows aj macOS. Vývoj Visual Studia začal v roku 1997, kedy bola vydaná prvá verzia s názvom Visual Studio 97, verzia 5.0. [39]

3.2 Android Studio

Android Studio je oficiálne integrované vývojové prostredie pre vývoj aplikácií pre operačný systém Android. Toto IDE je založený na softvéri JetBrains IntelliJ IDEA a využíva zostavovací systém založený na Gradle, emulátory, šablóny kódu a integráciu GitHub. Jedná sa o rýchly a funkciami bohatý emulátor s jednotným prostredím, kde je možné vyvíjať aplikácie pre všetky zariadenia s operačným systémom Android, s možnosťou aplikovať zmeny

v kóde a prostriedkoch priamo v spustenej aplikácii bez nutnosti reštartovania. Okrem toho poskytuje rozsiahle testovacie nástroje a frameworky, ktoré umožňujú rýchlo a jednoducho otestovať zvolenú aplikáciu. Lint nástroje sú vhodné na zachytenie problémov s výkonom, použiteľnosťou a kompatibilitou s rôznymi verziami, ako aj s ďalšími problémami. [40]

II. PRAKTICKÁ ČÁST

4 VZOROVÉ APLIKÁCIE

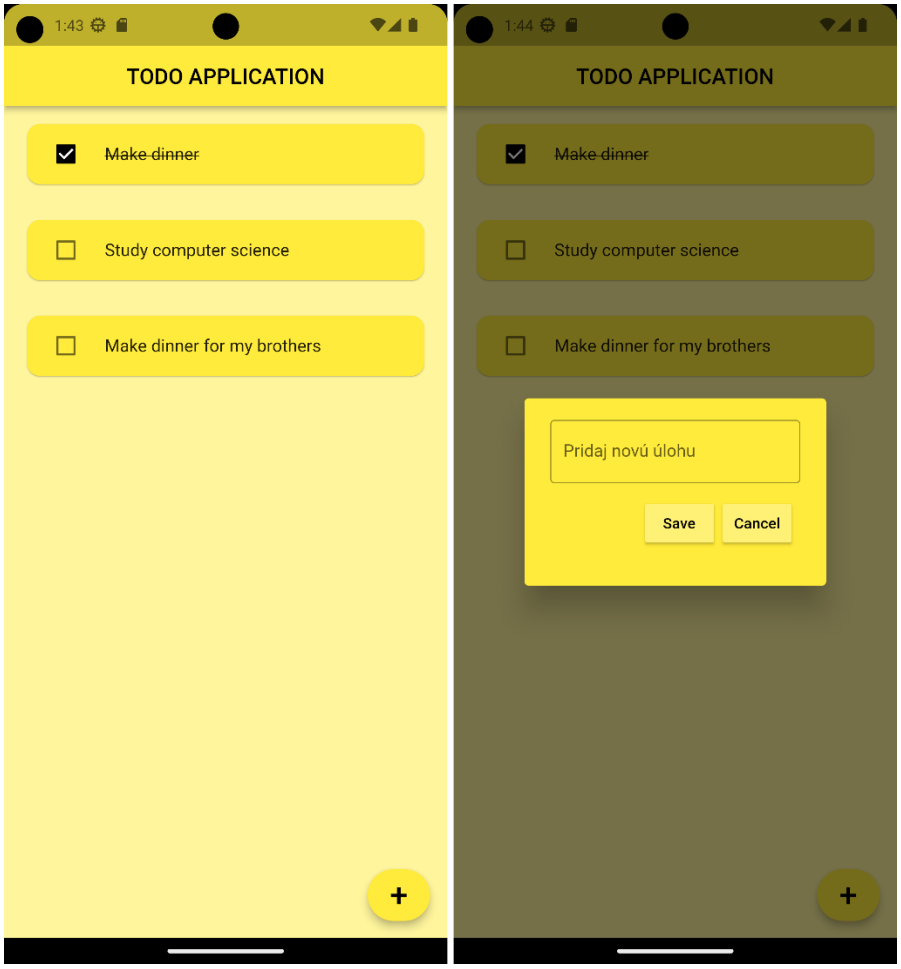
Cieľom je vytvoriť štyri vzorové aplikácie pre framework Flutter a štyri vzorové aplikácie pre framework .NET MAUI. Následne budú popísané jednotlivé kľúčové časti kódu a porovnajú sa riešenia pre oba frameworky pre každú aplikáciu. Popíšu sa prípadné obmedzenia frameworkov pri vytváraní daných aplikácií a celkovo sa zhodnotí použitie oboch frameworkov vrátane odporúčaní pre tvorbu nových aplikácií.

Vzorové aplikácie budú zahŕňať TODO aplikáciu na zobrazenie úloh s možnosťou vytvárania a mazania úloh, Google Maps aplikáciu na zobrazovanie mapy a vyhľadávanie miest, Navigation aplikáciu s postranným menu a možnosťou prechodu na rôzne obrazovky a Recipe aplikáciu na prezeranie receptov s možnosťou zobrazenia detailov receptu.

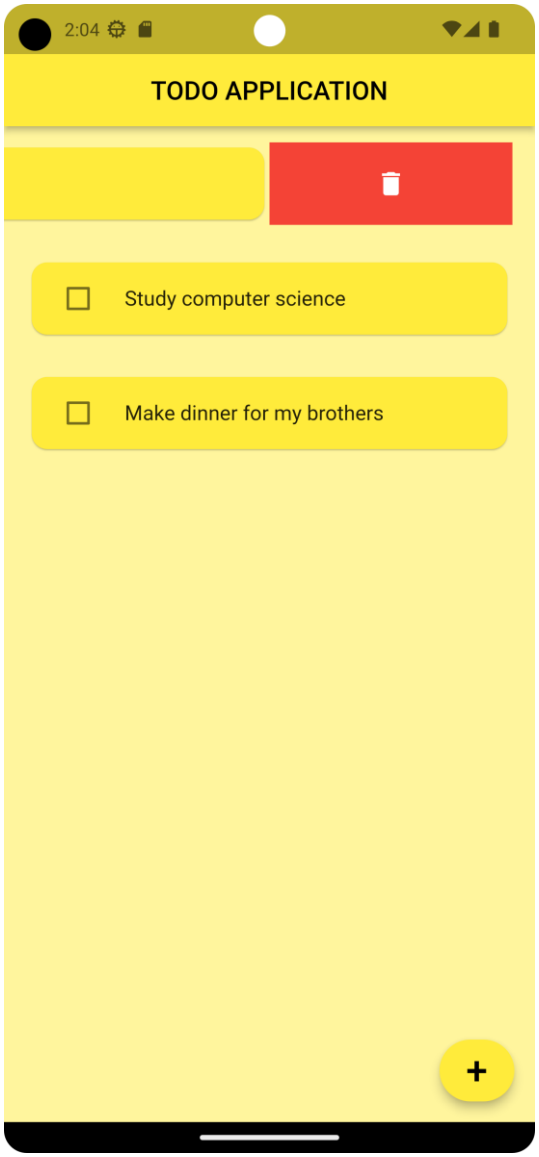
4.1 TODO APPLIKÁCIA

Cieľom tejto aplikácie bude vytvorenie zoznamu, kde budú zobrazené jednotlivé úlohy. Budeme pracovať s lokálnou databázou. Aplikácia bude zahŕňať funkcie ako pridávanie nových úloh do zoznamu, odstraňovanie úloh zo zoznamu a zmena stavu jednotlivých úloh. Aplikácia obsahuje dialógové okno, hlavnú stránku pre zobrazenie úloh a tlačidlo na pridanie úlohy.

Užívateľské rozhranie pre Flutter je vyobrazené na obrázkoch 3 a 4.

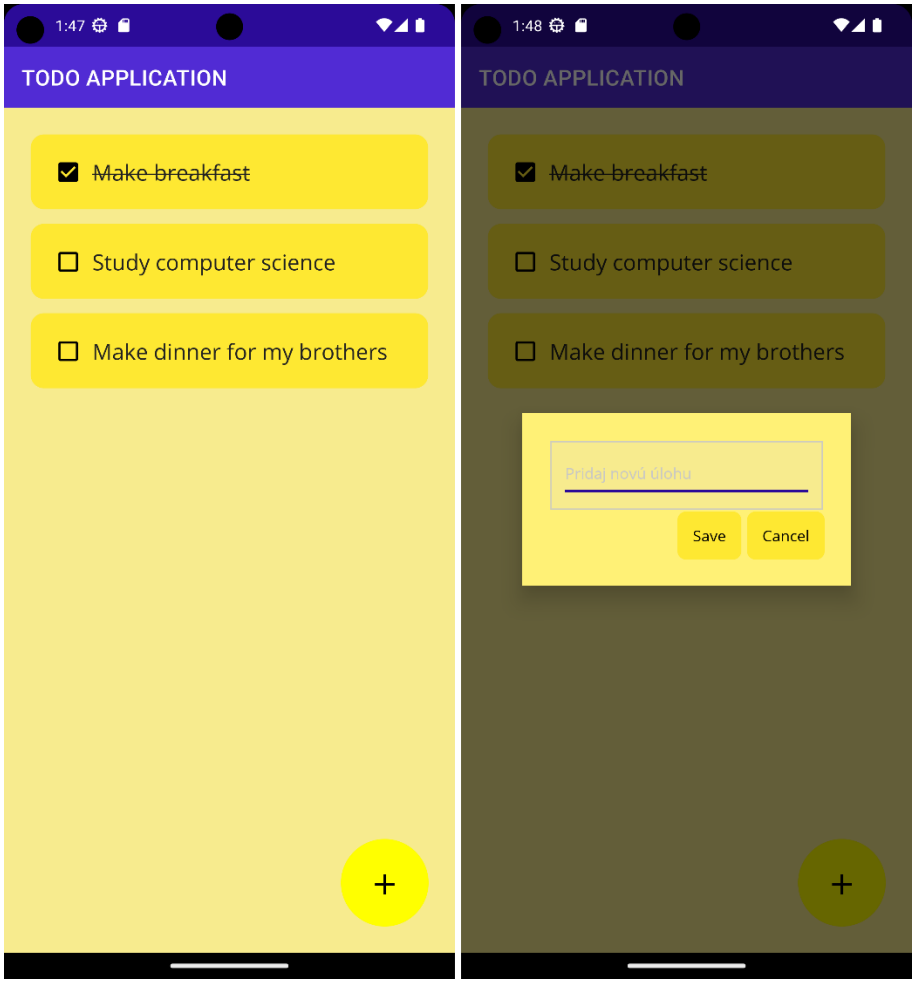


Obrázok 3 Flutter - Hlavná stránka a dialógové okno aplikácie

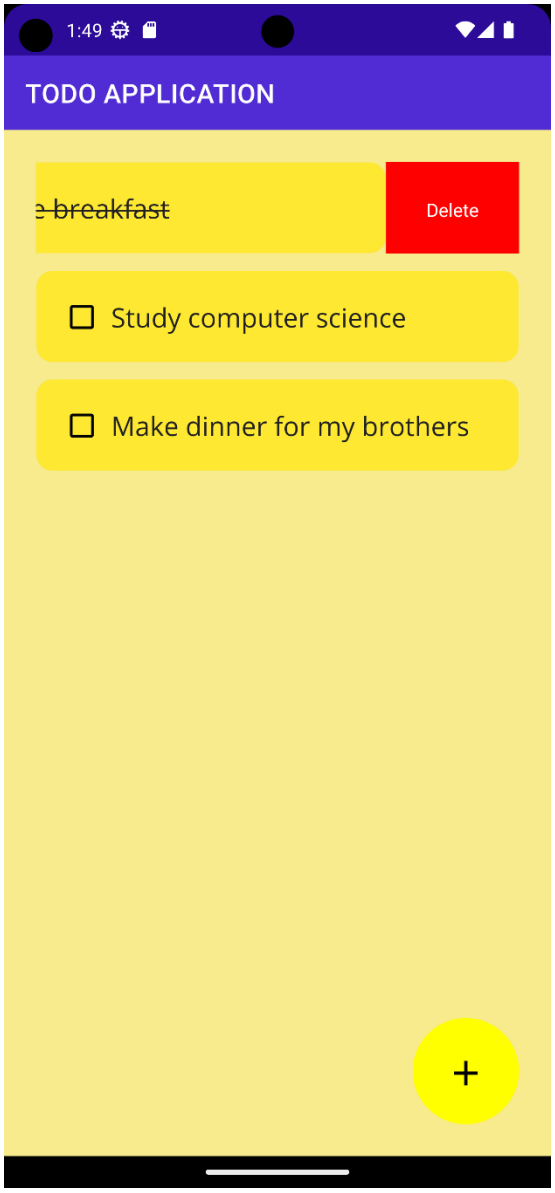


Obrázok 4 Flutter – Vymazanie úlohy

Užívateľské rozhranie pre .NET MAUI je vyobrazené na obrázkoch 5 a 6.



Obrázok 5 .NET MAUI - Hlavná stránka a dialógové okno aplikácie



Obrázok 6 .NET MAUI – Vymazanie úlohy

4.1.1 Flutter TODO aplikácia

Aby sme mohli pracovať s lokálnou databázou tak musíme pridať závislosti do našej aplikácie.

```
dependencies:
  flutter:
    sdk: flutter
  hive: ^2.2.3
  hive_flutter: ^1.1.0
```

Vytvoríme triedu TodoLocalStorage, ktorá predstavuje lokálnu databázu prostredníctvom knižnice Hive. Je nutné importovať knižnicu hive_flutter, ktorá umožňuje používať lokálnu databázu v aplikácií.

V triede TodoLocalStorage vytvoríme promennú tasks, ktorá zo začiatku bude inicializovaná ako prázdny list. Následne sa vytvorí inštancia Hive boxu, do ktorého sa budú ukladať dáta. Trieda TodoLocalStorage obsahuje metódu createInitialData, ktorá slúži na vytvorenie počiatočných dát. Ďalej obsahuje metódu dataLoadFromHive slúžiacu na načítanie a následne nahradenie hodnôt v liste tasks a databázu dataUpdateStorage, ktorá aktualizuje lokálnu databázu.

```
import 'package:hive_flutter/hive_flutter.dart';

class TodoLocalStorage {
  List tasks= [];
  final hive = Hive.box('hiveBox');

  void createInitialData(){
    tasks = [
      ["Make dinner", false],
    ];
  }
  void dataLoadFromHive(){
    tasks = hive.get("tasks");
  }
  void dataUpdateStorage(){
    hive.put("tasks", tasks);
  }
}
```

Vytvoríme triedu DialogWindow na zobrazenie dialógového okna (viď Obrázok 3). Trieda bude obsahovať tri parametre a to: taskNameController slúžiaci pre widget TextField, SaveClicked a CancelClicked, ktoré predstavujú metódy pre stlačenie príslušných tlačidiel v dialógovom okne.

```
class DialogWindow extends StatelessWidget {
  final taskNameController ;
  VoidCallback SaveClicked;
  VoidCallback CancelClicked;

  DialogWindow({
    super.key,
    required this.taskNameController ,
    required this.SaveClicked,
    required this.CancelClicked,
  });
```

Na zobrazenie dialógového okna sa používa widget AlertDialog, ktorý obsahuje widget Container s nastavenou požadovanou výškou. V rámci tohto kontajnera sa nachádza widget TextField, vytvárajúci textové pole. Hodnota z tohto poľa je uložená v parametri taskNameController. Widget ButtonBar sa používa na vytvorenie dvoch tlačidiel v horizontálnom usporiadaní. Tieto tlačidlá spúšťajú metódy SaveClicked a CancelClicked po ich kliknutí.

```
@override
Widget build(BuildContext context) {
  return AlertDialog(
    backgroundColor: Colors.yellow,
  content: Container(
    height: 130,
    child: Stack(
      children: [
        TextField(
            decoration: const InputDecoration(
              hintText: "Pridaj novú úlohu",
              border: OutlineInputBorder(),
            ),
          controller: taskNameController,
          ),
        Positioned(
          bottom: 2.0,
          right: 0.0,
          child: ButtonBar(
            alignment: MainAxisAlignment.center,
            children: [
              MaterialButton(
                onPressed: SaveClicked,
                color: Colors.yellow[300],
                child: const Text("Save"),
              ),
              MaterialButton(
                onPressed: CancelClicked,
                color: Colors.yellow[300],
                child: const Text("Cancel"),
              ),
              ...
```

Vytvoríme triedu Todo pre zobrazenie úlohy. V rámci triedy budú existovať štyri parametre. Parameter title bude obsahovať názov úlohy. Parameter taskStatus bude indikovať, či je úloha dokončená alebo nie. Na zmenu stavu checkboxu sa použije metóda changeTaskStatus. Metóda delete, ktorá sa zavolá pri vymazávaní úlohy.

```
class Todo extends StatelessWidget {
  final String title;
  final bool taskStatus;
  Function(bool?)? changeTaskStatus;
  Function(BuildContext)? delete;

  Todo({
    required this.title,
    required this.taskStatus,
    required this.changeTaskStatus,
    required this.delete,
  });
```

Na zobrazenie jednotlivých úloh sa využíva widget Container s nastaveným paddingom, ktorý poskytuje okraje okolo widgetu. Vnútri tohto kontajnera sa nachádza widget Slidable, ktorý umožňuje vymazanie úlohy posunutím prsta doprava (viď Obrázok 4).

Táto akcia je vykonávaná widgetom SlidableAction.

```
Container(
  padding: const EdgeInsets.symmetric(horizontal: 12.5, vertical: 12.5),
  child: Slidable(
    endActionPane: ActionPane(
      motion: const BehindMotion(),
      children: [
        SlidableAction(
          backgroundColor: Colors.red,
          icon: Icons.delete,
          onPressed: delete,
        ),
      ],
    ),
```

V rámci widgetu Card je umiestnený widget CheckboxListTile, ktorý reaguje na zmenu stavu úlohy a volá príslušnú metódu definovanú v parametroch ako changeTaskStatus. Tento widget obsahuje aj widget Text, ktorý využíva parameter title.

```
Card(
  color: Colors.yellow,
  shape: const RoundedRectangleBorder(
    borderRadius: BorderRadius.all(Radius.circular(12)),
  ),
  child: CheckboxListTile(
    activeColor: Colors.black,
    title: Text(
      title,
      style: TextStyle(
        decoration: taskStatus ? TextDecoration.lineThrough : null,
      ),
    ),
    value: taskStatus,
    onChanged: changeTaskStatus,
  ),
),
```

Vytvoríme hlavnú triedu HomePage ktorá bude zobrať všetky úlohy a tlačidlo na pridanie úlohy (viď Obrázok 3).

Trieda HomePage je hlavná stránka aplikácie a je dedená z triedy StatefulWidget, pri inicializácii musíme inicializovať knižnicu Hive a triedu TodoLocalStorage.

```
final _hiveBox = Hive.box('hiveBox');
TodoLocalStorage db = TodoLocalStorage();
```

Metóda initState kontroluje či databáza nie je prázdna, ak nie je prázdna, zavolá sa metóda dataLoadFromHive, ak je prázdna lokálna databáza sa naplní počiatočnými dátami pomocou metódy createInitialData.

```
void initState() {
  hive.get("tasks") != null
      ? storage.dataLoadFromHive()
      : storage.createInitialData();
  super.initState();
}
```

Funkcia changeStatusOfTodo slúži na zmenu hodnoty stavu konkrétnej úlohy a následne je celá databáza aktualizovaná pomocou zavolania metódy dataUpdateStorage.

```
void changeStatusOfTodo(bool? value, int index){
  setState(() {
    final negatedValue = !storage.tasks[index][1];
    storage.tasks[index][1]=negatedValue;
  });
  storage.dataUpdateStorage();
}
```

Táto metóda save slúži na pridanie novej úlohy do lokálnej databáze, následne vymaže obsah vstupného poľa a nakoniec je celá lokálna databáza aktualizovaná pomocou metódy dataUpdateStorage.

```
void save(){
  setState(() {
    final newTodo = [textEditorController.text,false];
    storage.tasks.add(newTodo);
    textEditorController.clear();
  });
  Navigator.of(context).pop();
  storage.dataUpdateStorage();
}
```

Metóda create otvorí dialógové okno v ktorom môže užívateľ vytvoriť novú úlohu. Dialógové okno je implementované pomocou triedy DialogBox. Po kliknutí na tlačidlo sa zavolá metóda save ktorá pridá novú úlohu do lokálnej databáze.

```
void create(){
  showDialog(
    context: context,
    builder: (context){
      return DialogWindow(
        CancelClicked: ()=> Navigator.of(context).pop(),
        SaveClicked: save,
        taskNameController : textEditorController,
      );
    },
  );
}
```

Metóda delete najskôr vymaže konkrétnu úlohu a následne aktualizuje lokálnu databázu pomocou metódy dataUpdateStorage.

```
void delete(int todoIndex){
  setState(() {
    storage.tasks.removeAt(todoIndex);
  });
  storage.dataUpdateStorage();
}
```

Na zobrazenie aplikácie sa používa widget Scaffold, kde je definovaný nadpis pomocou AppBar a nastavené pozadie na žltú farbu. Pre vypísanie všetkých položiek z lokálnej databázy sa používa widget ListView.builder, ktorý dynamicky vykresľuje jednotlivé úlohy. Každá položka je zobrazená pomocou triedy Todo, ktorá pri zmene stavu úlohy volá metódu changeStatusOfTodo a pri vymazaní konkrétnej úlohy sa zavolá metóda delete. Okrem toho je tiež zobrazené tlačidlo, ktoré po kliknutí volá metódu create.

```
Scaffold(
  backgroundColor: Colors.yellow[200],
  appBar: AppBar(
    title: const Text('TODO APPLICATION'),
    centerTitle: true,
  ),
  body: ListView.builder(
    itemCount: storage.tasks.length,
    itemBuilder: (context, index) {
      return Padding(
        padding: const EdgeInsets.symmetric(horizontal: 5.0),
        child: Todo(
          changeTaskStatus: (value) => changeStatusOfTodo(value, index),
          taskStatus: storage.tasks[index][1],
          delete: (context) => delete(index),
          title: storage.tasks[index][0],
        ),
      );
    },
  ),
  floatingActionButton: FloatingActionButton.extended(
    backgroundColor: Colors.yellow,
    onPressed: create,
    label: const Text(
      '+',
        style: TextStyle(fontSize: 30)),
  ),
);
```

4.1.2 .MAUI TODO aplikácia

Na ukladanie úloh budeme využívať knižnicu SQLite-net a na správne fungovanie Todo aplikácie je potrebné nainštalovať všetky potrebné NuGet Packages:

- sqlite-net-pcl
- SQLitePCLRaw.bundle_green

Vytvorí sa trieda Todo, ktorá reprezentuje jednotlivé úlohy. Táto trieda obsahuje vlastnosť Id, ktorá slúži ako primárny kľúč a automaticky sa zväčšuje, aby každá úloha mala jedinečné Id. Ďalej má vlastnosť TaskName, ktorá reprezentuje názov úlohy, a vlastnosť TaskCompleted, ktorá určuje, či je úloha dokončená.

Trieda Todo implementuje rozhranie INotifyPropertyChanged, čo umožňuje informovať aplikáciu v prípade, že sa zmenia vlastnosti TaskName a TaskCompleted.

```
public class Todo : INotifyPropertyChanged
{
    [PrimaryKey, AutoIncrement]
    public int Id { get; set; }
    public string taskName;
    public bool taskCompleted;

    public string TaskName
    {
        get => taskName;
        set
        {
            taskName = value;
            OnPropertyChanged();
        }
    }

    public bool TaskCompleted
    {
        get => taskCompleted;
        set
        {
            taskCompleted = value;
            OnPropertyChanged();
        }
    }

    public event PropertyChangedEventHandler PropertyChanged;

   protected virtual void OnPropertyChanged([CallerMemberName] string proper-
tyName = null)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(property-
Name));
    }
}
```

Vytvoríme triedu TodoItemDatabase slúžiacu ako prostredník medzi aplikáciu a lokálnou databázou.

V triede TodoItemDatabase definujeme premennú Database typu SQLiteAsyncConnection ktorá bude použitá pre pripojenie k lokálnej databázy.

```
SQLiteAsyncConnection Database;
```

Definujeme asynchrónnu metódu Init slúžiacu k inicializácii prepojenia k lokálnej databázy SQLite a vytvorený tabuľky Todo.

```
public async Task Init()
{
    if (Database is not null)
        return;

    Database = new SQLiteAsyncConnection(Constants.DatabasePath, Con-
stants.Flags);
    Database.CreateTableAsync<Todo>().Wait();
    var count = await Database.Table<Todo>().CountAsync();
    if (count == 0)
    {
        var todo = new Todo
        {
            TaskName = "Make dinner",
            TaskCompleted = false
        };
        await Database.InsertAsync(todo);
    }
}
```

Asynchrónna metóda GetItemsAsync načíta všetky úlohy typu Todo z lokálnej databáze a vráti zoznam všetkých úloh, uložených v lokálnej databáze.

```
public async Task<List<Todo>> GetItemsAsync()
{
    await Init();
    return await Database.Table<Todo>().ToListAsync();
}
```

Na odstraňovanie jednotlivých úloh slúži asynchrónna metóda DeleteItemAsync ktorá prijíma ako vstupný parameter úlohu typu Todo.

```
public async Task<int> DeleteItemAsync(Todo item)
{
    await Init();
    return await Database.DeleteAsync(item);
}
```

Asynchrónna metóda SaveItemAsync prijíma úlohu typu Todo ako vstupný parameter a pokiaľ úloha už existuje v lokálnej databázy tak metóda aktualizuje existujúcu úlohu. Ak neexistuje tak metóda ju vloží do lokálnej databáze ako novú.

```
public async Task<int> SaveItemAsync(Todo todo)
{
    await Init();

    if (todo.Id != 0)
    {
        return await Database.UpdateAsync(todo);
    }
    else
    {
        return await Database.InsertAsync(todo);
    }
}
```

Vytvoríme ViewModel, ktorý využíva triedy Todo a TodoItemDatabase.

Definujeme triedu TodoListViewModel, ktorá je odvodená od triedy ObservableObject. Trieda ObservableObject obsahuje metódu SetProperty, ktorá aktualizuje hodnotu vlastnosti a zavolá udalosť PropertyChanged ak sa vlastnosť zmenila.

```
public class TodoListViewModel : ObservableObject
```

Property ToDolist typu ObservableCollection<Todo> predstavuje zoznam úloh, ktoré majú byť zobrazené v zozname. Ak sa zmení obsah tohto zoznamu, vyvolá sa udalosť PropertyChanged, ktorá zabezpečí aktualizáciu.

```
public ObservableCollection<Todo> ToDolist
{
    get => toDolist;
    set
    {
        SetProperty(ref toDolist, value);
        OnPropertyChanged(nameof(ToDolist));
    }
}
```

Metóda LoadDataAsync načíta všetky úlohy z databázy a vytvorí z nich novú kolekciu ObservableCollection<Todo>.

```
public async Task LoadDataAsync()
{
    var todos = await database.GetItemsAsync();
    ToDolist = new ObservableCollection<Todo>(todos);
}
```

Konštruktor triedy TodoListViewModel inicializuje novú inštanciu triedy TodoItemDatabase a načíta existujúce položky z lokálnej databázy pomocou metódy LoadDataAsync.

```
private TodoListViewModel()
{
    database = new TodoItemDatabase();
    _ = LoadDataAsync();
}
```

Na odstránenie úlohy slúži metóda DeleteTodo. Táto metóda odstráni položku z lokálnej databáze a opätovne načíta zoznam úloh pomocou metódy LoadDataAsync.

```
public async Task DeleteTodo(Todo todo)
{
    await database.DeleteItemAsync(todo);
    await LoadDataAsync();

}
```

Na zobrazenie dialógového okna slúži trieda PopupPage (viď Obrázok 6).

Trieda PopupPage je potomkom triedy Popup a má dve metódy OnSaveButtonClicked a OnCancelButtonClicked.

```
public partial class PopupPage : Popup
```

Po kliknutí na tlačidlo Save sa zavolá metóda OnSaveButtonClicked ktorá vytvorí novú úlohu a pridá ju do kolekcie viewModel.ToDolist, uloží úlohu do lokálnej databáze pomocou metódy SaveItemAsync, znova načíta dáta a nakoniec zavrie dialógové okno.

```
private async void OnSaveButtonClicked(object sender, EventArgs e)
{
    Todo newTodo = new Todo {
        TaskName = editor.Text,
        TaskCompleted = false };

    viewModel.ToDolist.Add(newTodo);
    await database.SaveItemAsync(newTodo);
    await TodoListViewModel.Instance.LoadDataAsync();
    Close();
}
```

Po kliknutí na tlačidlo Cancel sa zavolá metóda OnCancelButtonClicked ktorá zavrie dialógové okno.

```
private void OnCancelButtonClicked(object sender, EventArgs e)
{
    Close();
}
```

Pre zobrazenie užívateľského rozhrania dialógového okna je definovaný Grid, ktorý obsahuje Editor ktorý slúži na zadávanie názvu novej úlohy a dve tlačidlá ktoré môžu pridať novú úlohu do lokálnej databáze pomocou metódy OnSaveButtonClicked alebo zavrieť dialógové okno metódou OnCancelButtonClicked.

```
<Editor
    x:Name="editor"
    Placeholder="Pridaj novú úlohu"
    WidthRequest="230"
    />
</Border>
<HorizontalStackLayout Grid.Row="1" Spacing="5" HorizontalOptions="End">
<Button Text="Save"
        BackgroundColor="#FEE832"
        TextColor="Black"
        Clicked="OnSaveButtonClicked"/>
<Button Text="Cancel"
        BackgroundColor="#FEE832"
        TextColor="Black"
        Clicked="OnCancelButtonClicked"/>
</HorizontalStackLayout>
```

Za zobrazenie hlavnej stránky aplikácie je zodpovedná trieda MainPage (viď Obrázok 5). V metóde MainPage sa inicializuje BindingContent na inštanciu triedy TodoListViewModel.

```
public MainPage()
{
   InitializeComponent();
   BindingContext = TodoListViewModel.Instance;
   database = new TodoItemDatabase();

}
```

V metóde CheckBox_CheckedChanged sa nastavuje stav danej úlohy a ukladá sa to do lokálnej databáze pomocou metódy SaveItemAsync.

```
private async void CheckBox_CheckedChanged(object sender,
                                    CheckedChangedEventArgs e)
{
    var checkBox = (CheckBox)sender;
    var todo = (Todo)checkBox.BindingContext;
    todo.TaskCompleted = checkBox.IsChecked;

    await database.SaveItemAsync(todo);
}
```

Metóda OnDeleteSwipeItemInvoked slúži na vymazanie úlohy z lokálnej databáze pomocou metódy DeleteTodo (viď Obrázok 6).

```
async void OnDeleteSwipeItemInvoked(object sender, EventArgs e)
{
    var task = (sender as SwipeItem).BindingContext as Todo;

    await TodoListViewModel.Instance.DeleteTodo(task);
}
```

Na otvorenie dialógového okna, v ktorom môže užívateľ vytvoriť novú úlohu slúži metóda ButtonClicked ktorá zavolá metóda ShowPopup(), ktorá zobrazí novú stránku na obrazovke.

```
private async void ButtonClicked(object sender, EventArgs e)
{
    await TodoListViewModel.Instance.LoadDataAsync();
    this.ShowPopup(new PopupPage(TodoListViewModel.Instance));
}
```

Na zobrazenie užívateľského rozhrania hlavnej stránky slúži ContentPage, ktorý obsahuje zoznam úloh a tlačidlo na pridanie novej úlohy.

Zoznam úloh je zobrazený pomocou CollectionView, kde každá úloha je reprezentovaná widgetom SwipeView. SwipeView obsahuje tlačidlo pre vymazanie položky, ktoré je aktivované posunom prsta vpravo. Taktiež obsahuje Checkbox, ktorý zobrazuje stav dokončenia danej úlohy, a Label pre zobrazenie názvu úlohy.

```
<CollectionView.ItemTemplate>
    <DataTemplate x:DataType="models:Todo">
        <SwipeView BindingContext="{Binding }">
            <SwipeView.RightItems>
                <SwipeItem Text="Delete"
                           BackgroundColor="Red"
                           Invoked="OnDeleteSwipeItemInvoked"/>
            </SwipeView.RightItems>
            <Frame CornerRadius="12"
                   BackgroundColor="#FEE832"
                   Padding="12.5"
                   HasShadow="False"
                   BorderColor="Transparent">
                <Grid>
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="Auto" />
                        <ColumnDefinition Width="*" />
                    </Grid.ColumnDefinitions>
                    <CheckBox
                        x:Name="checkBox"
                        IsChecked="{Binding TaskCompleted}"
                        VerticalOptions="Center"
                        CheckedChanged="CheckBox_CheckedChanged"
                         Color="Black"/>
                    <Label
                        FontSize="20"
                        Grid.Column="1"
                        Text="{Binding TaskName}"
                        VerticalOptions="Center">
                        <Label.Triggers>
                        <DataTrigger
                            TargetType="Label"
                            Binding="{Binding Source={x:Reference checkBox},
                            Path=IsChecked}"
                            Value="true">
                            <Setter
                                Property="TextDecorations"
                                Value="Strikethrough" />
                        </DataTrigger>
                        </Label.Triggers>
                    </Label>
                </Grid>
            </Frame>
        </SwipeView>
    </DataTemplate>
</CollectionView.ItemTemplate>
```

Tlačidlo je umiestnené v pravom dolnom rohu obrazovky a slúži na otvorenie dialógového okna.

```
<Button
    AbsoluteLayout.LayoutBounds="0.95,0.95,80,80"
    AbsoluteLayout.LayoutFlags="PositionProportional"
    CornerRadius="40"
    HeightRequest="80"
    WidthRequest="80"
    Text="+"
    HorizontalOptions="End"
    Grid.Row="1"
    FontSize="38"
    BackgroundColor="Yellow"
    TextColor="Black"
    Clicked="ButtonClicked"
/>
```

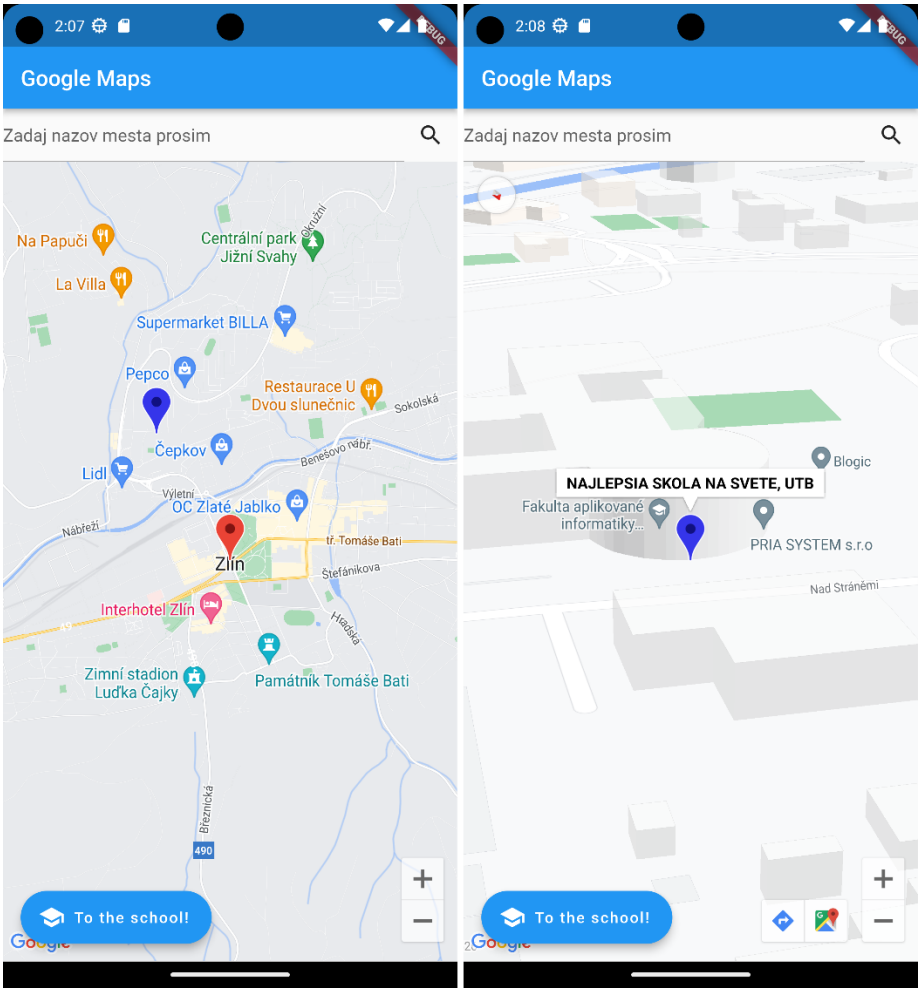
4.1.3 Porovnanie riešení

Pri vývoji TODO aplikácie s použitím lokálnej databázy Hive pre framework Flutter a lokálnej databázy SQLite pre framework .NET MAUI bolo jedným z hlavných rozdiel to, že práca s lokálnou databázou Hive bola jednoduchšia a intuitívnejšia, na rozdiel od lokálnej databáze SQLite, kde práca s lokálnou databázou spôsobovala rôzne problémy. Navyše pri SQLite lokálnej databázy bola nutnosť inštalácie dodatočných závislostí. Počas vývoja TODO aplikácie v frameworku .NET Maui sa objavovali rôzne chybové hlášky, ktoré neboli veľmi indikujúce o aký typ chyby sa jednalo, čo spôsobovalo viac času stráveného nad hľadaným chyby a následného nie vždy jednoduchého riešenia.

4.2 Google Maps aplikácia

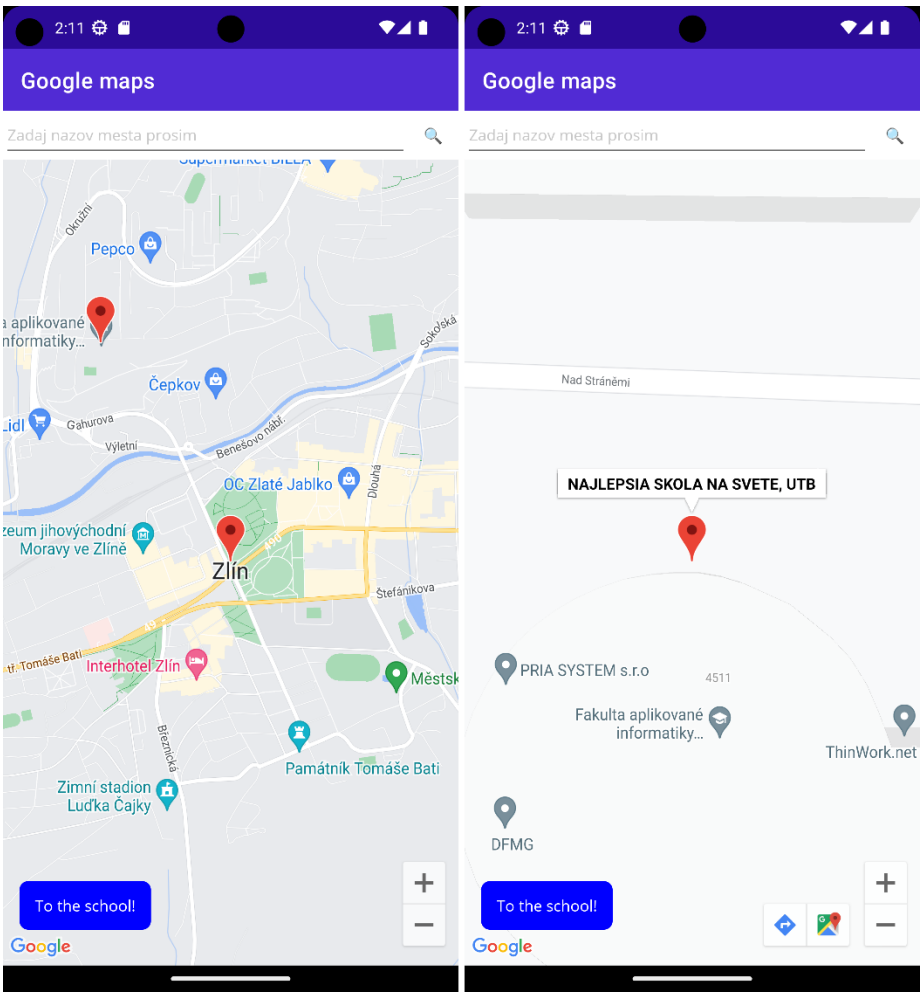
Cieľom tejto aplikácie je zobrazenie Google Maps s možnosťou vyhľadávania polohy, zobrazenie značky u mesta Zlín a u Fakulty aplikované informatiky Univerzity Tomáše Bati a vytvorenie tlačidla ktoré zobrazí polohu Fakulty aplikované informatiky Univerzity Tomáše Bati. Na zobrazenie Google Maps využijeme služby od spoločnosti Google, ktoré sú dostupné v rámci Google Maps API. Pre použitie týchto služieb bude potrebné vytvoriť API kľúč na webovej stránke [41].

Užívateľské rozhranie pre Flutter je vyobrazené na obrázku 7.



Obrázok 7 Flutter – Hlavná stránka aplikácie a zobrazenie univerzitné budovy

Užívateľské rozhranie pre .NET MAUI je vyobrazené na obrázku 8.



Obrázok 8 .NET MAUI – Hlavná stránka a zobrazenie univerzitné budovy

4.2.1 Flutter Google Maps aplikácia

Aby sme mohli pracovať s Google Maps widgetom a s HTTP musíme pridať závislosti do našej aplikácie.

```
dependencies:
  flutter:
    sdk: flutter
  google_maps_flutter: ^2.2.5
  http: ^0.13.5
```

Aby bolo možné používať API kľúč v aplikácii, je nutné ho pridať do súboru AndroidManifest.xml.

```
<meta-data android:name="com.google.android.geo.API_KEY"
    android:value="API KLÚČ"/>
```

Vytvoríme triedu LocationService obsahujúcu metódu getCity.

V rámci prvej časti metódy sa získa ID konkrétneho miesta na základe textového vstupu. Na získanie tohto ID je potrebné vykonať HTTP get požiadavky a následne spracovať odpoveď do formátu JSON. Po získaní ID konkrétneho miesta sa vykoná ďalšia HTTP GET požiadavka a odpoveď sa opäť spracuje do formátu JSON. Následne je táto odpoveď prevodená do formátu Map<String, dynamic>.

```
Future<Map<String,dynamic>> getCity(String input) async {
  const String apiKey = 'API KLÚČ';

  final String urlAddressForId =
      'https://maps.googleapis.com/maps/api/place/findplacefrom-
text/json?input=$input&inputtype=textquery&key=$apiKey';
  var responseIdFromApi = await http.get(Uri.parse(urlAddressForId));
  var convertedResponseIdToJson = convert.jsonDecode(responseIdFro-
mApi.body);

  var cityId =convertedResponseIdToJson['candidates'][0]['place_id'] as
String;

  final String urlAddress =
      'https://maps.googleapis.com/maps/api/place/de-
tails/json?place_id=$cityId&key=$apiKey';
  var responseFromApi = await http.get(Uri.parse(urlAddress));
  var convertedResponseToJson = convert.jsonDecode(responseFromApi.body);
  var data = convertedResponseToJson['result'] as Map<String,dynamic>;

  return data;
}
```

Za zobrazenie hlavnej stránky aplikácie je zodpovedá trieda MyHomePage (viď Obrázok 7). V rámci triedy MyHomePage je implementovaný GoogleMaps widget, ktorý dokáže zobrazovať mapu a umožňuje používateľom interagovať s mapou, vyhľadávanie konkrétnych miest je umožnené pomocou TextFormField widgetu. Okrem toho, trieda MyHomePage poskytuje aj tlačidlo na presunutie na predom nastavenú pozíciu na mape.

```
Expanded(child: TextFormField(
    controller: _searchController,
    decoration: const InputDecoration(hintText: 'Zadaj nazov mesta pro-
sim'),

  ),),
  IconButton(onPressed: () async {
    var place = await LocationService().getCity(_searchController.text);
    changeCity(place);

  }, icon: const Icon(Icons.search),),
],),
Expanded(
  child: GoogleMap(
```

```
    mapType: MapType.normal,
    markers: {
      Marker(
        markerId: MarkerId('UTB'),
        infoWindow: InfoWindow(title: 'NAJLEPSIA SKOLA NA SVETE, UTB'),
        icon: BitmapDescriptor.defaultMarkerWithHue(
            BitmapDescriptor.hueBlue),
        position: LatLng(49.2309423640983, 17.657050811858245),
      ),
      const Marker(
        markerId: MarkerId('Zlin'),
        infoWindow: InfoWindow(title: 'DOMOV'),
        icon: BitmapDescriptor.defaultMarker,
        position: LatLng(49.22450149827196, 17.662751211644206),
      ),
    },
    initialCameraPosition: const CameraPosition(
      target: LatLng(49.22450149827196, 17.662751211644206),
      zoom: 14,
    ),
    onMapCreated: (GoogleMapController controller) {
      _controller.complete(controller);
    },
  ),
),
```

4.2.2 .NET MAUI Google Maps aplikácia

Na prácu s Google Maps budeme potrebovať nainštalovať NuGet Packages Microsoft.Maui.Controls.Maps.

Aby bolo možné používať API kľúč v aplikácii, je nutné ho pridať do súboru AndroidManifest.xml.

```
<meta-data android:name="com.google.android.geo.API_KEY"
           android:value="API KLÚČ" />
```

Vytvoríme triedu LocationService obsahujúcu dve metódy getCity a getCityId.

Metóda getCityId získa ID konkrétneho miesta na základe textového vstupu. Na získanie ID sa použije klientska trieda HttpClient, ktorá odošle požiadavku a následne získa odpoveď. Táto odpoveď sa prevedie do formátu JSON, z ktorého sa získa ID miesta.

```
public async Task<string> getCityId(string input)
{
    string urlAddress =
        $"https://maps.googleapis.com/maps/api/place/findplacefro-
text/json?iput={input}&inputtype=textquery&key={apiKey}";
    using (HttpClient httpClient = new HttpClient())
    {
        HttpResponseMessage responseFromApi = await httpClient.GetAsync(urlAd-
dress);
```

```
        string responseFromApiString = await responseFromApi.Content.ReadAs-
StringAsync();
        using (JsonDocument convertedResponseToJson = JsonDocument.Parse(res-
ponseFromApiString))
        {
            JsonElement root = convertedResponseToJson.RootElement;
            JsonElement candidates = root.GetProperty("candidates");
            JsonElement firstCandidate = candidates[0];
            string placeId = firstCandidate.GetProperty("place_id").Get-
String();

            return placeId;
        }
    }
}
```

Metóda getCity využíva ID miesta získaného metódou GetCityId. Používa sa klientska trieda HttpClient na odoslanie požiadavky a na získanie odpovede. Potom sa odpoveď prevedie na JSON formát a z neho sa získa slovník hodnôt.

```
public async Task<Dictionary<string, object>> getCity(string input)
{
    string cityId = await getCityId(input);
    string urlAddress =
            $"https://maps.googleapis.com/maps/api/place/de-
tails/json?place_id={cityId}&key={apiKey}";
    using (HttpClient httpClient = new HttpClient())
    {
        HttpResponseMessage responseFromApi = await httpClient.GetAsync(urlAd-
dress);
        string responseFromApiString = await responseFromApi.Content.ReadAs-
StringAsync();

        using (JsonDocument convertedResponseToJson = JsonDocument.Parse(res-
ponseFromApiString))
        {
            JsonElement root = convertedResponseToJson.RootElement;
            JsonElement result = root.GetProperty("result");
            Dictionary<string, object> data = JsonSerializer.Deserialize<Dic-
tionary<string, object>>(result.GetRawText());

            return data;
        }
    }
}
```

V konštruktore MainPage sa pridajú značky na mapu. V prvom rade sa inicializuje komponenta a potom sa pomocou metódy myMap.Pins.Add sa pridajú dve značky na mapu.

```
public MainPage()
{
itializeComponent();
    myMap.Pins.Add(new Microsoft.Maui.Controls.Maps.Pin
    {
        Label = "DOMOV",
        Location = new Location(49.224655968514085, 17.66275823784387),
    });
    myMap.Pins.Add(new Microsoft.Maui.Controls.Maps.Pin
    {
        Label = "NAJLEPSIA SKOLA NA SVETE, UTB",
        Location = new Location(49.2309423640983, 17.657050811858245),
    });
}
```

Metóda MoveMapToSelectedPlace slúži na zobrazenie zadaného miesta na mape. Táto metóda získa zadané súradnice miesta zo vstupného parametru placeDetails, následne sa vytvorí nový MapSpan a nastaví sa pohľad na mapu.

```
async void MoveMapToSelectedPlace(Dictionary<string, object> placeDetails)
{
    var lat = ((JsonElement)placeDetails["geometry"])
        .GetProperty("location").GetProperty("lat").GetDouble();
    var lng = ((JsonElement)placeDetails["geometry"]).
        GetProperty("location").GetProperty("lng").GetDouble();
    var mapSpan = new MapSpan(new Location(lat, lng), 0.01, 0.01);
    myMap.MoveToRegion(mapSpan);
}
```

Metóda OnGetPlaceClicked vykonáva vyhľadávanie miesta na mape na základe zadaného textu pomocou metódu getCity. Následne zavolá metódu MoveMapToSelectedPlace, ktorá presunie mapu na zadané miesto.

```
private async void OnGetPlaceClicked(object sender, EventArgs e)
{
    var input = txtInput.Text;
    var locationService = new LocationService();
    var placeDetails = await locationService.getCity(input);
    MoveMapToSelectedPlace(placeDetails);
}
```

Metóda GoToUTB slúži na posunutie mapy na predom nastavenú pozíciu na mape.

```
private async void GoToUTB(object sender, EventArgs e)
{
    MapSpan mapSpan = MapSpan.FromCenterAndRadius(
        new Location(49.2309423640983, 17.657050811858245, 68.0),
        Distance.FromKilometers(0.02));
    myMap.MoveToRegion(mapSpan);
}
```

Na zobrazenie užívateľského rozhrania hlavnej stránky slúži ContentPage, ktorý obsahuje mapu s možnosťou vyhľadávania miest a tlačidlo na posunutie mapy na preddefinovanú polohu (viď Obrázok 8).

Vyhľadávanie miest je zobrazené pomocou StackLayout, ktorý pozostáva z elementu Entry ktorý slúži na zadanie textu pre vyhľadávanie a Button ktorý po kliknutí zavolá metódu OnGetPlaceClicked.

```
<StackLayout Grid.Row="0"
             Orientation="Horizontal"
             HorizontalOptions="FillAndExpand">
    <Entry x:Name="txtInput"
           Placeholder="Zadaj nazov mesta prosim"
           HorizontalOptions="FillAndExpand">
    </Entry>
    <Button Text="🔍"
            Clicked="OnGetPlaceClicked"
            BackgroundColor="White"/>
</StackLayout>
```

Mapa a tlačidlo sú zobrazené pomocou AbsoluteLayout. V elemente MapSpan je definované počiatočné zobrazenie mapy s konkrétnymi súradnicami. V elemente je pridaný event handler ktorý zavolá metódu GoToUTB.

```
<AbsoluteLayout Grid.Row="1">
     <maps:Map x:Name="myMap"
               AbsoluteLayout.LayoutBounds="0,0,1,1"
               AbsoluteLayout.LayoutFlags="All">
         <x:Arguments>
             <MapSpan>
                 <x:Arguments>
                     <sensors:Location>
                         <x:Arguments>
                             <x:Double>49.224655968514085</x:Double>
                             <x:Double>17.66275823784387</x:Double>
                         </x:Arguments>
                     </sensors:Location>
                     <x:Double>0.02</x:Double>
                     <x:Double>0.02</x:Double>
                 </x:Arguments>
             </MapSpan>
         </x:Arguments>
     </maps:Map>

     <Button
         AbsoluteLayout.LayoutBounds="0.05,0.95,120,80"
         AbsoluteLayout.LayoutFlags="PositionProportional"
         Text="To the school!"
         VerticalOptions="End"
         HorizontalOptions="Start"
         BackgroundColor="Blue"
         TextColor="White"
         Clicked="GoToUTB"
        />
 </AbsoluteLayout>
```

4.2.3 Porovnanie riešení

Pri vývoji aplikácie Google Maps sme využili služby od spoločnosti Google, konkrétne Google Maps API. Vývoj tento aplikácie pre oba frameworky prebiehal celkom pohodlne a bez väčších ťažkostí, avšak u frameworku .NET MAUI sa vyskytol problém s tým, že nie je možné otočiť uhol zobrazenia kamery.

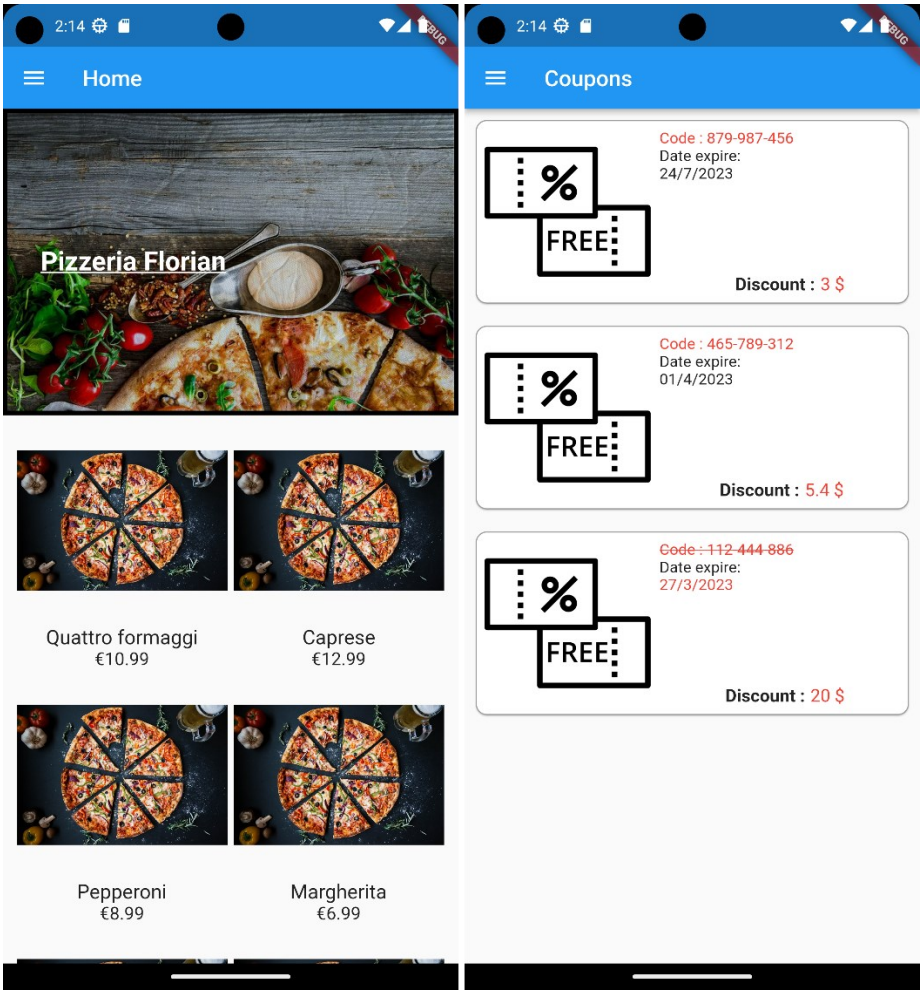
4.3 Navigation menu aplikácia

Cieľom tejto aplikácie je vytvorenie postranného menu, kde budú zobrazené jednotlivé záložky Home, Coupons, Orders, My payment methods. Každá záložka bude mať možnosť prekliknutia na príslušnú obrazovku. V hlavičke postranného menu bude obrázok zákazníka, meno zákazníka a bude mať obrázok ako pozadie.

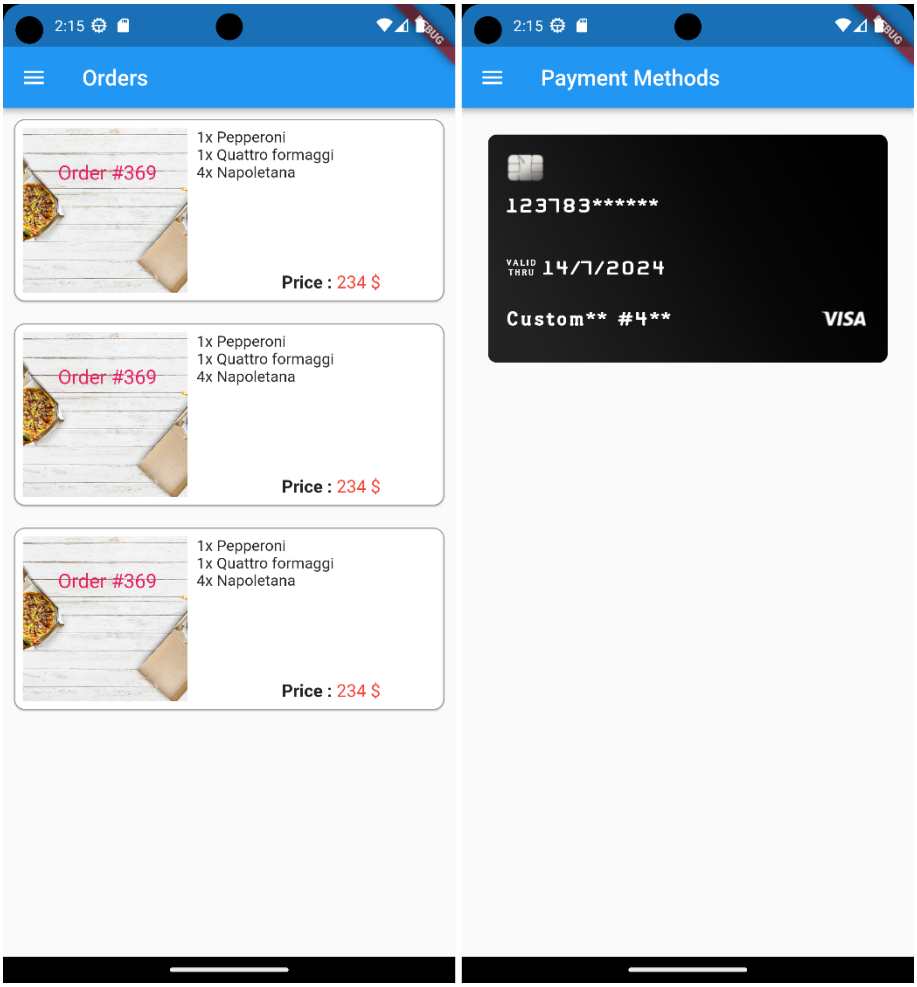
Všetky použité obrázky u aplikácie Navigation menu boli použité z stránok:

- Flaticon [42]
- Pixabay [43]
- Freepik [44]
- Vecteezy [45]

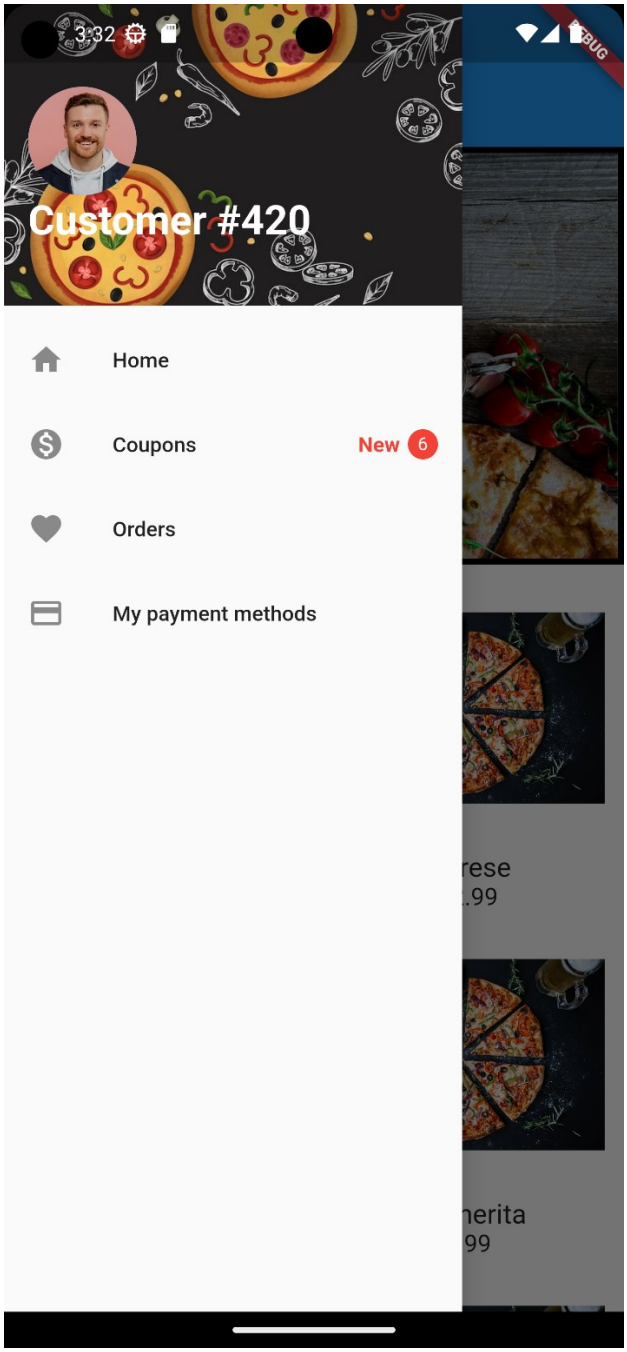
Užívateľské rozhranie pre Flutter je vyobrazené na obrázkoch 9, 10 a 11.



Obrázok 9 Flutter - Hlavná stránka a Coupons stránka aplikácie

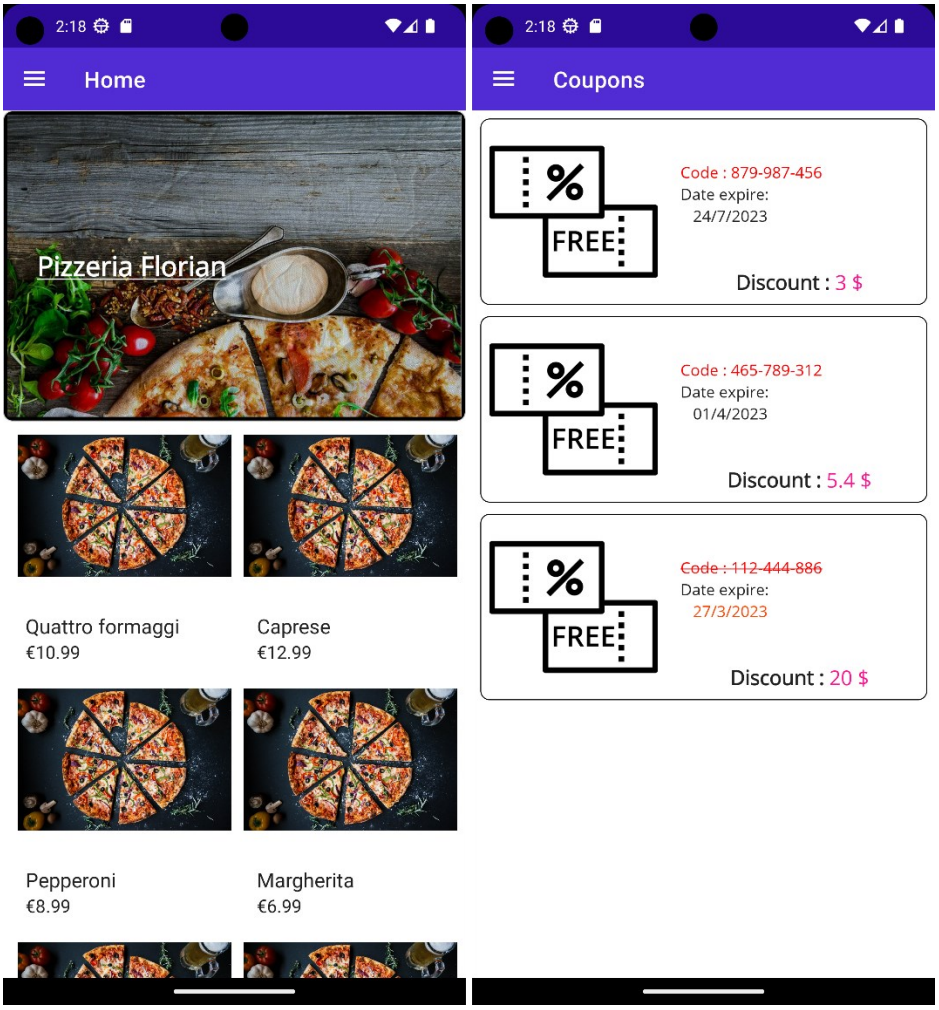


Obrázok 10 Flutter – Orders stránka a Payment Methods stránka aplikácie

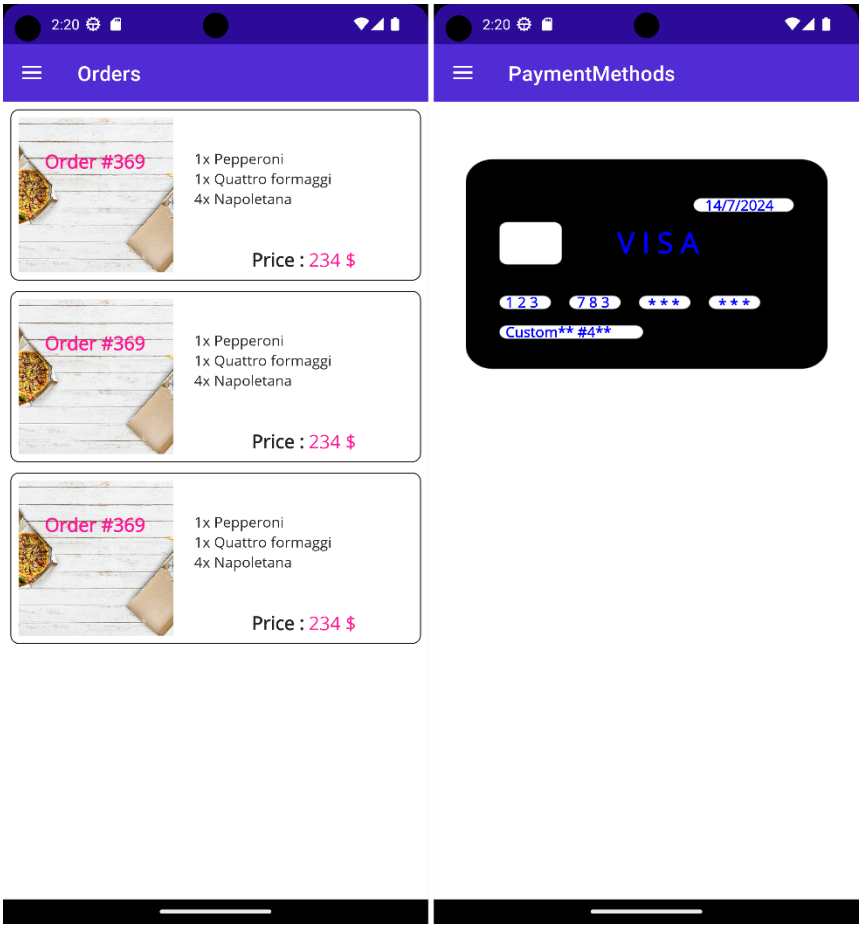


Obrázok 11 Flutter - Postranné menu aplikácie

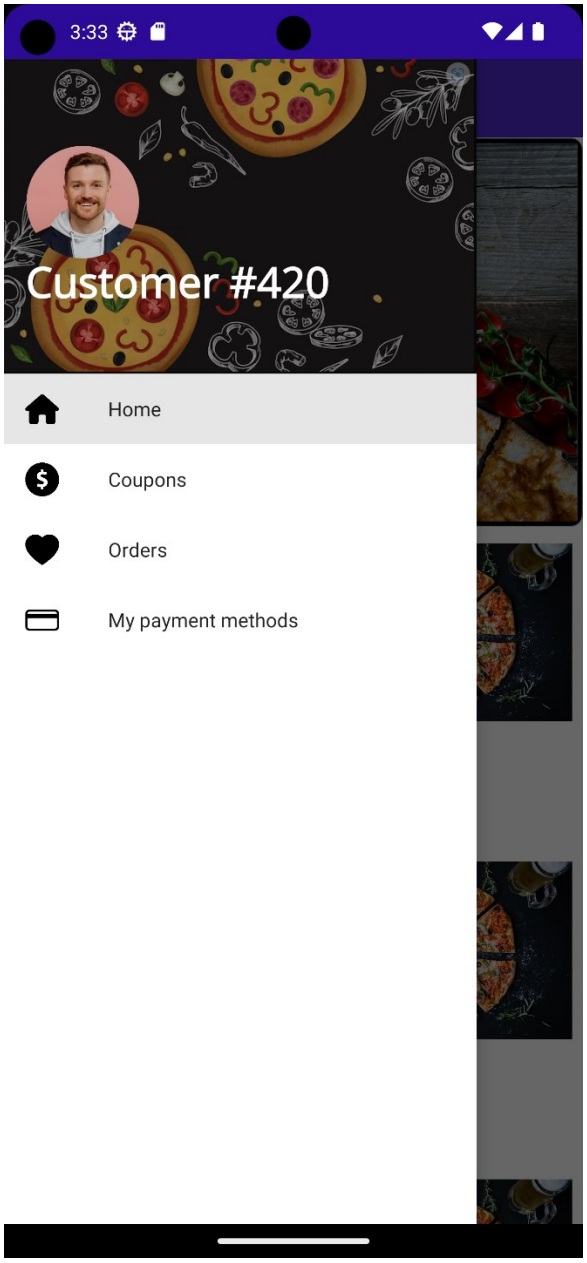
Užívateľské rozhranie pre .NET MAUI je vyobrazené na obrázkoch 12,13 a 14.



Obrázok 12 .NET MAUI - Hlavná stránka a Coupons stránka aplikácie



Obrázok 13 .NET MAUI – Orders stránka a Payment Methods stránka aplikácie



Obrázok 14 .NET MAUI - Postranné menu aplikácie

4.3.1 Flutter Navigation menu aplikácia

Aby sme mohli pracovať s CreditCardWidgetom musíme pridať závislosti do našej aplikácie.

```
dependencies:
  flutter:
    sdk: flutter
  flutter_credit_card: ^3.0.6
```

Vytvoríme triedu PaymentMethods a jej užívateľské rozhranie je tvorené pomocou widgetu CreditCardWidget (viď Obrázok 10). Aby sme mohli používať CreditCardWidget tak musíme importovať balíček flutter_credit_card.dart.

```
import 'package:flutter_credit_card/flutter_credit_card.dart';
```

Widget CreditCardWidget obsahuje informácie o kreditnej karte ako číslo karty, dátum platnosti karty, meno vlastníka karty a cvv číslo.

```
CreditCardWidget(
  cardType: CardType.visa,
  cardBgColor: Colors.black,
  isHolderNameVisible: true,
  cardNumber: '123783******',
  expiryDate: '14/7/2024',
  cardHolderName: 'Custom** #4**',
  cvvCode: '678',
  showBackView: false,
  onCreditCardWidgetChange: (CreditCardBrand ) {  },
),
```

Trieda Orders slúži na zobrazenie objednávok (viď Obrázok 10) a jej užívateľské rozhranie je zobrazené pomocou widgetu Column.

Samotná objednávka je definovaná pomocou widgetu Card, ktorý obsahuje niekoľko widgetov. Widget Stack je využitý na umiestnenie widgetov cez seba a obsahuje dva widgety a to Image slúžiaci na zobrazenie obrázku, ktorý sa načíta z adresy poskytnutej v parametroch NetworkImage a widget Text ktorý je zobrazený na obrázku pomocou widgetu Positioned.

```
Stack(
  children: const [
    Image(
      image: NetworkImage('https://img.freepik.com/free-photo/eating-
pizza-with-social-distancing-new-normal-way_53876-
98423.jpg?w=1380&t=st=1680899016~exp=1680899616~hmac=474b01970178018ef027
191f1ded5fd3f5642b2b2caa3c5ce8b858cae2225370'),
      height: 150,
      width: 150,
      fit: BoxFit.cover,
    ),
    Positioned(
      top: 30,
      left: 32,
```

```
      child: Text(
        "Order #369",
        style: TextStyle(
          color: Colors.pink,
          fontSize: 18,
        ),...
```

Widget Column obsahuje niekoľko widgetov Text ktoré reprezentujú objednané položky v objednávke.

```
Column(
  crossAxisAlignment: CrossAxisAlignment.start,
  children: const [
    Text("1x Pepperoni"),
    Text("1x Quattro formaggi"),
    Text("4x Napoletana"),
  ],
),
```

Widget Row slúži na zobrazenie viacerých Text widgetov vedľa seba v rade a pomocou MainAxisAlignment.end sú zarovnané na pravú stranu.

```
Row(
  mainAxisAlignment: MainAxisAlignment.end,
  children: const [
    Text(
      'Price : ',
      style: TextStyle(
        fontWeight: FontWeight.bold,
        fontSize: 16,
      ),
    ),
    Text(
      '234 \$',
      style: TextStyle(
        color: Colors.red,
        fontSize: 16,
      ),
    ),
  ],
),
```

Trieda Coupons slúži na zobrazenie kupónov (viď Obrázok 9) a jej užívateľské rozhranie je zobrazené pomocou widgetu Column.

Samotný kupón je definovaná pomocou widgetu Card, ktorý obsahuje niekoľko widgetov. Widget Row obsahuje widgety Image slúžiaci na zobrazenie obrázka, ktorý sa načíta

z adresy poskytnutej v parametroch NetworkImage a widget Column slúži na zobrazenie viacerých Text widgetov pod sebou.

```
Row(
  crossAxisAlignment: CrossAxisAlignment.start,
  children: [
    const Image(
      image: NetworkImage('https://cdn-icons-png.flati-
con.com/512/819/819178.png?w=826&t=st=1680893122~exp=1680893722~hmac=9f04
2e6c25adbeed56064873d172e9eb8b44a3a11b0279653517eb36ad5cacaf'),
      height: 150,
      fit: BoxFit.cover,
    ),
    SizedBox(width: 8,),
    Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: const [
        Text(
          'Code : 879-987-456',
          style: TextStyle(color: Colors.red),
        ),
        Text('Date expire:'),
        Text('24/7/2023'),
      ],
    ),
  ],
),
```

Widget Positioned slúži na presné umiestnenie vnútorného widgetu Row obsahujúceho dva textové widgety umiestnené vedľa seba pomocou MainAxisAlignment.end.

```
Positioned(
  bottom: 0,
  right: 50,
  child: Row(
    mainAxisAlignment: MainAxisAlignment.end,
    children: const [
      Text(
        'Discount : ',
        style: TextStyle(
          fontWeight: FontWeight.bold,
          fontSize: 16,
        ),
      ),
      Text(
        '3 \$',
        style: TextStyle(
          color: Colors.red,
          fontSize: 16,
        ),
      ),
    ],
  ),
)
```

Trieda HomePage je hlavná stránka aplikácie (viď Obrázok 9) a jej užívateľské rozhranie je zobrazené pomocou widgetu SingleChildScrollView, ktoré poskytuje možnosť posúvania obsahu obrazovky.

Hlavička je vytvorená pomocou widgetu Stack obsahujúceho widgety Image a Text. Widget Image slúži na zobrazenie obrázka, ktorý sa načíta z adresy poskytnutej v parametroch NetworkImage. Widget Text je umiestnený na obrázku pomocou widgetu Positioned.

```
Stack(
  children: const [
    Image(
      image: NetworkImage("https://cdn.pi-
xabay.com/photo/2020/05/07/23/14/pizza-5143514_960_720.jpg"),
      fit: BoxFit.cover,
    ),
    Positioned(
      left: 30,
      top: 120,
      child: Text(
        'Pizzeria Florian',
        style: TextStyle(
          fontSize: 24,
          color: Colors.white,
          fontWeight: FontWeight.bold,
          decoration: TextDecoration.underline,
        ),
      ),
    ),
  ],
),
```

Hlavná stránka aplikácie obsahujem okrem hlavičky aj jednotlivé položky, vytvorené pomocou widgetu Column a obsahujúce widgety Image a Text.

```
Column(
  children: [
    Image.network("https://cdn.pixabay.com/photo/2017/12/09/08/18/pizza-
3007395_960_720.jpg", width: 190, height: 190),
    const Text("Caprese", style: TextStyle(fontSize: 18)),
    const Text("€12.99", style: TextStyle(fontSize: 16)),
  ],
```

Trieda NavBar reprezentuje postranné menu (viď Obrázok 11). Jeho užívateľské rozhranie je tvorené pomocou widgetu ListView.

Ako prvé je implementovaná hlavička pomocou widgetu UserAccountsDrawerHeader obsahujúca meno užívateľa, profilový obrázok a obrázok na pozadí.

```
UserAccountsDrawerHeader(
  accountName: const Text("Customer #420", style: TextStyle( fontWeight:
FontWeight.bold, fontSize: 27),),
  accountEmail: const Text(""),
  currentAccountPicture: CircleAvatar(
    child: ClipOval(
      child: Image.network(
        "https://static.vecteezy.com/system/resources/pre-
views/013/001/831/large_2x/portrait-of-cheerful-european-man-with-stub-
ble-dressed-in-hoodie-and-black-jacket-looks-directly-at-camera-has-
happy-face-expression-poses-against-pink-background-people-emotions-li-
festyle-free-photo.JPG",
        width: 90,
        height: 90,
        fit:  BoxFit.cover  ,
      ),
    ),
  ),
  decoration:
  const BoxDecoration(
      color: Colors.black ,
      image: DecorationImage(
        image: NetworkImage("https://img.freepik.com/free-vector/res-
taurant-mural-wallpaper-with-pizza_23-2148683829.jpg"),
        fit: BoxFit.cover,
      )
  ),
),
```

Následne sú definované jednotlivé záložky slúžiace ako tlačidlá, ktoré po kliknutí otvoria príslušnú obrazovku. Každá položka je definovaná pomocou ListTile.

```
ListTile(
  leading: const Icon(Icons.home),
  title: const Text('Home'),
  onTap: () {
    Navigator.push(
      context,
      MaterialPageRoute(builder: (context) => const MyHomePage()),
    );
  },
),
```

4.3.2 .NET MAUI Navigation menu aplikácia

Na užívateľské rozhranie triedy PaymentMethods (viď Obrázok 13) je použitý element Image a niekoľko Label elementov.

Jednotlivé Label elementy predstavujú príslušný text, umiestnený na konkrétnom miestne na obrázku pomocou vlastnosti Margin.

```
<Image
    Source="card.png"
    WidthRequest="350"
    HeightRequest="300"
    Aspect="Fill" />
<Label Text="14/7/2024"
        FontAttributes="Bold"
        FontSize="14"
        TextColor="Blue"
        Margin="255,83,0,0"/>
<Label Text="V I S A"
        FontAttributes="Bold"
        FontSize="28"
        TextColor="Blue"
        Margin="170,110,0,0"/>
<Label Text="1 2 3"
        FontAttributes="Bold"
        FontSize="14"
        TextColor="Blue"
        Margin="62,177,0,0"/>
<Label Text="7 8 3"
        FontAttributes="Bold"
        FontSize="14"
        TextColor="Blue"
        Margin="131,177,0,0"/>
<Label Text="* * *"
        FontAttributes="Bold"
        FontSize="14"
        TextColor="Blue"
        Margin="200,179,0,0"/>
<Label Text="* * *"
        FontAttributes="Bold"
        FontSize="14"
        TextColor="Blue"
        Margin="268,179,0,0"/>
<Label Text="Custom** #4**"
        FontAttributes="Bold"
        FontSize="14"
        TextColor="Blue"
        Margin="62,206,0,0"/>
```

Trieda Coupons slúži na zobrazenie kupónov (viď Obrázok 12) a jej užívateľské rozhranie je zobrazené pomocou VerticalStackLayout.

Samotný kupón je definovaný pomocou elementu Grid, ktorý sa skladá z dvoch riadkov a stĺpcov.

```
<Grid.RowDefinitions>
    <RowDefinition Height="Auto" />
    <RowDefinition Height="*" />
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
    <ColumnDefinition Width="Auto" />
    <ColumnDefinition Width="*" />
</Grid.ColumnDefinitions>
```

V rámci Gridu je definovaný Image a StackLayout.

Image má nastavenú výšku pomocou HeightRequest, šírku pomocou WidthRequest a AspectFill, čo zabezpečuje, že obraz upraví svoju veľkosť aby vyplnil celý priestor.

```
<Image
Source="ticket.png"
Grid.RowSpan="2"
WidthRequest="150"
HeightRequest="150"
Aspect="AspectFill" />
```

StackLayout sa skladá z VerticalStackLayout a HorizontalStackLayout elementu a tieto elementy obsahujú Label elementy slúžiace na zobrazenie kódu zľavového kupónu, dátum expirácie a hodnotu zľavy.

```
<StackLayout Grid.Column="1" Margin="20,30,0,0">
    <VerticalStackLayout>
        <Label Text="Code : 879-987-456" TextColor="Red"/>
        <Label Text="Date expire:" />
        <Label Text="   24/7/2023" />
    </VerticalStackLayout>
    <HorizontalStackLayout
        HorizontalOptions="Center"
        VerticalOptions="EndAndExpand">
        <Label Text="Discount : " FontAttributes="Bold" FontSize="Medium" />
        <Label Text="3 $" TextColor="DeepPink" FontSize="Medium"/>
    </HorizontalStackLayout>
</StackLayout>
```

Trieda Orders slúži na zobrazenie objednávok (viď Obrázok 13) a jej užívateľské rozhranie je zobrazené pomocou VerticalStackLayout.

Samotná objednávka je definovaná pomocou elementu Grid, ktorý sa skladá z dvoch riadkov a stĺpcov.

```
<Grid.RowDefinitions>
     <RowDefinition Height="Auto" />
     <RowDefinition Height="*" />
 </Grid.RowDefinitions>
 <Grid.ColumnDefinitions>
     <ColumnDefinition Width="Auto" />
     <ColumnDefinition Width="*" />
 </Grid.ColumnDefinitions>
```

V rámci Gridu je definovaný ďalší Grid a StackLayout.

Vnorený Grid obsahuje Image s nastavenou výškou pomocou HeightRequest a šírkou pomocou WidthRequest a AspectFill čo zabezpečuje, že obraz upraví svoju veľkosť aby vyplnil celý priestor a Label ktorý je umiestnený na obrázku pomocou vlastnosti Margin.

```
<Grid>
    <Image
    Source="pizzabox.jpg"
    Grid.RowSpan="2"
    WidthRequest="150"
```

```
    HeightRequest="150"
    Aspect="AspectFill" />
    <Label Text="Order #369"
           FontAttributes="Bold"
           FontSize="Medium"
           TextColor="DeepPink"
           Margin="25,30,0,0"/>
</Grid>
```

StackLayout sa skladá z VerticalStackLayout a HorizontalStackLayout elementu a tieto elementy obsahujú Label elementy slúžiace na zobrazenie objednaných položiek a celkovú cenu objednávky.

```
<VerticalStackLayout>
    <Label Text="1x Pepperoni"  />
    <Label Text="1x Quattro formaggi" />
    <Label Text="4x Napoletana" />
</VerticalStackLayout>
<HorizontalStackLayout
    HorizontalOptions="Center"
    VerticalOptions="EndAndExpand">
    <Label Text="Price : " FontAttributes="Bold" FontSize="Medium" />
    <Label Text="234 $" TextColor="DeepPink" FontSize="Medium"/>
</HorizontalStackLayout>
```

Trieda MainPage je hlavná stránka aplikácie (viď Obrázok 12) a jej užívateľské rozhranie je zobrazené pomocou ScrollView poskytujúceho možnosť posúvania obsahu obrazovky.

Hlavička je vytvorená pomocou elementu Grid obsahujúceho element Frame ktorý slúži na orámovanie obrázku a Label umiestnený na obrázku.

```
<Grid>
    <Frame Padding="4" BackgroundColor="Black" >
        <Image Source="pizza_background_mainpage.jpg" Aspect="AspectFit"/>
    </Frame>
    <Label Text="Pizzeria Florian"
           HorizontalOptions="Start"
           VerticalOptions="Center"
           Padding="30"
           FontSize="Title"
           TextColor="White"
           FontAttributes="Bold"
           TextDecorations="Underline"
           />
</Grid>
```

Hlavná stránka aplikácie obsahujem, okrem hlavičky, aj jednotlivé položky vytvorené pomocou Gridu obsahujúce elementy Image a Label.

```
<Image Grid.Row="0" Source="pizza.jpg"
       WidthRequest="190" HeightRequest="190" />
<Label Grid.Row="1" Text="Quattro formaggi"
       FontSize="18" FontFamily="Bold" />
<Label Grid.Row="2" Text="€10.99"
       FontSize="16" FontFamily="Bold" />
```

Na vytvorenie postranného menu (viď Obrázok 14) musí byť upravený súbor AppShell.xaml.

Trieda NavBar reprezentuje postranné menu. Jeho užívateľské rozhranie je tvorené pomocou elementu Shell.

Ako prvé je implementovaná hlavička pomocou Shell.FlyoutHeader obsahujúca meno užívateľa, profilový obrázok a obrázok na pozadí.

```
<Shell.FlyoutHeader>
    <Grid BackgroundColor="Black">
        <Image Aspect="AspectFill"
           Source="pizza_background.png"
           Opacity="0.6" />

        <Image Source="customer.png"
        Aspect="AspectFill"
        HeightRequest="80"
        WidthRequest="80"
        HorizontalOptions="EndAndExpand"
        Margin="0,0,240,20">
            <Image.Clip>
                <EllipseGeometry Center="40,40" RadiusX="40" RadiusY="40" />
            </Image.Clip>
        </Image>

        <StackLayout VerticalOptions="EndAndExpand"
                     Margin="0,0,0,30" Padding="15">
            <Label Text="Customer #420"
           TextColor="White"
           FontSize="30"
           FontAttributes="Bold"
           HorizontalOptions="Start"/>
        </StackLayout>
    </Grid>
</Shell.FlyoutHeader>
```

Následne sú definované jednotlivé záložky slúžiace ako tlačidlá, ktoré po kliknutí otvoria príslušnú obrazovku. Každá položka je definovaná pomocou FlyoutItem.

```
<FlyoutItem
    Title="Home"
    Icon="house_solid.svg">
    <ShellContent ContentTemplate="{DataTemplate local:MainPage}"/>
</FlyoutItem>
```

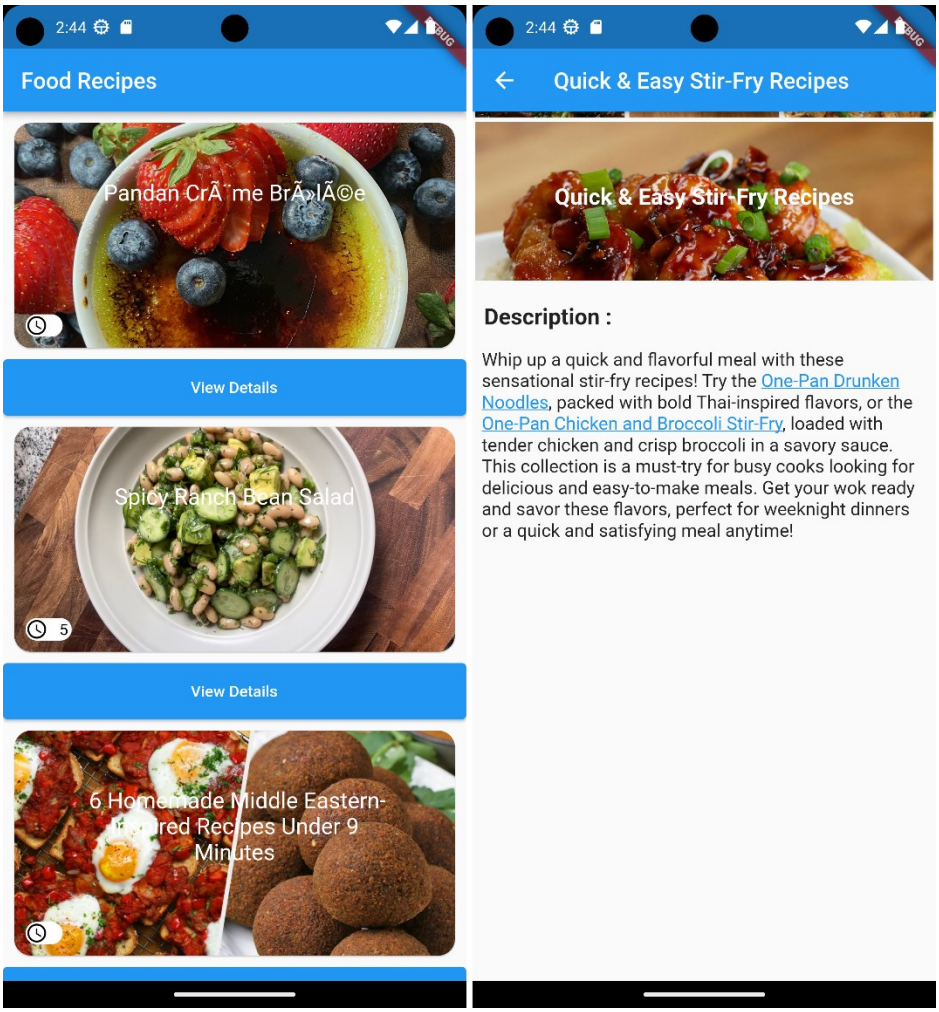
4.3.3 Porovnanie riešení

Pri vývoji aplikácie Navigation menu bola práca s obomi frameworkami príjemná a plynulá. Veľkou výhodou pri frameworku .NET MAUI bolo to, že nie je nutné vytvárať nový súbor pre postranné menu, čo ušetrí čas a zjednoduší prácu s aplikáciou. Na druhej strane veľkou nevýhodou frameworku .NET MAUI je to, že neumožňuje pridať dva názvy do jednej záložky (viď Obrázok 14 a 11), čo môže byť obmedzujúce pre určité druhy aplikácií.

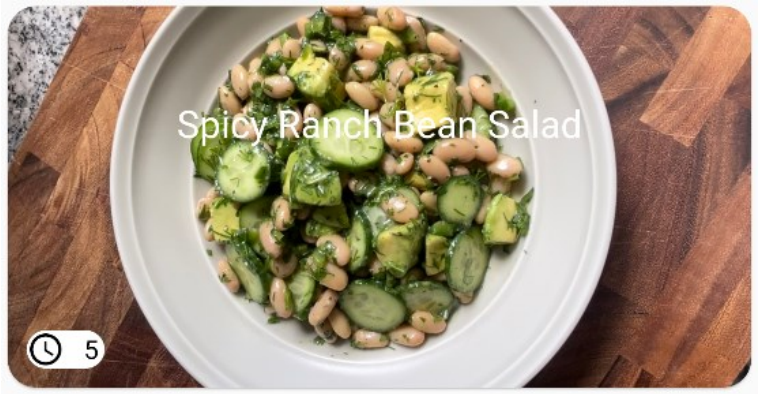
4.4 Recipe aplikácia

Cieľom tejto aplikácie je zobrazenie receptov s možnosťou kliknutia na tlačidlo detailu. Jednotlivé recepty bude zobrazené v okne ktoré obsahuje obrázok, názov receptu a čas prípravy. Obrazovka s detailom receptu bude obsahovať obrázok, názov receptu a popis receptu. Všetky tieto údaje budú dostupné z API. Pre použitie tejto API bude potrebné vytvoriť API kľúč na webovej stránke [46].

Užívateľské rozhranie pre Flutter je vyobrazené na obrázkoch 15 a 16.

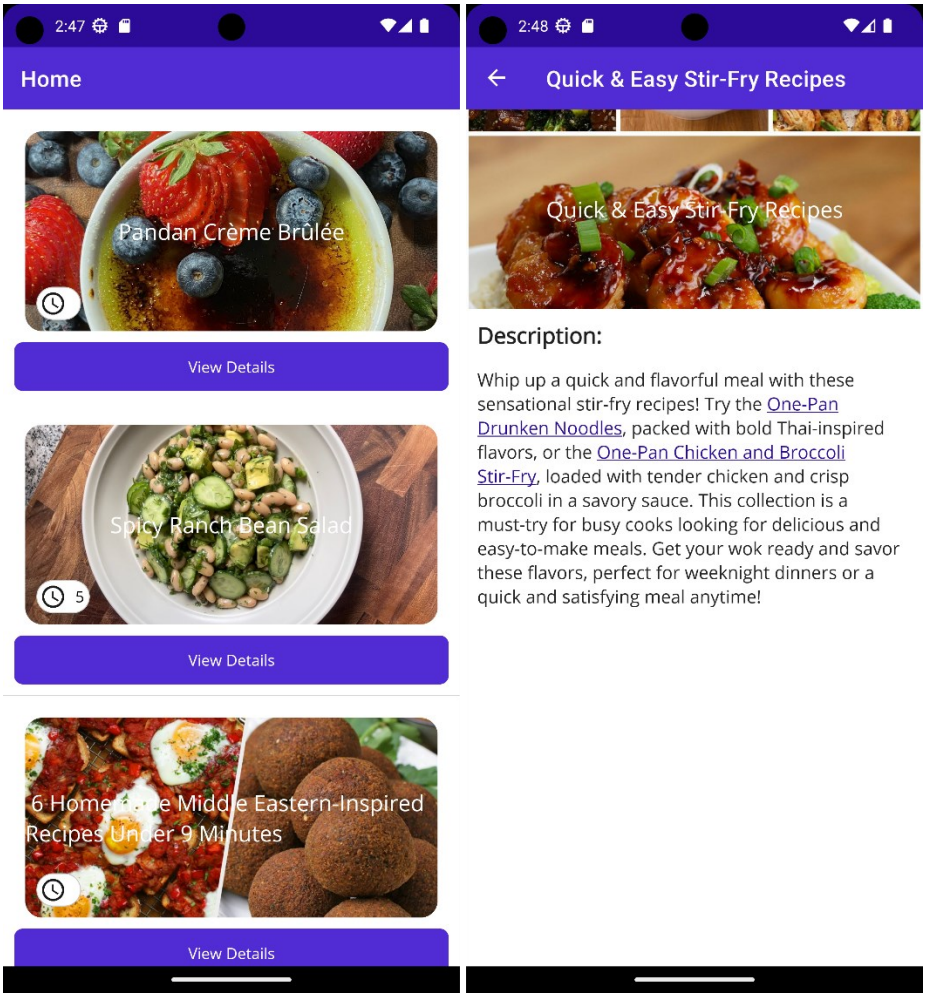


Obrázok 15 Flutter – Hlavná stránka a stránka detailu receptu aplikácie



Obrázok 16 Flutter – Karta receptu

Užívateľské rozhranie pre .NET MAUI je vyobrazené na obrázkoch 17 a 18.



Obrázok 17 .NET MAUI – Hlavná stránka a stránka detailu receptu aplikácie

Obrázok 18 .NET MAUI – Karta receptu

4.4.1 Flutter Recipe aplikácia

Aby sme mohli pracovať s http a flutter_html widgetom musíme pridať závislosti do našej aplikácie.

```
dependencies:
  flutter:
    sdk: flutter
  http: ^0.13.1
  flutter_html: ^2.1.5
```

Na správne fungovanie flutter_html widgetu musíme upraviť defaultConfig v súbore build.gradle aby podporoval minimálnu verziu Sdk 19.

```
minSdkVersion 19
```

Vytvoríme triedu RecipeApi obsahujúce metódu getRecipes.

V tejto metóde získame zoznam receptov pomocou HTTP get požiadavky a následne prevedieme odpoveď do JSON formátu. Na záver sa vytvorí prázdny zoznam dataList a pre každý záznam receptu sa pridá tento záznam do dataList.

Nakoniec sa zavolá metóda createRecipesFromData, ktorá prekonvertuje záznamov receptov na zoznam tried Recipe.

```
class RecipeApi {
  static Future<List<Recipe>> getRecipes() async{
    var url = Uri.https('tasty.p.rapidapi.com', '/recipes/list', {'from':
'0', 'size': '20'});
    final headers = {
    'X-RapidAPI-Key': 'API KLÚČ',
    'X-RapidAPI-Host': 'tasty.p.rapidapi.com',
    "useQueryString": "true"
    };

    final response = await http.get(url, headers: headers);
    List json = jsonDecode(response.body)['results'];
```

```
    List dataList = [];
    for (var i in json){
      dataList.add(i);
    }

    return Recipe.createRecipesFromData(dataList);
  }
}
```

Vytvoríme triedu Recipe ktorá bude mať štyri premenné a to recipeName, imageUrl, description a cookTimeInMinutes, ktoré sú inicializované v konštruktore.

```
class Recipe {
  final String recipeName;
  final String imageUrl;
  final String description;
  final int cookTimeInMinutes;

  Recipe({
    this.recipeName,
    this.imageUrl,
    this.description,
    this.cookTimeInMinutes,
  });
```

Metóda convertJson typu factory slúži na vytvorenie inštancie triedy Recipe.

```
factory Recipe.convertJson(dynamic json) {
  return Recipe(
    description: json['description'] as String,
    recipeName: json['name'] as String,
    cookTimeInMinutes: json['cook_time_minutes'] as int,
    imageUrl: json['thumbnail_url'] as String,
  );
}
```

Metóda createRecipesFromData slúži na vytvorenie zoznamu receptov a pre každý recept volá metódu convertJson.

```
static List<Recipe> createRecipesFromData(List data) {
  return data.map((json) {
    return Recipe.convertJson(json);
  }).toList();
}
```

Trieda RecipeCard obsahuje tri parametre a to recipeName, timeForCook a imageUrl sú inicializované v konštruktore a slúžia na zobrazenie receptu (viď Obrázok 16) a ich užívateľské rozhranie je zobrazené pomocou widgetu Card.

Widget Stack je využitý na umiestnenie widgetov cez seba a obsahuje tri widgety. Tieto widgety sú Image, Align a widget Positioned.

Image slúži na zobrazenie obrázku.

```
Image.network(
  imageUrl,
  height: 200,
  width: double.infinity,
  fit: BoxFit.cover,
),
```

Na zobrazenie widgetu Text na stred obrázku slúži widget Align.

```
Align(
  alignment: Alignment.center,
  child: Padding(
    padding: const EdgeInsets.all(50),
    child: Text(
      recipeName,
      style: const TextStyle(
        fontSize: 20,
        color: Colors.white,
      ),
      textAlign: TextAlign.center,
    ),
  ),
),
```

Widget Positioned obsahuje widget Icon a widget Text slúžiace na zobrazenie času prípravy receptu ak hodnota timeForCook nie je null.

```
Positioned(
  left: 10,
  bottom: 10,
  child: Container(
    decoration: BoxDecoration(
      color: Colors.white,
      borderRadius: BorderRadius.circular(10),
    ),
    child: Row(
      children: [
        const Icon(
          Icons.schedule,
          color: Colors.black,
          size: 20,
        ),
        const SizedBox(width: 10),
        Text(
          timeForCook != 'null' ? timeForCook : '',
          style: const TextStyle(color: Colors.black),
        ),
        const SizedBox(width: 3),
      ],
   ...
```

Trieda RecipeDetail obsahuje štyri parametre a to title, timeforCook, description a url ktoré sú inicializované v konštruktore a slúžia na zobrazenie detailnej stránky receptu (viď Obrázok 15). Ich užívateľské rozhranie je zobrazené pomocou widgetu Column.

```
class RecipeDetail extends StatelessWidget {
  final String title;
  final String timeForCook;
  final String description;
  final String url;

  RecipeDetail({
    this.title,
    this.timeForCook,
    this.description,
    this.url});
```

Na zobrazenie obrázka s textom umiestnený na stred slúži widget Container.

```
Container(
  height: 150,
  decoration: BoxDecoration(
    image: DecorationImage(
      image: NetworkImage(url),
      fit: BoxFit.cover,
    ),
  ),
  child: Center(
    child: Padding(
      padding: const EdgeInsets.all(10),
      child: Text(
        title,
        style: const TextStyle(
          fontWeight: FontWeight.bold,
          fontSize: 20,
          color: Colors.white,
        ),
        textAlign: TextAlign.center,
      ),
     ...
```

Zobrazenie popisu receptu je realizované pomocou widgetu Html, ktorý umožňuje zobrazenie HTML značiek.

```
Html(
  data: description,
  style: {
    'body': Style(fontSize: FontSize(16.0)),
  },
),
```

Za zobrazenie hlavnej stránky aplikácie je zodpovedná trieda MyHomePage (viď Obrázok 15). Užívateľské rozhranie je realizované po widgetu ListView.builder.

Widget ListView.builder zobrazuje jednotlivé recepty spoločne s tlačidlom ktoré po kliknutý otvorí detailnú obrazovku receptu.

Zobrazenie receptu je umožnené pomocou triedy RecipeCard.

```
RecipeCard(
  recipeName: recipes[index].recipeName,
  timeForCook: recipes[index].cookTimeInMinutes.toString(),
  imageUrl: recipes[index].imageUrl,
),
```

Tlačidlo na zobrazenie detailnej obrazovky je realizované pomocou widgetu ElevatedButton.

```
ElevatedButton(
  onPressed: () {
    Navigator.push(
      context,
      MaterialPageRoute(
        builder: (context) => RecipeDetail(
          title: recipes[index].recipeName,
          timeForCook: 5.toString(),
          description: recipes[index].description,
          url: recipes[index].imageUrl,
        ),
      ),
    );
  },
  child: const Text('View Details'),
  style: ElevatedButton.styleFrom(
    minimumSize: const Size(double.infinity, 50),
  ),
),
```

4.4.2 .NET MAUI Recipe aplikácia

Na zobrazenie jednotlivých receptov slúži trieda recipe_card a jej užívateľské rozhranie (viď Obrázok 18) je definované pomocou Grid elementu. Grid element obsahuje Frame element na orámovanie obrázku, Label element na zobrazenie textu umiestneného na stred obrazovky. Ďalším element Frame ktorý slúži na orámovanie elementu StackLayout ktorý obsahuje element Label na zobrazenie času prípravy a Image element na zobrazenie obrázku.

```
<Grid HeightRequest="180" Margin="10">
    <Frame CornerRadius="15" Padding="0">
        <Image Source="{Binding Image}" Aspect="AspectFill"/>
    </Frame>
    <Label Text="{Binding Title}" HorizontalOptions="Center" VerticalOp-
tions="Center" TextColor="White" FontSize="20"/>
    <Frame Opacity="0.4" CornerRadius="15" Padding="0" HorizontalOp-
tions="Start" VerticalOptions="End" Margin="10">
        <StackLayout Orientation="Horizontal">
```

```
            <Image Source="https://cdn-icons-png.flati-
con.com/512/60/60730.png" HeightRequest="20" Margin="5" BackgroundCo-
lor="White"/>
            <Label Text="{Binding CookTimeInMinutes}" Padding="5" TextCo-
lor="Black"/>
        </StackLayout>
    </Frame>
</Grid>
```

Na zobrazenie obrazovky s detailom receptu slúži trieda RecipeDetailPage. V konštruktore triedy RecipeDetailPage musíme nastaviť BindingContext umožňujúci prepojenie dát s prvkami na obrazovke.

```
public RecipeDetailPage(MyData data)
{
    InitializeComponent();
    BindingContext = data;
}
```

Užívateľské rozhranie triedy RecipeDetailPage zobrazuje detailnú obrazovku receptu (viď Obrázok 17) a je definované pomocou StackLayout elementu. StackLayout element obsahuje Grid element a další StackLayout element.

Grid element obsahuje obrázok a text umiestnený na stred obrázku.

```
<Grid HeightRequest="180" >
    <Image Source="{Binding Image}" Aspect="AspectFill"/>
    <Label Text="{Binding Title}" HorizontalOptions="Center"
           VerticalOptions="Center"
           TextColor="White" FontSize="20"/>
</Grid>
```

StackLayout zobrazuje popis daného receptu.

```
<StackLayout Padding="10">
    <Label Text="Description:" FontAttributes="Bold" FontSize="20"/>
    <BoxView HeightRequest="15" Color="White"/>
    <Label TextType="Html" Text="{Binding Description}" FontSize="16"/>
</StackLayout>
```

Trieda MyData slúži k reprezentácii dát receptu. Obsahuje vlastnosti ako názov receptu, url adresu na obrázok, dobu prípravy receptu a popis. Tieto vlastnosti sú označené atribútom JsonPropertyName.

```
public class MyData
{
    [JsonPropertyName("name")]
    public string Title { get; set; }

    [JsonPropertyName("thumbnail_url")]
    public string Image { get; set; }

    [JsonPropertyName("cook_time_minutes")]
    public int? CookTimeInMinutes { get; set; }

    [JsonPropertyName("description")]
    public string Description { get; set; }
}
```

Za zobrazenie hlavnej stránky aplikácie je zodpovedná trieda MainPage.

Asynchrónna metóda GetDataFromApiAsync slúži na získanie receptov z API. Na získanie dát z API sa použije inštancia triedy HttpClient, ktorá odošle požiadavku a následne sa získaná odpoveď sa spracuje a prevedie sa do zoznamu objektu typu MyData.

```
private async Task<List<MyData>> GetDataFromApiAsync()
{
    var client = new HttpClient();
    client.DefaultRequestHeaders.Add(
        "X-RapidAPI-Key", "API KLÚČ");
    client.DefaultRequestHeaders.Add(
        "X-RapidAPI-Host", "tasty.p.rapidapi.com");
    client.DefaultRequestHeaders.Add(
        "useQueryString", "true");

    var response = await client.GetAsync(
        "https://tasty.p.rapidapi.com/recipes/list?from=0&size=20");

    var json = await response.Content.ReadAsStringAsync();
    var jsonData = JsonDocument.Parse(json).
        RootElement.GetProperty("results").ToString();
    var options = new JsonSerializerOptions
    { PropertyNamingPolicy = JsonNamingPolicy.CamelCase };
    var data = JsonSerializer.Deserialize<List<MyData>>(jsonData, options);

    return data;
}
```

Užívateľské rozhranie hlavnej stránky (viď Obrázok 17) je realizované pomocou ListView elementu ktorý zobrazuje jednotlivé recepty a tlačidlá na zobrazenie detailu receptu.

```
<ListView x:Name="MyListView" HasUnevenRows="true" SelectionMode="None" >
    <ListView.ItemTemplate>
        <DataTemplate>
            <ViewCell  >
                <StackLayout Padding="10" VerticalOptions="FillAndExpand" Ho-
rizontalOptions="FillAndExpand" >
                    <local:recipe_card />
                    <Button Text="View Details" Clicked="OnViewDetailsClicked"
CommandParameter="{Binding .}" />
                </StackLayout>
            </ViewCell>
        </DataTemplate>
    </ListView.ItemTemplate>
</ListView>
```

4.4.3 Porovnanie riešení

Pri vývoji aplikácie Recipe bola práca s obomi frameworkami príjemná a bezproblémová. Pri použití frameworku .NET MAUI bola jednou z výhod absencia nutnosti inštalácie ďalšej knižnice na prácu s textom obsahujúcim HTML značky. Na druhej strane, framework Flutter vyžaduje zmenu minimálnej verzie SDK pre prácu s widgetom flutter_html, ktorý slúži na zobrazenie textu obsahujúceho HTML značky.

5 ZHODNOTENIE FRAMEWORKOV

Pri vývoji aplikácii v frameworku .NET MAUI a Flutter bola práca veľmi zaujímavá a oba frameworky majú veľa čo ponúknuť. Framework .NET MAUI a Flutter máju veľké množstvo výhod ale zároveň bolo možné identifikovať aj nedostatky. Porovnanie bolo vykonané na vzorových aplikáciách: TODO aplikácia, Google Maps aplikácia, Navigation aplikácia a Recipe aplikácia. Pri vytváraný týchto aplikácii nastali určité problémy. Pri aplikácii Google Maps pre framework .NET MAUI nebolo možné otočenie uhol kamery, pri frameworku Flutter bolo nutnosť zmenenia minimálnej verzie SDK pre použitie niektorých widgetov. V aplikácii TODO dominoval framework Flutter z dôvodu, že práca s lokálnou databázou Hive bola príjemnejšia a jej spracovanie bolo jednoduchšie a intuitívnejšie ako práca s lokálnou databázou SQLite pre framework .NET MAUI. Na druhej strane, veľkou výhodou u frameworku .NET MAUI v rámci vývoja aplikácie Navigation bolo to, že nebolo potrebné vytvárať nový súbor pre vytvorenie navigačného menu, čo pomohlo k zjednodušeniu vývoja tejto aplikácie.

.NET MAUI je stále relatívne nový framework, ktorý obsahuje nemálo chyb a nedokonalosti. Avšak, pri existujúcich skúsenostiach s .NET technológiami je jednoznačnou voľbou .NET MAUI. Prechod z iných .NET technológií na .NET MAUI by mal byť jednoduchý vďaka veľkej podobnosti medzi nimi. Napriek niektorým rozdielom by mal byť prechod plynulý a nebude ťažké si zvyknúť na nové technológie a funkcie ktoré sebou prináša .NET MAUI. Toto umožňuje vývojárom ľahko prejsť na novú platformu a využiť svoje existujúce znalosti a skúsenosti. To je jedným z dôvodov, prečo sa očakáva, že .NET MAUI bude v budúcnosti veľmi rozsiahlym a obľúbeným frameworkom v rámci .NET technológii.

Pri nulových skúsenostiach s .NET technológiami by som skorej odporúčal framework Flutter, z dôvodu rozsiahlej ponuky widgetov a prehľadnej a dostupnej dokumentácie. Okrem toho, framework Flutter nie je príliš náročný na naučenie efektívneho používania, dokonca aj pre neskúsených vývojárov. Má pomerne malú krivku učenia, čo znamená, že je relatívne jednoduchý na osvojenie. Je dôležité zdôrazniť, že Flutter je relatívne nový framework, avšak má svetlú budúcnosť vďaka veľkej komunite vývojárov a bohatého rozsahu widgetov.

ZÁVER

Cieľom tejto práce bolo porovnanie frameworkov .NET Maui a Flutter z hľadiska vývoja mobilných aplikácií. V teoretickej časti sme sa zamerali na prehľad vývoja mobilných aplikácií vrátane ich histórie. Rozoberali sme natívne a multiplatformové spôsoby vývoja mobilných aplikácií a popísali ich výhody a nevýhody. Bola venovaná pozornosť aj inovatívnym technológiám ako cloud computing, umelá inteligencia a progresívne webové aplikácie. Ďalej bola práca zameraná na popis frameworkov .NET MAUI a Flutter, ich fungovanie, kľúčové vlastnosti a bezpečnosť. V poslednej kapitole teoretickej časti bol zahrnutý krátky popis vývojových prostredí Visual Studio a Android Studio, ktoré sú vhodné pre vývoj mobilných aplikácií pomocou frameworkov .NET MAUI a Flutter.

V praktickej časti boli vytvorené vzorové aplikácie: TODO aplikácia, Google Maps aplikácia, Navigation aplikácia a Recipe aplikácia. V rámci jednotlivých ukážkových aplikácii v oboch frameworkoch boli popísané kľúčové časti kódu, bolo vykonané ich porovnanie a následne boli zhodnotené. V poslednej časti bolo popísané celkové zhodnotenie frameworkov a odporučený pro tvorbu nových aplikácii.

Výsledkom praktickej časti tejto bakalárskej práce je zistenie, že oba frameworky majú svoje výhody, avšak tiež boli identifikované nedostatky. V porovnaní, ktoré sme vykonali na vzorových aplikáciách (TODO aplikácia, Google Maps aplikácia, Navigation aplikácia a Recipe aplikácia), sme zaznamenali niekoľko problémov. Aj napriek niektorým chybám a nedostatkom, .NET MAUI ponúka jednoduchý prechod z iných .NET technológií a vývojári môžu využiť svoje existujúce skúsenosti. Na druhej strane, Flutter je skvelou voľbou pre vývojárov bez skúseností s vývojom vývojom mobilných aplikácií. Vďaka jeho relatívne jednoduchej krivke učenia umožňuje rýchle osvojenie a efektívne používanie tohto frameworku.

ZOZNAM POUŽITEJ LITERATÚRY

[1] HISTORY COMPUTER STAFF. Simon Personal Communicator: A Complete Guide. *History-Computer* [online]. 2022 [cit. 2023-05-14]. Dostupné z: https://history-computer.com/simon-personal-communicator/

[2] RAJPUT, Mehul. Tracing the History and Evolution of Mobile Apps. *Tech* [online]. 2021 [cit. 2023-05-14]. Dostupné z: https://tech.co/news/mobile-app-history-evolution-2015-11

[3] Apple Reinvents the Phone with iPhone. *Apple* [online]. 2007 [cit. 2023-05-14]. Dostupné z: https://www.apple.com/newsroom/2007/01/09Apple-Reinvents-the-Phone-with-iPhone/

[4] CHOVAŇÁK, Fero. Prvý telefón s Androidom bol predstavený pred 5 rokmi. *Mojandroid* [online]. 2013 [cit. 2023-05-14]. Dostupné z: https://www.mojandroid.sk/prvy-telefon-s-androidom-bol-predstaveny-pred-5-rokmi/

[5] PALUŠ, Tony. Pozrite si celú históriu vývoja mobilných aplikácií od IBM Simon po Snapchat : Infografika. *Mojandroid* [online]. 2017 [cit. 2023-05-14]. Dostupné z: https://www.mojandroid.sk/infografika-vyvoj-mobilne-aplikacie/

[6] KOĎOUSKOVÁ, Barbora. 11 Trendů ve vývoji mobilních aplikací v roce 2023. *Rascasone* [online]. 2023 [cit. 2023-05-14]. Dostupné z: https://www.rascasone.com/cs/blog/trendy-vyvoj-mobilnich-aplikaci

[7] What Are the Different Types of Mobile Apps?. *Duckma* [online]. © 2023 [cit. 2023-05-14]. Dostupné z: https://blog.duckma.com/en/types-of-mobile-apps/

[8] KATONA, Ján. 15 dôvodov, prečo firmy potrebujú mobilnú aplikáciu. *Elite Blog* [online]. 2020 [cit. 2023-05-14]. Dostupné z: https://www.eliteml.sk/blog/15-dovodov-preco-firmy-potrebuju-mobilnu-aplikaciu/

[9] STALLINGS, William a Lawrie BROWN. *Computer security: principles and practice*. Third edition. Boston: Pearson, 840 s. Always learning. ISBN 9781292066172.

[10] SCHMITT, Jacob. Native vs cross-platform mobile app development. *Circleci* [online]. 2022 [cit. 2023-05-14]. Dostupné z: https://circleci.com/blog/native-vs-cross-platform-mobile-dev/

[11] Native, hybrid, or cross-platform apps?. *Microsoft* [online]. © 2023 [cit. 2023-05-14]. Dostupné z: https://powerapps.microsoft.com/en-us/native-vs-cross-platform-apps/

[12] What's The Difference Between Web Apps, Native Apps, And Hybrid Apps?. *Amazon Web Services* [online]. © 2023 [cit. 2023-05-14]. Dostupné z: https://aws.amazon.com/compare/the-difference-between-web-apps-native-apps-and-hybrid-apps/

[13] MANNOTRA, Vivek. How to build Cross-Platform Mobile Apps. *BrowserStack* [online]. 2023 [cit. 2023-05-14]. Dostupné z: https://www.browserstack.com/guide/build-cross-platform-mobile-apps

[14] What is cloud computing?. *Microsoft* [online]. © 2023 [cit. 2023-05-14]. Dostupné z: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-cloud-computing/

[15] What is cloud computing?. *Amazon Web Services* [online]. © 2023 [cit. 2023-05-14]. Dostupné z: https://aws.amazon.com/what-is-cloud-computing/

[16] What is Artificial Intelligence (AI)?. *Google Cloud* [online]. [cit. 2023-05-14]. Dostupné z: https://cloud.google.com/learn/what-is-artificial-intelligence

[17] KHAN, Waleed. How To Use Artificial Intelligence In Mobile Apps. *ELearning Industry* [online]. 2022 [cit. 2023-05-14]. Dostupné z: https://elearningindustry.com/how-to-use-artificial-intelligence-in-mobile-apps

[18] RICHARD, Sam a Pete LEPAGE. What are Progressive Web Apps?. *Web.dev* [online]. 2020 [cit. 2023-05-14]. Dostupné z: https://web.dev/what-are-pwas/

[19] Introduction to progressive web apps. *MDN Web Docs* [online]. 2023 [cit. 2023-05-14]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Tutorials/js13kGames/Introduction

[20] What Is a Framework?. *Codecademy* [online]. 2021 [cit. 2023-05-14]. Dostupné z: https://www.codecademy.com/resources/blog/what-is-a-framework/

[21] Most loved, dreaded, and wanted. *Stackoverflow* [online]. 2022 [cit. 2023-05-14]. Dostupné z: https://survey.stackoverflow.co/2022/#technology-most-loved-dreaded-and-wanted

[22] Co je .NET MAUI?. *Microsoft* [online]. 2023 [cit. 2023-05-14]. Dostupné z: https://learn.microsoft.com/cs-cz/dotnet/maui/what-is-maui?view=net-maui-7.0

[23] Data binding and MVVM. *Microsoft* [online]. 2023 [cit. 2023-05-14]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/maui/xaml/fundamentals/mvvm

[24] Data binding and MVVM. *Github* [online]. 2022 [cit. 2023-05-14]. Dostupné z: https://github.com/MicrosoftDocs/windows-dev-docs/blob/docs/hub/apps/develop/data-binding/data-binding-and-mvvm.md

[25] Secure storage. *Microsoft* [online]. 2023 [cit. 2023-05-14]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/maui/platform-integration/storage/secure-storage?view=net-maui-7.0&tabs=windows

[26] Web authenticator. *Microsoft* [online]. 2023 [cit. 2023-05-14]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/maui/platform-integration/communication/authentication?view=net-maui-7.0&tabs=windows

[27] FAQ. *Flutter documentation* [online]. 2023 [cit. 2023-05-14]. Dostupné z: https://docs.flutter.dev/resources/faq

[28] The Good and the Bad of Flutter App Development. *Altexsoft* [online]. 2022 [cit. 2023-05-14]. Dostupné z: https://www.altexsoft.com/blog/engineering/pros-and-cons-of-flutter-app-development/

[29] Flutter architectural overview. *Flutter documentation* [online]. 2023 [cit. 2023-05-14]. Dostupné z: https://docs.flutter.dev/resources/architectural-overview

[30] What is widgets in Flutter?. *Geeksforgeeks* [online]. 2022 [cit. 2023-05-14]. Dostupné z: https://www.geeksforgeeks.org/what-is-widgets-in-flutter/

[31] StatefulWidget class. *Flutter API documentation* [online]. 2023 [cit. 2023-05-14]. Dostupné z: https://api.flutter.dev/flutter/widgets/StatefulWidget-class.html

[32] Widget class. *Flutter API documentation* [online]. 2023 [cit. 2023-05-14]. Dostupné z: https://api.flutter.dev/flutter/widgets/Widget-class.html

[33] StatelessWidget class. *Flutter API documentation* [online]. 2023 [cit. 2023-05-14]. Dostupné z: https://api.flutter.dev/flutter/widgets/StatelessWidget-class.html

[34] Security. *Flutter documentation* [online]. 2023 [cit. 2023-05-14]. Dostupné z: https://docs.flutter.dev/security

[35] Flutter_secure_storage. *Pub* [online]. 2023 [cit. 2023-05-14]. Dostupné z: https://pub.dev/packages/flutter_secure_storage

[36] Creating flavors for Flutter. *Flutter documentation* [online]. 2023 [cit. 2023-05-14]. Dostupné z: https://docs.flutter.dev/deployment/flavors

[37] Build Windows apps with .NET MAUI. *Microsoft* [online]. 2023 [cit. 2023-05-14]. Dostupné z: https://learn.microsoft.com/en-us/windows/apps/windows-dotnet-maui/

[38] Integrated Development Environment (IDE) : The Complete Guide. *Xenonstack* [online]. 2023 [cit. 2023-05-14]. Dostupné z: https://www.xenonstack.com/insights/integrated-development-environment

[39] What is Visual Studio?. *Microsoft* [online]. 2023 [cit. 2023-05-14]. Dostupné z: https://learn.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2022

[40] Meet Android Studio. *Android Developers* [online]. 2023 [cit. 2023-05-14]. Dostupné z: https://developer.android.com/studio/intro

[41] Overview. *Google Maps Platform* [online]. 2023 [cit. 2023-05-14]. Dostupné z: https://console.cloud.google.com/google/maps-apis/overview.

[42] *Flaticon* [online]. Málaga: Freepik Company S.L., © 2010-2023 [cit. 2023-05-14]. Dostupné z: https://www.flaticon.com

[43] *Pixabay* [online]. Berlin: Pixabay, © 2010-2023 [cit. 2023-05-14]. Dostupné z: https://pixabay.com/

[44] *Freepik* [online]. Málaga: Freepik Company S.L., © 2010-2023 [cit. 2023-05-14]. Dostupné z: https://www.freepik.com/

[45] *Vecteezy* [online]. Helsinki: Eezy, © 2023 [cit. 2023-05-14]. Dostupné z: https://www.vecteezy.com/

[46] *Tasty* API Documentation. Rapidapi [online]. © 2023 [cit. 2023-05-14]. Dostupné z: https://rapidapi.com/apidojo/api/tasty

ZOZNAM POUŽITÝCH SYMBOLOV A SKRATOK

JSON	JavaScript object notation
RSA	Reverts-Shamir-Adleman
URL	Uniform resource locator
IoT	Internet of Things
UX	User Experience
SDK	Software development kit
IT	Information technology
PWA	Progressive Web Apps
SEO	Search Engine Optimization
UI	User Interface
UWP	Universal Windows Platform
IL	Intermediate Language
ARM	Advanced RISC Machine
MVVM	Model–view–viewmodel
API	Application programming interface
AES	Advanced Encryption Standard
GUI	Graphical user interface
IDE	Integrated development environment

ZOZNAM OBRÁZKOV

ZOZNAM PRÍLOH

P1 CD disk

PRÍLOHA P I: CD DISK

Priložené CD obsahuje súbor priloha.zip s zdrojovým kódom aplikácie a bakalárskou práci vo formátu pdf.