Mobilní aplikace pro plánování nákupu

Bc. Marián Pjajčík

Diplomová práce
2022

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

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

Akademický rok: 2021/2022

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Marián Pjajčík**
Osobní číslo:	**A20196**
Studijní program:	**N0613A140022 Informační technologie**
Specializace:	**Softwarové inženýrství**
Forma studia:	**Kombinovaná**
Téma práce:	**Mobilní aplikace pro plánování nákupu**
Téma práce anglicky:	**Mobile Application for Purchase Planning**

Zásady pro vypracování

1. Nastudujte aktuální možnosti a technologie spojené s vývojem mobilních aplikací pro iOS.
2. Vytvořte literární rešerši řešení problému obchodního cestujícího a jeho variant.
3. Popište existující aplikace věnující se plánování nákupu a identifikujte jejich nedostatky.
4. Na základě zjištěných informací zvolte vhodnou metodu a technologii pro implementaci.
5. Pomocí zvolené technologie vytvořte interaktivní mobilní aplikaci pro plánování nákupu.
6. Vytvořte dokumentaci vzniklé aplikace.

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

Seznam doporučené literatury:

1. LACKO, Luboslav. Vývoj aplikací pro iOS. Brno: Computer Press, 2018, 479 s. ISBN 9788025149423.
2. REGO, César, Dorabela GAMBOA, Fred GLOVER a Colin OSTERMAN. Traveling salesman problem heuristics: Leading methods, implementations and latest advances. European Journal of Operational Research. 2011, 211(3), 427-441. ISSN 0377-2217. Dostupné z: doi: 10.1016/j.ejor.2010.09.010
3. Swift Documentation [online]. [cit. 2021-11-22]. Dostupné z: https://www.swift.org/documentation/
4. SwiftUI Framework [online]. [cit. 2021-11-22]. Dostupné z: https://developer.apple.com/documentation/swiftui/
5. REINELT, Gerhard. TSPLIB-A Traveling Salesman Problem Library. ORSA Journal on Computing [online]. 1991, 3(4), 376-384 [cit. 2021-11-30]. ISSN 0899-1499. Dostupné z: doi:10.1287/ijoc.3.4.376

Vedoucí diplomové práce: **Ing. Adam Viktorin, Ph.D.**
Ústav informatiky a umělé inteligence

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

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

doc. Mgr. Milan Adámek, Ph.D.
děkan

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

Ve Zlíně dne 24. ledna 2022

Jméno, příjmení: Bc. Marián Pjajčík
Název bakalářské/diplomové práce: Mobilní aplikace pro plánování nákupu
Prohlašuji, že

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

Prohlašuji,

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

Ve Zlíně, dne

Bc. Marián Pjajčík v. r.
podpis diplomanta

ABSTRAKT

Tato práce se zabývá popisem současných technologií pro vývoj na platformě iOS. Dále rozebírá problém obchodního cestujícího a jeho variant a mutací. Praktická část práce popisuje vybrané aplikace s podporou nákupního listu a jejich zhodnocení. Druhá kapitola praktické části je technická dokumentace zadané interaktivní aplikace pro nákupní list, která byla vytvořena pomocí technologie Swift a Swift UI. Grafika této aplikace byla vytvořena v aplikaci Figma.

Klíčová slova: Swift, Swift UI, React Native, PWA, TSP, iOS, mobilní technologie

ABSTRACT

This thesis describes possibilities of current development for iOS platform. It also discusses the problem of the traveling salesman and its variants and mutations. The practical part of the work describes the selected application with the support of the shopping list and their evaluation. The second chapter of the practical part is the technical documentation of the developed app – interactive shopping list application, which was created using Swift and Swift UI technologies. The design of the application was created in the Figma tool.

Keywords: Swift, Swift UI, React Native, PWA, TSP, iOS, mobile technology

Chtěl bych poděkovat mému vedoucímu Ing. Adamu Viktorinovi, Ph.D., který otevřel takto zajímavé téma na vývoj aplikace, která by ulehčila plánování nákupu a za věcné připomínky k práci.

Dále bych chtěl poděkovat Ing. Danieli Korousovi za designerské tipy při tvorbě loga.

Moje poslední poděkování patří firmě CN Group, která poskytla hardware pro vývoj ve studiu Xcode.

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

OBSAH

ÚVOD

Vývoj mobilních aplikací je v současné době jednou z nejrozšířenějších oblastí v IT. Mobilní aplikace jsou všude kolem nás a používá je většina lidí každý den. Je to jedno z nejvýdělečnějších odvětví v moderním IT světě. Technologie pro vývoj mobilních aplikací se neustále vyvíjí a posouvají dopředu.

Cílem této práce je popsat vybrané aktuální možnosti pro vývoj mobilních aplikací. V první části této práce je popsán vývoj pomocí technologie Swift a Swift UI, který cílí na zařízení od společnosti Apple – jako iPhone, iPad nebo Apple Watch. Další část se zabývá technologií React Native, ve které je možno vyvíjet aplikace jak pro iOS, tak pro Android, což jí přidává značnou flexibilitu. A poslední část popisu technologií pro vývoj mobilních aplikací se soustředí na progresivní webové aplikace, které nejsou omezeny na žádný operační systém a běží ve webovém prohlížeči.

Druhá polovina diplomové práce se zabývá problematikou obchodního cestujícího, kde popisuje samotný problém obchodního cestujícího a taktéž jeho mutace a varianty.

V praktické části je za úkol vypracovat interaktivní mobilní aplikaci pro plánování nákupu, která dostala název Šopák a poté popsat vybrané již existující aplikace na toto téma a popsat jejich nedostatky. Pro tuto práci byly vybrané aplikace – AnyList, OurGroceries, Shoppka a Listonic. Poslední část praktické kapitoly práce je technická dokumentace vyvíjené aplikace Šopák.

I. TEORETICKÁ ČÁST

1 SWIFT

Programovací jazyk Swift je univerzální programovací jazyk s open-source kódem navržený společností Apple. Podle stránky educative.io je *jazyk je ovlivněn Pythonem, takže je rychlý a intuitivní*. [1]

Swift se používá hlavně pro nativní vývoj iOS a macOS. Mnoho populárních aplikací jako LinkedIn, Lyft a Wordpress je napsáno ve Swiftu. [1]

1.1 Co je to Swift?

Jazyk Swift se vyznačuje svým více pragmatismem, univerzálností a je open-source programovací jazyk (jeho zdrojový kód lze nalézt na GitHubu), který slouží pro vývoj na platformách, které patří pod společnost Apple. Byl vytvořen v roce 2014, aby poskytl vývojářům, kteří vyvíjí na platformách iOS výkonný programovací jazyk. Podle swift.org byl jazyk navržen tak, aby byl bezpečný, rychlý a expresivní. Je to náhrada za jazyky založené C, které nebyly vyvinuty firmou Apple. Tento jazyk se stále vyvíjí a jeho komunita stále roste. [1]

1.2 Funkce Swiftu

- Generika: Generika umožňuje psát flexibilní, opakovaně použitelné funkce a typy.
- Nativní zpracování chyb: Swift podporuje vyskakování, zachycování, šíření chyb a manipulování s chybami za běhu aplikace.
- Struktury a třídy: Swift umožňuje definovat struktury nebo třídy v jediném souboru a používat je kdekoliv v kódu bez nutnosti je importovat.
- Rozšíření protokolů: Swift dává možnost definovat chování na samotných protokolech.
- Bezpečnost paměti: Automatická správa paměti a obrana proti nebezpečnému chování v kódu.
- Správa paměti: Díky automatickému počítání referencí, Swift sleduje a spravuje využití paměti aplikace – to znamená, že se Swift o správu paměti stará sám.
- Správce balíčků: Multiplatformní nástroj, který se dá použít k různým operacím s knihovnami a spustitelných souborů Swift.

- Debugging: Swift používá ladící program LLDB, který poskytuje REPL a debugger umožňující integrované ladění, zotavení po selhání, konzistentní formátování a vyhodnocování výrazů
- Zdrojová a binární kompatibilita: Nejnovější (Swift 5.1) verze jazyka Swift má binární kompatibilitu pro aplikace. Takže aplikace bude fungovat na jakékoliv verzi systému bez nutnosti ji znova kompilovat.
- N-tice: Umožňují vytvářet a sdílet hodnotová seskupení – je možné využít n-tice k vrácení více hodnot jako jedné hodnoty.
- Syntaxe uzavření: Swift má lehkou syntaxi uzavření, která umožňuje psát přehledný a jasný kód. [1, 3]

1.3 Proč používat Swift?

Tento jazyk byl navržen tak, aby se dal snadno naučit, snadno používat a aby s ním neměl větší problém žádný nový vývojář. Tento jazyk je vhodný jak pro lidi, kteří jsou stále na škole nebo zkoušejí něco nového, je jednoduchý a intuitivní a pro začátečníky bylo vytvořeno "Swift Playgrounds" pro iPady, které usnadňuje začátky s programováním ve Swift. Swift vývojáři jsou na trhu velmi žádaní. Podle Apple společnost vytvořila ve Spojených státech kolem dvou milionů pracovních míst a podle Indeed je průměrný plat pro vývojáře iOS v USA 116 804 dolarů ročně. [1, 3]

1.3.1 Výhody

- Rychlost a výkonnost: Swift využívá technologii kompilátoru LLVM a jeho standardní knihovna umožňuje intuitivní a efektivní psaní kódu
- Modernost: Swift API se snadno čtou a udržují. Díky odvozeným typům je kód čistší a méně náchylný k chybám. Moduly eliminují záhlaví a poskytují jmenné prostory.
- Lehce se učí: Swift byl navržen s ohledem na začínající programátory. K učení bylo navrženo Swift Playground pro iPad, které slouží k začátkům kódování ve Swift, má taky spoustu kurzů pro Xcode.
- Bezpečnost: Swift má řadu bezpečnostních funkcí, jako je automatická správa paměti, typy hodnot a inicializace proměnných. Ve Swiftu nemohou být objekty nikdy nulové a kompilátor Swift zastaví jakýkoliv kód, který takové objekty obsahuje. Tyto funkce pomáhají předcházet pádům běhového prostředí.

- Dynamické knihovny: Dynamické knihovny existují mimo kód a v případě potřeby se nahrávají. Tyto knihovny jsou integrovány do každé verze zařízení.
- Velká komunita: Swift má jednu z nejaktivnějších a nejbohatších komunit s otevřeným zdrojovým kódem. Existuje také mnoho zdrojů, které pomohou s naučením tohoto jazyka. [1, 3]

1.3.2 Nevýhody

- Relativně nový jazyk: Swift je stále mladý jazyk. To znamená, že některé jeho schopnosti a zdroje nejsou tak robustní jako u jiných programovacích jazyků.
- Slabá podpora napříč platformami: I když Swift podporuje všechny platformy Apple, Linux a Windows, funguje nejlépe pro nativní vývoj iOS.
- Časté aktualizace: Swift je novější jazyk a má časté aktualizace. To může ztížit hledání správných nástrojů, které pomohou s určitými úkoly. [1]

1.4 Swift vs Objective-C

Objective-C je univerzální, objektově orientovaný programovací jazyk. Byl to primární programovací jazyk používaný pro vývoj na OS X a iOS před příchodem Swift v roce 2014. Kombinuje funkce C a Smalltalku. Po vytvoření Swiftu začala Objective-C klesat používanost a popularita.

Swift není přímým nástupcem Objective-C. Oba jazyky mají různé možnosti a lze je používat společně pro vývoj mobilních aplikací. Celkově je Swift vhodnější jazyk, protože je bezpečnější, rychlejší, intuitivnější a interaktivnější. Swift má Objective-C interoperabilitu, což znamená, že Swift kód může fungovat vedle existujících Objective-C souborů. Poskytuje též přístup k Objective-C API. [1, 4]

Tabulka 1 Swift vs Objective-C [1]

Swift	Objective-C
Swift může být použit v Xcode, Swift Playgrounds, Cocoa Touch a dalších.	Objective-C je hlavně používáno v Xcode.
Swift vyžaduje alespoň iOS 7. To znamená, že iPhony a iPady vyvinuté před rokem 2014 nebudou kompatibilní.	Aplikace poběží na jakékoliv dostupné verzi iOS.

Swift je moderní člověku příjemný jazyk s jednoduchou syntaxí, což znamená, že rychlost kódování je vyšší.	Použití Objective-C obvykle vede ke snížení rychlosti kódování, protože jazyk není tak intuitivní a uživatelsky přívětivý.
Podle společnosti Apple je Swift 2.6x rychlejší než Objective-C z pohledu běhu aplikace.	Objective-C je 2.6x pomalejší než Swift z pohledu běhu aplikace a jeho zápis trvá déle kvůli jeho složitější syntaxi.
Swift má bohatou dokumentaci, která se neustále aktualizuje.	Objective-C má bohatou dokumentaci též, ale ne už tak často aktualizovanou.
Poptávka po vývojářích Swift roste.	Naopak poptávka po Objective-C vývojářích klesá.

1.5 Nastavení vývojového prostředí

Pro programování v jazyce Swift je potřeba vývojové prostředí Xcode. K tomu je potřeba účet vývojáře na webu Apple. Je potřeba se přihlásit na web Apple, a zde si stáhnout Xcode na svůj počítač. Po spuštění a přijetí podmínek je možno začít programovat ve Swift. Pro začátečníky je zde i možnost zapnout Playground. [2]

1.5.1 Online možnosti vývoje

Programátoři mohou též využít online nástroje pro kompilátory k psaní a spouštění kódu pro Swift. Na nich je tak možno vyzkoušet programovací jazyk Swift bez instalování Xcode.

Příklady

- Swifstub.com
- Runswiftlang.com
- Iswift.org
- online.swiftplayground.run [2]

1.6 Návrhové vzory

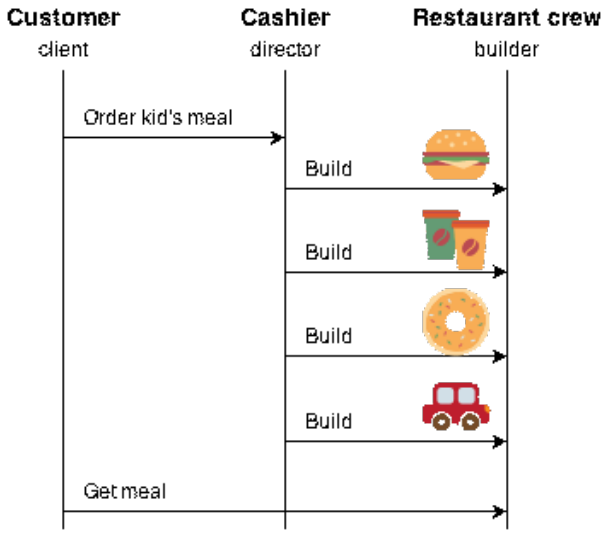
Swift je poměrně nový programovací jazyk, a tak mnoho vývojářů neví, které návrhové vzory by s ním měli používat a jak je implementovat. Umět použít relevantní návrhový vzor je předpokladem pro vytváření funkčních, vysoce kvalitních a bezpečných aplikací. [5]

Nejčastěji používané návrhové vzory ve Swift jsou

- Builder
- Adapter
- Decorator
- Facade
- Template Method [5]

1.6.1 Builder

Stejně jako mnoho abstrakcí a vzorů v programování je cílem návrhového vzoru Builder snížit potřebu udržovat proměnlivý stav – výsledkem jsou objekty, které jsou jednodušší a obecně předvídatelnější. Tím, že objektům je umožněno stát se bez stavovými, je obvykle mnohem snazší je testovat a ladit – protože jejich logika sestává pouze z čistého vstupu a výstupu. [6]

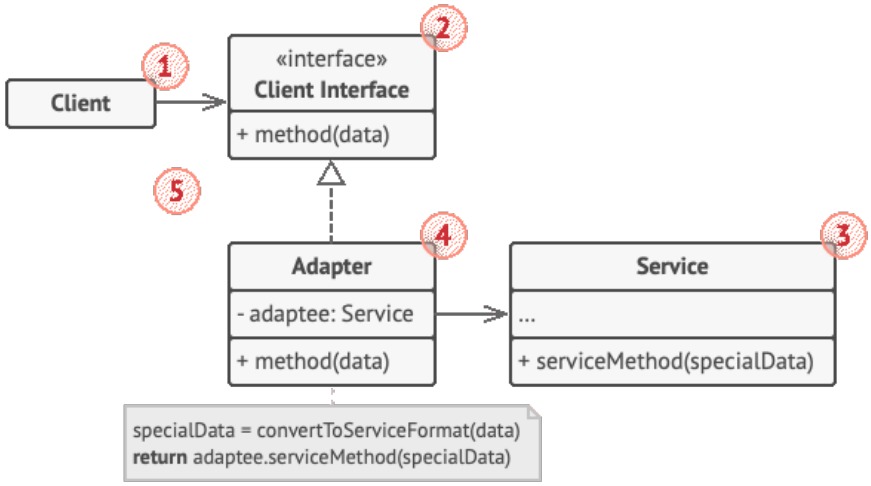


Obrázek 1 Builder [19]

Základní myšlenkou vzoru je, že proces nastavení objektu je prováděn vyhrazeným typem builder, nikoli objektem samotným. Namísto přístupu k dílčím „views“ použijeme builder k nastavení všech vlastností, které potřebujeme, prostřednictvím řady zřetězených volání metod a poté skončíme voláním „build()“ k vytvoření instance. Běžnou praxí je, že se builder vrací z každého volání metody, což usnadňuje řetězení nastavujících příkazů bez nutnosti zavádět další místní proměnné. [6]

1.6.2 Adapter

Jedná se o speciální objekt, který převádí rozhraní jednoho objektu tak, aby mu jiný objekt rozuměl. Adapter obalí jeden z objektů, aby skryl složitost konverze probíhající v zákulisí, zabalený objekt si ani neuvědomuje, že je obalen adapterem. Je možno například zabalit objekt, který používá metry a kilometry a pomocí adaptéru, převést všechna data na imperiální jednotky, jako jsou stopy a míle. [7]



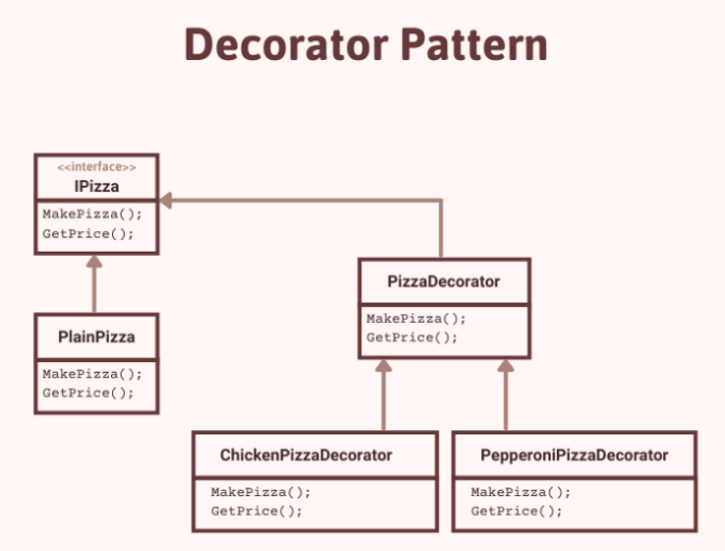
Obrázek 2 Adapter [7]

Adaptéry mohou nejen převádět data do různých formátů, ale mohou také pomáhat objektům s různými rozhraními spolupracovat. Funguje to takto – Adaptér získá rozhraní, kompatibilní s jedním ze stávajících objektů, pomocí tohoto rozhraní může existující objekt bezpečně volat metody adaptéru. Po přijetí volání adaptér předá požadavek druhému objektu, ale ve formátu a pořadí, které druhý objekt očekává. [7]

1.6.3 Decorator

Vzor decorator (také známý jako „Wrapper“) se používá k rozšíření nebo změně funkčnosti objektů za běhu jejich zabalením do objektu třídy dekoratérů. To poskytuje flexibilní alternativu k použití dědičnosti k úpravě chování. Dekoratéři nabízí flexibilní způsob, jak přidat odpovědnost k jednotlivým objektům. Na rozdíl od dědičnosti nejsou zdobené předměty omezeny svými nadřazenými třídami. Řečeno jinými slovy, klient má kontrolu nad tím, jak a kdy ozdobit komponentu. [8]

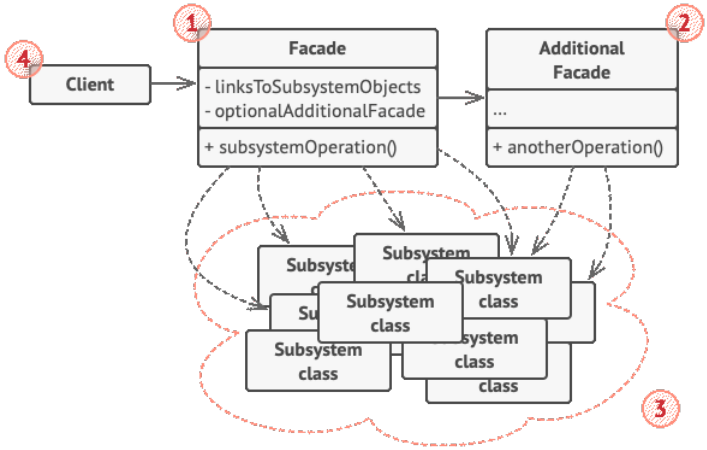
Dekoratér můžeme použít k připojování odpovědnosti k jednotlivým objektům dynamicky a transparentně, aniž bychom ovlivnili ostatní objekty, dále jej můžeme využít pro vlastnosti nebo odpovědnosti, které lze odejmout a když je podtřídění nepraktické. [8]



Obrázek 3 Decorator [18]

1.6.4 Facade

Facade je strukturální návrhový vzor, který poskytuje zjednodušené rozhraní pro knihovnu, framework nebo jakoukoli jinou komplexní sadu tříd. Je to třída, která poskytuje jednoduché rozhraní ke složitému subsystému, který obsahuje mnoho pohyblivých částí. Facade může poskytovat omezenou funkčnost ve srovnání s přímou prací se subsystémem. Zobrazuje však pouze ty funkce, na kterých klientům skutečně záleží. [9]



Obrázek 4 Facade [9]

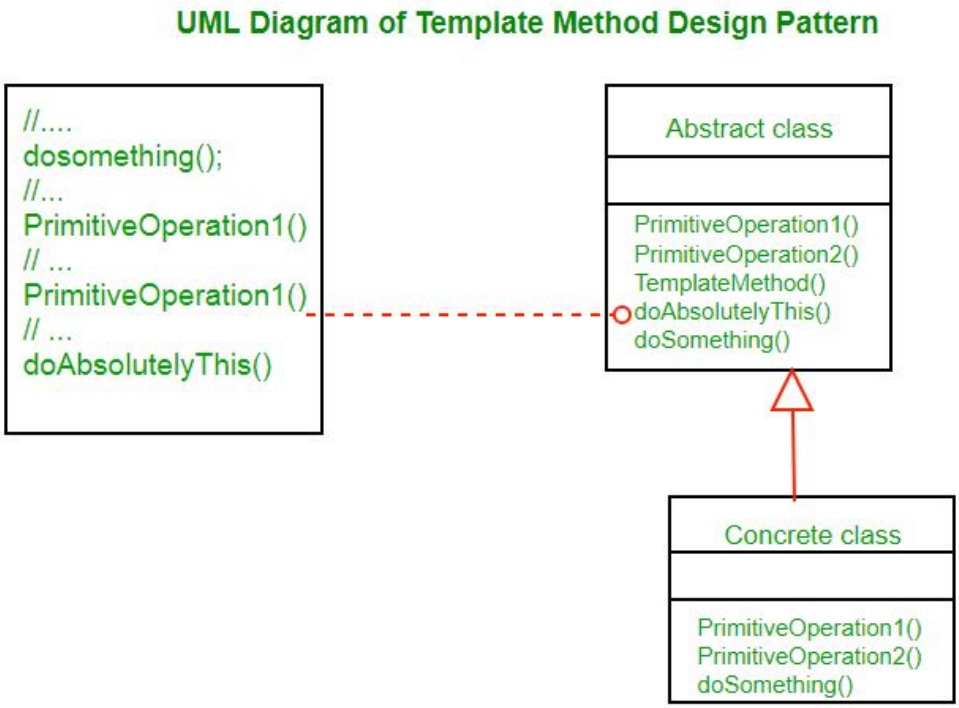
Mít facade je užitečné, když potřebujete integrovat svou aplikaci se sofistikovanou knihovnou, která má desítky funkcí, ale potřebujete z ní jen malý kousek její funkčnosti. Například aplikace, která nahrává krátká vtipná videa s kočkami na sociální média, by mohla potenciálně využívat profesionální knihovnu pro konverzi videa. Vše, co skutečně potřebuje,

je třída s jedinou metodou „encode(název souboru, formát)“. Po vytvoření takové třídy a jejím propojení s knihovnou pro převod videa bude vytvořena první facade. [9]

1.6.5 Template Method

Vzorem návrhu Template Method je definovat algoritmus jako kostru operací a ponechat detaily, které mají být implementovány podřízenými třídami. Celková struktura a sekvence algoritmu jsou zachovány nadřazenou třídou. Šablona znamená přednastavený formát jako HTML šablony, které mají pevně přednastavený formát. Podobně ve vzoru Template Method máme metodu přednastavené struktury nazývanou metoda šablony, která se skládá z kroků.

Tyto kroky mohou být abstraktní metodou, která bude implementována svými podtřídami. Tento návrhový vzor chování je jedním z nejjednodušších na pochopení a implementaci. Používá se s oblibou při vývoji frameworku. To také pomáhá vyhnout se duplicitě kódu. [10]



Obrázek 5 Template method design pattern [10]

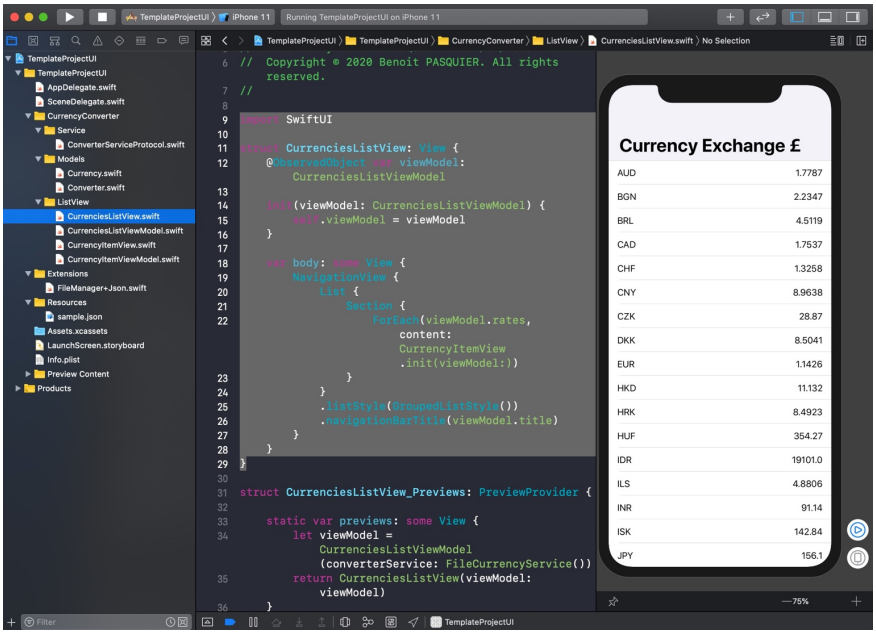
2 SWIFT UI

Podle firmy Apple *SwiftUI pomáhá vytvářet skvěle vypadající aplikace na všech platformách Apple pomocí síly Swift – a s co nejmenším množstvím kódu. SwiftUI může všem uživatelům na jakémkoli zařízení Apple přinést ještě lepší zážitky pomocí jediné sady nástrojů a rozhraní API.* [11]

2.1 Co je to SwiftUI?

SwiftUI je zcela nový designový framework, ale vypadá a působí povědomě. Koncepty, které jsou základem SwiftUI se velmi liší ve srovnání s UIKit a AppKit. Framework nabízí vše, co je potřeba k vytvoření uživatelských rozhraní, jako jsou seznamy, zásobníky, tlačítka, výběry a mnoho dalších komponent, které již byly používány v UIKit a AppKit, poskytuje též nástroje, díky kterým je možné vytvářet vlastní views, přidávat animace a integrovat gesta.

Podporuje taktéž technologie jako, přístupnost pro zdravotně postižené a adaptivní rozvržení. Tyto technologie jsou zabudovány přímo do frameworku. Základní myšlenkou SwiftUI je tedy poskytování nástrojů, které jsou potřeba k snadnému a rychlému vyvíjení uživatelských rozhraní, umožňuje je navrhovat a vyvíjet deklarativně s menším množstvím kódu. Na rozdíl od UIKit, ve kterém se používají storyboardy je SwiftUI zcela softwarový framework. Syntaxe je však velmi snadno pochopitelná a projekt zde lze rychle zobrazit pomocí automatického náhledu. [12, 16]



Obrázek 6 Swift UI [17]

SwiftUI bylo vytvořeno pro použití napříč platformami k vytváření aplikací s menším množstvím kódu než UIKit, ale se stejnou složitostí. Je důležité mít na paměti, že UIKit a SwiftUI se vzájemně nevylučují. Je možné použít kód UIKit v zobrazení SwiftUI a naopak. [15, 16]

2.1.1 Deklarativní syntaxe

SwiftUI používá deklarativní syntaxi, takže je možno jednoduše uvést, co má uživatelské rozhraní dělat. Je možno například napsat, že má vytvořit seznam položek sestávající z textových polí, a poté popsat zarovnání, písmo a barvu každého pole. Kód je jednodušší a snáze čitelný, což šetří čas a údržbu. Tento deklarativní styl se dokonce vztahuje na komplexní koncepty, jako jsou animace. Je snadné přidat animaci k téměř jakémukoliv ovládacímu prvku a je možné si vybrat kolekci efektů připravených k použití s pouhými několika řádky kódu. Za běhu systém zpracovává všechny kroky potřebné k vytvoření plynulého pohybu, a dokonce se vypořádá s přerušením, aby byla aplikace stabilní. [11]

2.1.2 Nástroje pro design

Xcode obsahuje intuitivní nástroje pro design, díky kterým je možné vytvářet rozhraní pomocí přetahování. Při práci na návrhovém plátně je vše, co je právě upravováno, zcela synchronizováno s kódem v přilehlém editoru. Kód je okamžitě viditelný jako náhled při psaní a jakákoli změna, která je v náhledu provedena se okamžitě objeví v kódu. Xcode změny okamžitě překompiluje a vloží je do běžící verze aplikace. Design lze lehce vytvořit pomocí „drag and dropu“, dynamického nahrazování prvků a živého náhledu. [11]

2.1.3 UIKit

Framework UIKit poskytuje požadovanou infrastrukturu pro aplikace na iOS nebo tvOS. Poskytuje architekturu oken a zobrazení pro implementaci rozhraní, infrastrukturu pro zpracování událostí pro poskytování Multi-Touch a dalších typů vstupů do aplikace a smyčku hlavního běhu potřebnou ke správě interakcí mezi uživatelem, systémem a aplikací. Mezi další funkce nabízené frameworkem patří – podpora animací, podpora dokumentů, podpora kreslení a tisku, informace o aktuálním zařízení, správa a zobrazení textu, podpora vyhledávání, podpora usnadnění, podpora rozšíření aplikací a správa zdrojů. [13]

2.1.4 AppKit

AppKit obsahuje všechny objekty, které jsou potřeba k implementaci uživatelského rozhraní pro aplikace pro macOS – okna, panely, tlačítka, nabídky, posuvníky a textová pole – a zpracovává všechny podrobnosti za uživatele, protože efektivně kreslí na obrazovku, komunikuje s hardwarem zařízení a vyrovnávací pamětí obrazovky. Maže též oblasti obrazovky před vykreslením a ořezává pohledy. [14]

2.2 Výhody a nevýhody SwiftUI

SwiftUI je zcela nový framework, který umožňuje navrhovat a vyvíjet uživatelská rozhraní deklarativně a s menším množstvím kódu. Zde je popis dalších výhod a nevýhod tohoto frameworku. [16]

2.2.1 Výhody

- Lze jej smíchat s UIKit pomocí UIHostingController
- Umožňuje snadno spravovat témata. Vývojáři mohou do svých aplikací snadno přidat tmavý režim, nastavit jej jako výchozí motiv a uživatelé mohou tmavý režim snadno povolit
- SwiftUI poskytuje mechanismy pro nadšence reaktivního programování s BindableObject, ObjectBinding a celým frameworkem Combine
- Nabízí živý náhled, který umožňuje vidět výsledky práce v reálném čase
- Nepotřebuje interface builder. Nahradil jej Canvas (živý náhled) a když jsou vytvořeny prvky vizuální prezentace, automaticky se objeví v kódu
- Aplikace už nebude padat, pokud se zapomenou aktualizovat přidružení @IBOutlet s proměnnou
- Neexistují žádné AutoLayout nebo související problémy. Místo toho se používají věci jako HStack, VStack, ZStack, skupiny, seznamy a další. Na rozdíl od AutoLayout vytváří SwiftUI vždy platné rozvržení. Neexistuje zde nic takového jako nejednoznačné nebo neuspokojivé rozvržení
- SwiftUI nahrazuje storyboardy kódem, což usnadňuje vytváření znovu použitelného pohledu a zabraňuje konfliktům souvisejícím se současným používáním jednoho projektu vývojovým týmem [16]

2.2.2 Nevýhody

- Podporuje pouze iOS 13 a Xcode 11. Což ale u uživatelů Apple není velký problém, protože většina uživatelů Apple udržují svá zařízení aktualizovaná
- Stále mladý framework, takže na Stack Overflow není mnoho diskusí. To znamená, je těžší najít pomoc při řešení složitějších problémů
- Neumožňuje zkoumat hierarchii zobrazení v náhledech Xcode

3 REACT NATIVE

React Native je JavaScriptový framework pro psaní skutečných, nativně vykreslovaných mobilních aplikací pro iOS a Android. Je založen na React, Javascriptové knihovně Facebooku pro vytváření uživatelských rozhraní, ale místo cílení na prohlížeč cílí na mobilní platformy. Jinými slovy: weboví vývojáři nyní mohou psát mobilní aplikace, které vypadají skutečně „nativní“. Navíc, většina kódu lze sdílet mezi platformami, čímž React Native usnadňuje současný vývoj pro Android i iOS.

React Native obsahuje mnoho „ready-to-go“ šablon, velkou komunitu, má skvělý výkon a nízké náklady na výrobu a údržbu. Tento framework je vhodné použít, pokud je potřeba, aby aplikace fungovaly na všech zařízeních, operačních systémech Android a iOS s jednou kódovou základnou.

Největší rozdíl mezi nativními aplikacemi a aplikacemi pro různé platformy je ten, že nativní aplikace jsou vytvořeny tak, aby fungovaly na konkrétním operačním systému. To znamená že mohou využívat všechny prostředky daného operačního systému, ale nemůžou fungovat nikde jinde. Přestože aplikace pro více platforem byly vytvořeny tak, aby fungovaly všude, stále jim chybí flexibilita a výkon v konkrétním operačním systému, jaký mají nativní aplikace. [20, 21]

3.1 Výhody vs Nevýhody

Donedávna bylo vytvoření aplikace velmi časově a finančně náročné. Podniky proto začaly hledat alternativy a jako jedno z možných řešení pro vývoj napříč platformami vznikl React Native. Zde jsou výhody a nevýhody tohoto frameworku. [21]

3.1.1 Výhody z business hlediska

- **Rychlost vývoje a náklady** – lze zde recyklovat a znovu používat komponenty vyvinuté dříve, taktéž jde sdílet kódovou základnu
- **Funguje všude** – aplikace lze vytvářet jak pro iOS, tak pro Android nebo Windows
- **Lze v něm vytvořit výborné UX**
- **Rychlé uvedení na trh** – aplikaci lze dostat na trh mnohem rychleji pro otestování MVP a lze tak získat zpětnou vazbu od uživatelů
- **Pomoc na vyžádání** – komunita React Native je obrovská. Mnoho problémů, se kterými se dá setkat během vývoje, již může být někde vyřešeno

- **Náklady na údržbu** – jedna kódová základna znamená menší náklady na údržbu
- **React Native aplikace jsou viditelné** – je snadné dostat aplikace na AppStore a Play Store na rozdíl od PWA. [21]

3.1.2 Výhody z vývojového hlediska

- **Snadná práce** – Smysluplné chybové zprávy, úspora času a robustní nástroje
- **Náhled změn** – Pro zobrazení změn není třeba aplikaci znovu sestavovat. Šetří se tím spoustu času a změny se dají dělat rychle a efektivně, stačí pouze stisknout tlačítko command a R.
- **Snazší ladění** – jako výchozí používá Flipper.
- **Znovupoužitelnost kódu** – vývojáři mohou snad integrovat 90 % nativního frameworku a použít kód pro jakoukoliv platformu. Tato funkce nejen šetří čas, ale také pomáhá snížit náklady na vytvoření dvou aplikací. Lze zde též použít kód webové aplikace (napsaný v Reactu).
- **Předem vyvinuté komponenty** – pro urychlení práce je k dispozici řada open source knihoven.
- **Funkce živého načítání** – pomáhá kompilovat a číst soubor od bodu, kde vývojář provedl změny. Poté je simulátoru nabídnut nový soubor, který automaticky načte soubor od začátku.
- **Kompatibilní s pluginy třetích stran** – nevyžaduje velkou paměť k zpracování.
- **Jedna kódová základna pro všechny** – možnost jedné kódové základy pro Android, iOS a webový prohlížeč.
- **Velká komunita vývojářů, Inteligentní nástroje pro ladění a mechanismy hlášení chyb**, **Rychlé iterační cykly** [21]

3.1.3 Nevýhody

- **Výkon je stále nižší než nativní** – React Native nedokáže využít všechna požehnání a potenciál konkrétní platformy. Na druhou stranu nativní aplikace nedokážou skutečně maximalizovat funkce a ve výsledku poskytnout maximální uživatelský zážitek. Re-architektura však vynakládá velké úsilí na to, aby byl React Native výkonnější.

- **Neefektivní design** – pokud projekt obsahuje komplikované návrhy nebo pokročilé interakce za klíčovou součást, měl by se spíš použít nativní vývoj
- **Problémy s aktualizací** – je těžké udržovat aplikaci aktualizovanou na nejnovější verzi React Native. Aktualizace verzí React Native je ve většině případů komplikovaný proces. [21]

3.2 React Native vs Swift/Swift UI

3.2.1 Uživatelské Rozhraní

React Native využívá nativní komponenty a API a umožňuje vytvářet nativní aplikace, které vypadají skoro jako nativní. Swift byl postaven výhradně pro nativní vývoj iOS, který zaručuje špičkové prvky uživatelského rozhraní pro iOS zařízení. [22]

3.2.2 Stabilita

Zatímco React Native může poskytnout slušné nativní uživatelské rozhraní, nemůže dosáhnout stejné úrovně stability ve srovnání se Swiftem. Pomocí nativního vývojového jazyka lze lépe využít funkce zařízení a vytvořit stabilnější a na funkce bohatší aplikaci. [22]

3.2.3 Výkon

I zde se Swift vyznačuje lepšími vlastnostmi, protože nativní vývoj téměř vždy poskytuje lepší výkon než vývoj napříč platformami. Zejména pokud jde o aplikaci s dynamickými a grafickými efekty. [22]

3.2.4 Rychlost kódování

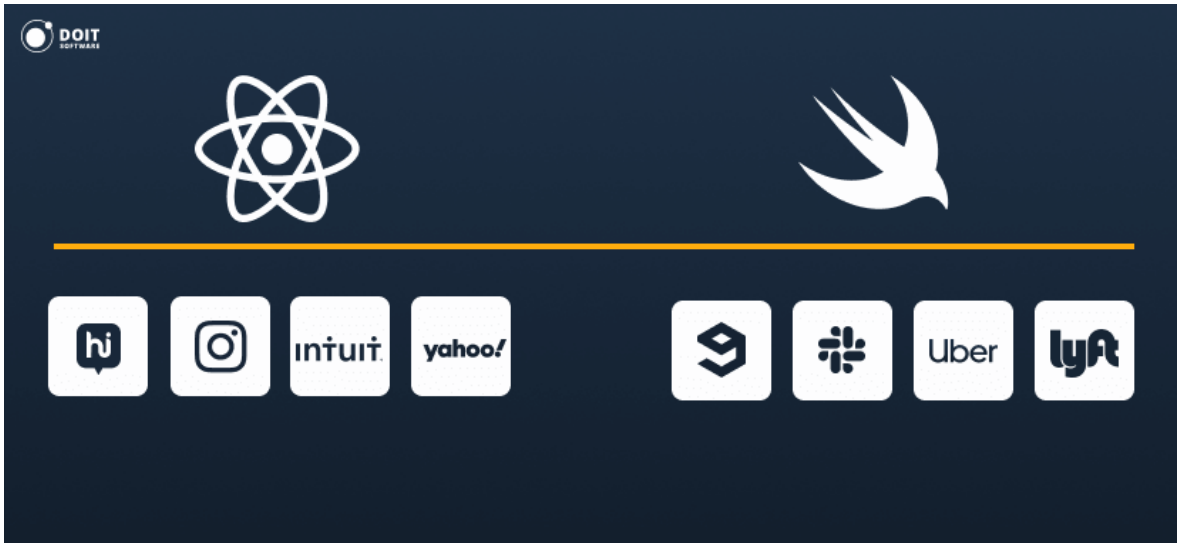
Zde to závisí na cílech vývojáře. Pokud je aplikace vyvíjena pouze pro iOS, pak je Swift určitě lepší volbou. Pokud je cílem vývoj i Android, tak React Native je lepší volba. [22]

3.2.5 Komunita

Ačkoli je těžké je srovnávat, protože React Native je framework a Swift je jazyk, stále lze usoudit že React Native má mnohem větší komunitu. Swift se ale ukázal jako velmi oblíbený a užitečný mezi komunitou vývojářů mobilních aplikací a postupem času si získává stále větší pozornost. [22]

3.2.6 Dokumentace

Vzhledem k tomu, že oba nástroje byly vyvinuty uznávanými technologickými giganty – Facebookem (Meta) a Applem, mají React Native i Swift vynikající dokumentaci. [22]



Obrázek 7 Populární aplikace React Native a Swift [22]

3.3 Návrhové vzory pro JavaScript

React Native používá jazyk JavaScript, proto zde budou popsány nejoblíbenější návrhové vzory pro jazyk JavaScript.

Návrhové vzory JavaScriptu pomáhají vývojářům psát organizované, přehledné a dobře strukturované kódy. Přestože návrhové vzory, když jsou použity, lze snadno znovu použít, nikdy nemohou doplnit vývojáře, spíš je podporují tím, že předcházejí drobným problémům při vývoji webových/mobilních aplikací, tím, že poskytují zobecněná řešení, která nejsou vázána na konkrétní problém. [23]

Mezi nejoblíbenější návrhové vzory pro JS patří:

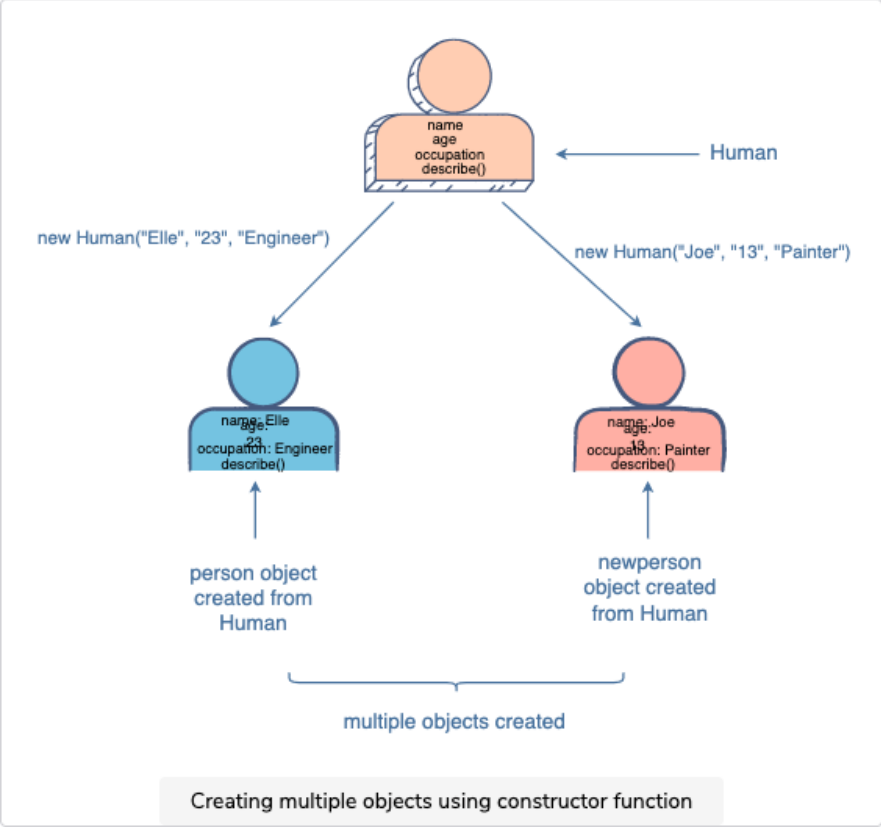
- Návrhový vzor konstruktoru
- Prototypový vzor
- Návrhový vzor modulu
- Návrhový vzor Singleton
- Vzor továrna [23]

3.3.1 Návrhový vzor konstruktoru

Jedná se o speciální metodu, která se používá k inicializaci nově vytvořených objektů po přidělení paměti. V JavaScriptu se nachází konstruktory – jedná se o speciální funkce, které inicializují objekty se specifickými vlastnostmi a metodami. Vzor konstruktoru, jak název definuje, je vzor založený na třídě, která používá konstruktory přítomné ve třídě k vytváření specifických typů objektů. [23, 24]

Návrhový vzor konstruktor se používá, když je potřeba více instancí stejné šablony, protože instance mohou sdílet metody, ale přesto se mohou lišit. Využívá se třeba v knihovnách a pluginech. [24]

Existuje mnoho způsobů, jak vytvářet objekty v JavaScriptu, například pomocí zápisu {} nebo pomocí Object.create. V JavaScriptu je však použití vzoru konstruktor velmi oblíbené, protože dokáže vytvořit více objektů určitého druhu



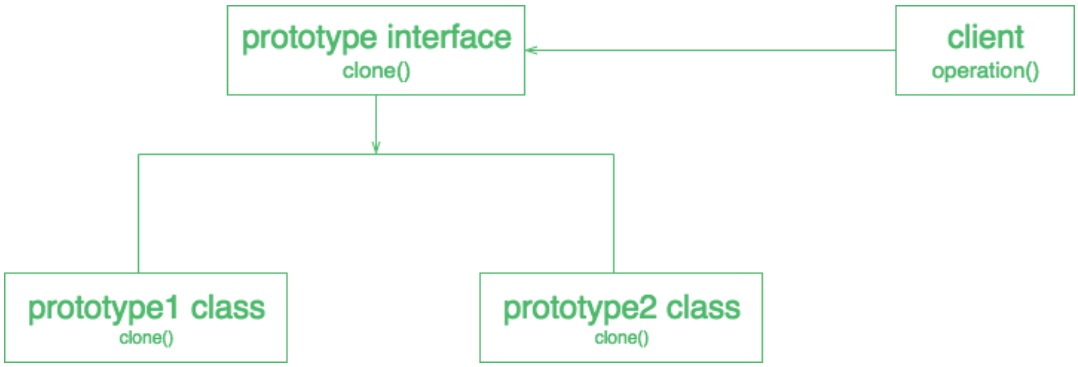
Obrázek 8 Vytvoření více objektů s použitím konstruktoru [24]

3.3.2 Prototypový vzor

Vzor prototypu je založen na prototypické dědičnosti, kdy jsou objekty vytvořené tak, aby fungovaly jako prototypy pro jiné objekty. Ve skutečnosti prototypy fungují jako plán pro

každý vytvořený konstruktor objektu. Používá se především pro vytváření objektů v situacích náročných na výkon. [23, 25]

Vytvořené objekty jsou klony původního objektu, které jsou předávány. Jedním případem použití vzoru prototypu je provedení rozsáhlé databázové operace k vytvoření objektu používaného pro jiné části aplikace. Pokud jiný proces potřebuje použít tento objekt, místo toho, aby musel provádět tuto podstatnou databázovou operaci, bylo by výhodné klonovat již dříve vytvořený objekt. [25]

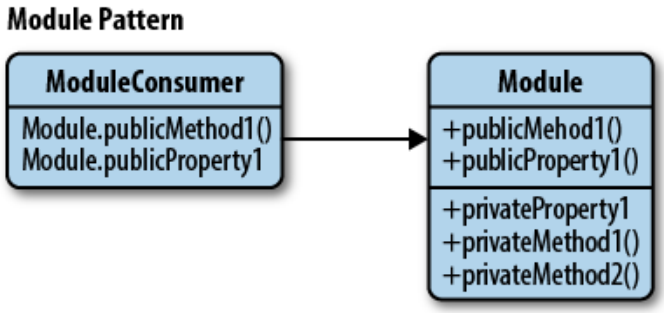


Obrázek 9 Prototypový vzor UML [26]

3.3.3 Návrhový vzor modulu

V návrhovém vzoru modulu došlo ke zlepšení oproti vzoru prototypu. Různé typy modifikátorů (soukromé i veřejné) jsou nastaveny ve vzoru modulu a podobné funkce nebo vlastnosti mohou být vytvářeny bez konfliktů. Existuje zde taktéž flexibilita veřejného přejmenování funkcí. Odstrašující je na tom neschopnost přepsat vytvořené funkce z vnějšího prostředí. Jedná se o běžně používaný návrhový vzor, který se používá k zabalení sady proměnných a funkcí do jednoho rozsahu. Používá se k definování objektů a specifikaci proměnných a funkcí, ke kterým lze přistupovat mimo rozsah funkce. Určité vlastnosti a funkce vystavujeme jako veřejné a můžeme také omezit rozsah vlastností a funkcí v rámci samotného objektu a učinit je soukromými. To znamená, že k těmto proměnným nelze přistupovat mimo rozsah funkce. Pomocí toho vzoru můžeme dosáhnout dat skrývajících abstrakci. [23, 27]

Vzor modulu umožňuje lepší udržovatelnost, protože veškerý související kód lze zapouzdřit do jediného logického bloku. Tyto logicky nezávislé bloky se relativně snadněji aktualizují. Dále umožňuje opětovné použití, jednu jednotku kódu lze znovu použít v celé aplikaci. Funkcionalitu uzavřenou jako modul lze znovu použít a nemusíme definovat stejné funkce ve více bodech. [27]

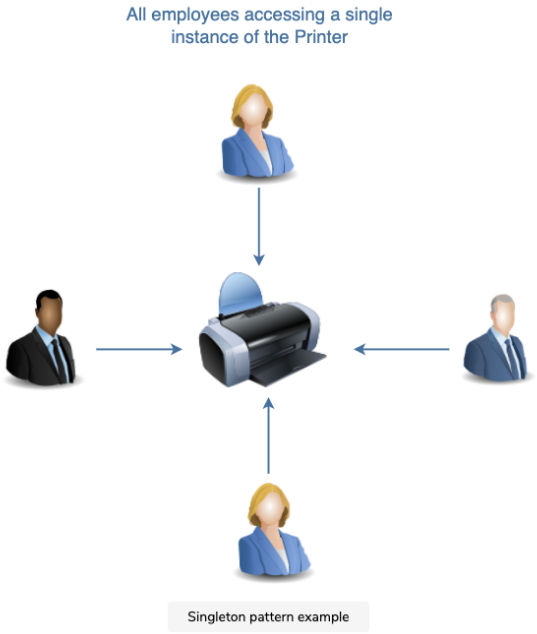


Obrázek 10 Návrhový vzor modulu [28]

3.3.4 Návrhový vzor Singleton

Tento vzor je nezbytný ve scénáři, kde je třeba vytvořit pouze jednu instanci, například připojení k databázi. Singleton umožňuje třídě vytvořit instanci třídy při prvním vytvoření instance, při dalším pokusu je však vrácena existující instance třídy. Vytvořit instanci je možné pouze při zavřeném připojení nebo po ujištění, že před otevřením nové instance byla stará uzavřena. Tento vzor je také označován jako přísný vzor, jednou nevýhodou tohoto vzoru je jeho skličující zkušenost s testováním kvůli jeho skrytým objektům závislostí, které nelze snadno vybrat pro testování. [23, 29]

Příkladem ze skutečného života je tiskárna, kterou chtějí zaměstnanci kanceláře používat. Bude to sdílený zdroj mezi všemi zaměstnanci. Je tedy vyžadována jedna instance tiskárny, kterou může sdílet každý, místo aby měla novou instanci pro každého zaměstnance, který chce něco vytisknout. [29]

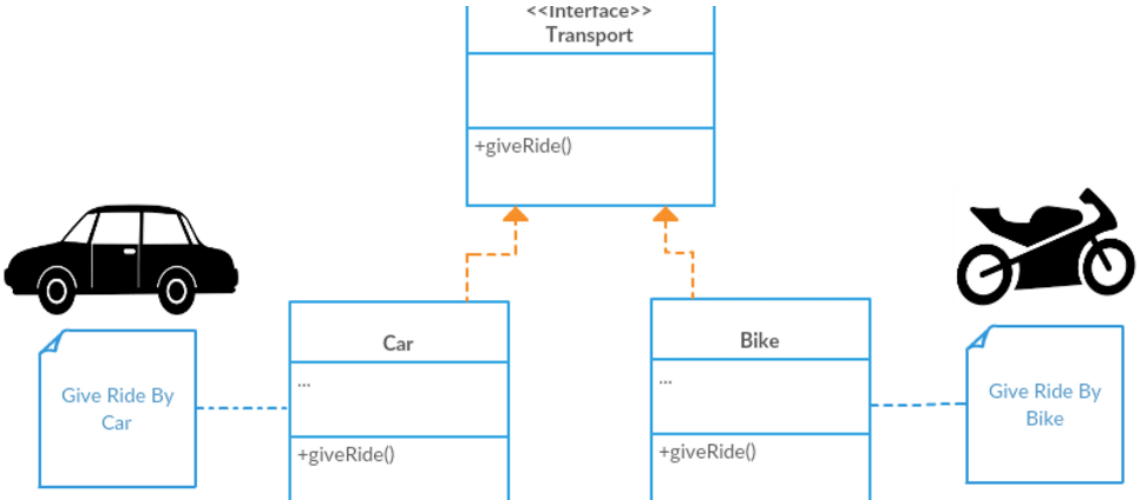


Obrázek 11 Singleton příklad [29]

Vzor Singleton se používá většinou v případech, kdy je potřeba, aby jeden objekt koordinoval akce v rámci systému. Najdeme ho například ve službách, databázích nebo v konfiguracích. [29]

3.3.5 Vzor továrna

Zabývá se tvorbou objektu bez potřeby konstruktoru. Poskytuje obecné rozhraní pro vytváření objektů, kde může být specifikován typ továrních objektů, které mají být vytvořeny. Objekt je tedy pouze specifikován a továrna jej vytvoří a vrátí k použití. Je moudré použít tovární vzor, když nastavení komponenty objektu má vysokou úroveň složitosti a když je potřeba snadno vytvářet různé instance objektů v závislosti na prostředí, ve kterém se objekt nachází. Může se též použít při práci s mnoha malými objekty sdílející stejné vlastnosti a při skládání objektů, které vyžadují oddělní. Díky tomuto vzoru je kód lépe čitelný, snižuje redundanci a vytváří abstrakci. Řídí se metodikou DRY. Jak název napovídá, instance objektů jsou vytvářeny pomocí továrny, aby vytvořily pro aplikaci požadovaný objekt. [23, 30]



Obrázek 12 Vzor továrna [31]

4 PWA – PROGRESIVNÍ WEBOVÁ APLIKACE

Web je rozsáhlá platforma. Díky kombinaci všudypřítomnosti napříč zařízeními a operačními systémy, modelu zabezpečení zaměřeného na uživatele a skutečnosti, že ani jeho specifikace, ani implementace nejsou řízeny jedinou společností, je web jedinečnou platformou pro vývoj softwaru. V kombinaci s jeho vlastní propojitelností je možné jej prohledávat a sdílet cokoli s kýmkoli a kdekoli. Webové aplikace se mohou dostat ke komukoli, kdekoli a na jakémkoli zařízení s jedinou kódovou základnou. [32]

Alex Russell poznamenal že „Lidé nepoužívají web, tak jak se o něj opírají, spoléhají na něj a stávají se na něm závislí na počítači“. Ve své přednášce poznamenal, že lidé používají web asi 4 % času, kdy používají telefony a tyto procenta ubývají. Po zbytek času uživatelé obvykle komunikují s mobilními aplikacemi, nikoli s prohlížečem. [33]

Mnoho z toho je způsobeno skutečností, že společnosti, které vlastní platformy (zejména Google a Apple), se primárně zaměřují na vývoj nativních aplikací. Toto zaměření na vývoj mobilních aplikací tlačí trh k tomu, aby akceptoval důležitost mobilní aplikace, a přiměje uživatele opustit web za nativním prostředím. Vývoj mobilních aplikací uzamkne zkušenost na zařízení a operační systémy, což zvyšuje náklady na vývoj. Uživatelé zase očekávají mobilní aplikace, a to i pro jednoduché úkoly, jako je prohlížení tras autobusů nebo vyplňovaní formuláře. [33]

Progresivní webové aplikace jsou sestaveny a vylepšeny pomocí moderních rozhraní API, aby poskytovaly vylepšené možnosti, spolehlivost a instalovatelnost a zároveň byly dostupné komukoli, kdekoli a na jakémkoli zařízení pomocí jediné kódové základy. PWA jsou vytvořeny se standardními webovými technologiemi – HTML, CSS, JavaScript a moderními rozhraními API prohlížeče – aby poskytovaly lepší zážitek při jejich používaní na podporovaných platformách. S využitím nejnovějších funkcí prohlížeče mohou weboví vývojáři vytvořit stejnou zkušenost, jakou očekávají uživatelé od nativních aplikací. Mezi tyto funkce patří – přístup ke kameře, Bluetooth a síťové informace – dokonce i rozšířená/virtuální realita a platby. Prohlížeče pracují na podpoře těchto funkcí už roky a některé společnosti (jako Twitter) vytvářejí PWA, aby poskytovaly vylepšené prostředí pro své platformy. [32,33]

4.1 Jak vytvořit PWA?

4.1.1 Poskytování přes HTTPS

SSL přidává na web další vrstvu zabezpečení a pomáhá uživatelům cítit se bezpečně při používaní webu. U PWA je HTTPS zásadní pro použití servisních pracovníků a umožnění instalace na domovskou obrazovku. Certifikát SSL je možno koupit u registrátora domény s malými náklady a poté jej nakonfigurovat prostřednictvím hostingové služby. [34]

4.1.2 Vytvoření prostředí aplikace

Prostředí je první věc, která se načte – první věc, kterou uživatel uvidí. Mělo by existovat celé v indexovém HTML dokumentu s vloženým CSS, aby se zajistilo, že se zobrazí co nejrychleji a uživatel nebude muset čekat na bílé obrazovce déle, než je nutné. Aplikační obal tvoří součást vzoru progresivního vylepšovaní. Aplikace by měla uživateli poskytnout obsah co nejdříve a poté jej postupně vylepšovat, jak se načítá více dat. [34]

4.1.3 Registrace servisního pracovníka

Aby bylo možno využívat celé spektrum PWA (notifikace, ukládání do mezipaměti, výzvy k instalaci), je nutné mít servisního pracovníka (Service Worker). Servisní pracovníci jsou lehce nastavitelní. Nejprve je potřeba zkontrolovat, zda prohlížeč uživatele podporuje servisní pracovníky. Pokud ano, je třeba zaregistrovat pracovníka v souboru service-worker.js. Na začátku může být soubor prázdný. [34]

4.1.4 Přidání push notifikací

Servisní pracovníci umožňují uživatelům přijímat oznámení push prostřednictvím webového rozhraní Push API. Je možné se k němu dostat ze souboru servisního pracovníka na self.registration.pushManager. Odesílání push notifikací závisí do značné míry na backendovém nastavení aplikace. [34]

4.1.5 Přidání manifestu webové aplikace

Aby bylo možné aplikaci nainstalovat, musí být v kořenovém adresáři aplikace zahrnut soubor manifest.json. Slouží k popisu aplikace, podobný tomu, co je třeba na AppStore. Obsahuje ikony, úvodní obrazovku, název a popis.

Existuje také určitá konfigurace toho, jak se aplikace zobrazí, když je spuštěna z domovské obrazovky uživatele – například zobrazení řádku adresy v prohlížeči, barvu stavového řádku

a tak dále. Správný manifest.json by měl zahrnovat celé spektrum velikostí ikon pro různá zařízení. [34]

4.1.6 Konfigurace výzvy k instalaci

Když uživatel navštíví PWA se servisním pracovníkem a manifestem, Chrome jej automaticky vyzve, aby si jej nainstaloval na domovskou obrazovku, a to za následujících podmínek: uživatel musí web navštívit dvakrát, mezi návštěvami musí být pět minut.

Myšlenka je zde počkat, až uživatel projeví zájem o aplikaci a poté jej požádat, aby z ní udělal součást svého zařízení. Mohou však nastat případy, kdy je potřeba zobrazit výzvu k instalaci v různých situacích, například poté, co uživatel provede určitou užitečnou akci. Za tímto účelem zachytíme událost beforeinstallprompt a uložíme ji na později a poté nasadíme výzvu, když to považujeme za vhodné. [34]

4.1.7 Analýza výkonu aplikace

Výkon je hlavní myšlenkou PWA. Aplikace by měla být rychlá pro uživatele ve všech podmínkách sítě. Ukládání do mezipaměti a offline funkce pomáhají, ale aplikace by měla být rychlá, i když prohlížeč uživatele nepodporuje servisní pracovníky. To je definice progresivního vylepšování. K tomu je užitečnou sadou metrik systém RAIL. RAIL je to, co Google nazývá „modelem výkonu zaměřeným na uživatele“ – soubor pokynů pro měření výkonu aplikace.

4.1.8 Audit aplikace pomocí Lighthouse

Google je největší firma prosazující progresivní webové aplikace jako budoucnost webu. Jako takový poskytuje užitečný nástroj pro vedení vývoje PWA. Lighthouse je součástí Chrome DevTools na kartě Audits. Lighthouse spouští aplikaci za různých podmínek a měří její odezvu podle pokynů PWA. Poté ohodnotí aplikaci na škále 0-100. [34]

4.2 Výhody a nevýhody PWA

Výhody u PWA výrazně převyšují nad nevýhodami. Navzdory, že PWA existují relativně krátkou dobu, již dokázaly zavést novou filozofii tvorby webových stránek a žádná společnost, která chce být v mobilní éře relevantní, si nemůže dovolit je ignorovat. [35]

4.2.1 Výhody

Offline mód – PWA mohou být ve webovém prohlížeči uloženy do mezipaměti a používány, i když jsou offline. Využití najdeme třeba u firem s katalogy produktů, offline mód umožňuje jejich zákazníkům procházet produkty, i když nejsou připojeni k internetu, což zvyšuje míru zapojení uživatelů a potenciálně vede k vyšším výnosům. [35]

Zlepšený výkon – „53 % uživatelů opustí web, pokud jeho načítání trvá déle než 3 sekundy. A po načtení uživatelé očekávají, že bude rychlý – žádné trhané rolování nebo rozhraní s pomalou odezvou.“ Uvádí Google na svém webu. Protože PWA využívají servisní pracovníky, kteří běží odděleně od hlavního vlákna prohlížeče a proaktivně řídí ukládání prostředků do mezipaměti, mohou poskytovat mnohem lepší výkon než tradiční webové aplikace. [35]

Nevyžadují instalaci ani ruční aktualizace - Pokud uživatel chce používat progresivní webovou aplikaci Twitteru, není třeba k instalaci navštěvovat Obchod Play nebo App Store. Uživatelé mohou jednoduše navštívit mobile.twitter.com a bez prodlení se přihlásit. Když Twitter aktualizuje své PWA, uživatelé jej nemusí instalovat ručně – všechny nové funkce a opravy chyb jsou k dispozici bez nutnosti jakékoli ruční instalace. [35]

Funkce specifické pro platformu – PWA mohou využívat mnoho funkcí specifických pro platformu. PWA mohou například existovat na domovské obrazovce uživatele a doručovat webová oznámení push, která se zobrazují stejně jako běžná oznámení push. Mohou běžet na celé obrazovce, změnit orientaci displeje, začít s vlastní úvodní obrazovkou, získat přístup k datům o poloze a mnoho dalšího. [35]

Spotřebují málo dat – PWA jsou mnohem menší než mobilní aplikace a vyžadují mnohem menší šířku pásma než tradiční webové aplikace, protože mohou mnohem lépe využívat cachování. Například PWA společnosti Tinder je jen 2,8 MB velká, zatímco její aplikace pro Android má velikost 30 MB. [35]

Nezávislé na obchodu s aplikacemi – Toto je výhoda pro menší podniky a nezávislé vývojáře aplikací, kteří nechtějí platit poplatek Applu ve výši 99 USD nebo doživotní poplatek Googlu ve výši 25 USD jen za zveřejnění své aplikace. Nezávislost na obchodě s aplikacemi také umožňuje vývojářům aplikací vytvářet libovolnou aplikaci, aniž by byli spoutáni zásadami a omezeními obchodu s aplikacemi. [35]

4.2.2 Nevýhody

Kompatibilita s iOS – od iOS 11.3 je možné spouštět PWA na zařízeních Apple, ale nejsou kompatibilní se staršími zařízeními. Dále Apple neumožňuje PWA přístup k mnoha důležitým funkcím, včetně Touch ID, Face ID, ARKit, Bluetooth, Beacons, senzoru výškoměru a informace o baterii. [35]

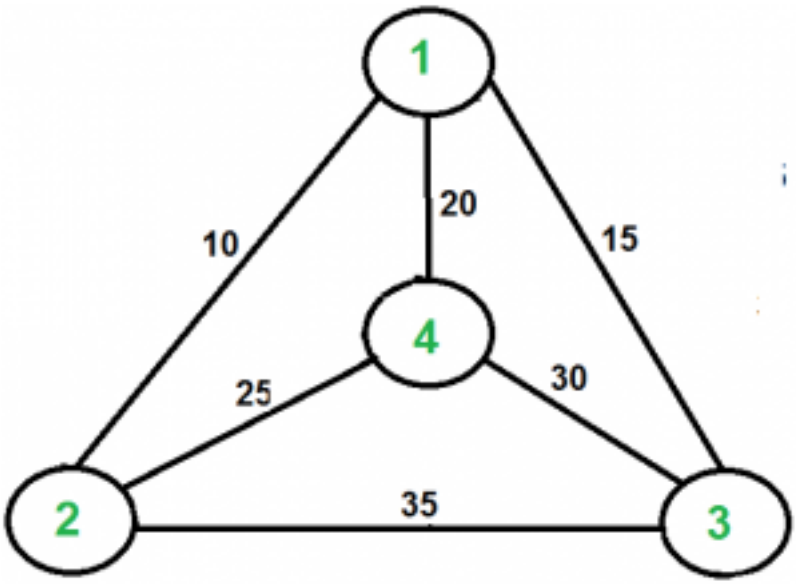
Problémy se staršími zařízeními – PWA existují jen pár let, takže starší zařízení se zastaralými webovými prohlížeči je příliš nepodporují. Tento problém se pravděpodobně v budoucnu vyřeší, ale pro některé společnosti může být zdrojem stížností zákazníků. [35]

PWA nedokážou všechno – přestože jsou PWA ve srovnání s tradičními webovými aplikacemi schopné, nemohou dělat vše, co mobilní aplikace. Protože jsou napsány v JavaScriptu, nejsou tak energeticky úsporné jako aplikace napsané v nativních jazycích, jako je Kotlin nebo Swift. Jejich výkon není tak dobrý jako výkon nativních aplikací, což má hodně společného s tím, že JavaScript je jednovláknový programovací jazyk. [35]

5 OBCHODNÍ CESTUJÍCÍ A JEHO VARIANTY

Tento problém byl formulován v roce 1930. Je však jedním z nejvíce studovaných kombinatorických optimalizačních NP-těžkých problémů i dnes. Problém obchodního cestujícího je algoritmický problém, jehož úkolem je najít nejkratší cestu mezi sadou bodů a místy, které je třeba navštívit. V tomto problému jsou body města, která může prodejce navštívit. Cílem prodejce je udržet jak cestovní náklady, tak ujetou vzdálenost co nejnižší. [36, 43, 50]

Problém cestujícího obchodníka má minimalizovat vzdálenost mezi různými městy, kam obchodník cestuje. Přesněji řečeno, TSP je problém, ve kterém obchodník začíná v jednom městě, jednou navštíví všechna ostatní města a vrátí se do výchozího města, následně znovu hledá další kratší cestu, aby minimalizoval celkovou vzdálenost. Způsob získání optimálního řešení pro TSP je velmi obtížný kvůli existenci mnoha proveditelných řešení. [50]



Obrázek 13 Traveling Salesman Problem [37]

Například graf zobrazený na obrázku. Cesta obchodního cestujícího by měla být 1-2-4-3-1. To znamená že cena této cesty by byla 10+25+30+15 což je 80. [37]

Problém cestujícího prodejce lze popsat následovně:

$$TSP = \{(G, f, t): G = (V, E)$$

F je funkce *VxV -> Z*,

$t \in Z$,

G je graf, který obsahuje cestu obchodního cestujícího s náklady, které nepřekročí t. [42]

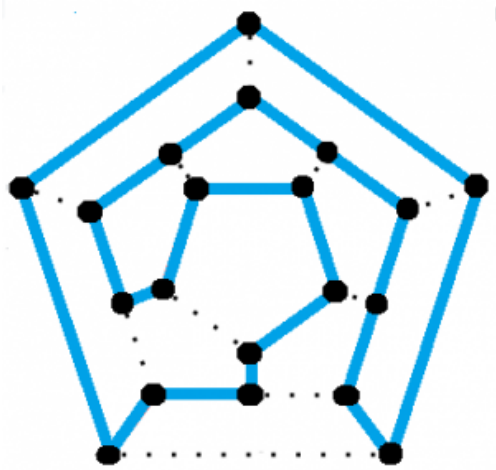
V počátcích počítačoví vědci doufali, že někdo vyvine vylepšený algoritmus pro řešení problému v rozumném čase. Zatímco vědci pokročili s řešením konkrétních scénářů, efektivní řešení pro celý problém obchodního cestujícího nalezeno nebylo. Algoritmus, který by vyřešil celý problém TSP by snad ani nebyl možný. [43]

TSP stále nachází uplatnění ve všech odvětvích. Efektivní řešení nalezená prostřednictvím TSP se používají v astronomii, počítačové vědě, a dokonce i při trasování vozidel. Astronomové používají TSP k určení pohybu dalekohledu mezi různými hvězdami v souhvězdí. [43]

V roce 1972 Richard Karp dokázal, že problém Hamiltonova cyklu je NP-úplný. To znamená, že TSP je NP-složitý. S přidáním dalších měst do grafu se složitost výpočtu nejkratší trasy faktoriálně zvýší. Například ve čtyřech městech mohou být tři možné trasy. V šesti městech by ale mohlo být 360 možných tras. Pro tento problém neexistuje žádné polynomiální časově známé řešení. [37, 43]

5.1 Hamiltonovský cyklus

Hamiltonovský cyklus je uzavřená smyčka na grafu, kde je každý uzel (vertex) navštíven právě jednou. Takže hamiltonovský cyklus je cesta putující z bodu zpět k sobě samému a navštěvující každý uzel na cestě. Pokud má graf s více než jedním uzlem tento typ cyklu, nazýváme ho hamiltonovský graf. Neexistuje žádná rovnice nebo obecný trik, jak zjistit, zda má graf Hamiltonovský cyklus. Jediný způsob, jak to zjistit, je provést úplné a vyčerpávající vyhledávání a projít všechny možnosti. [44]



Obrázek 14 Hamiltonský cyklus [44]

5.2 P, NP, NP-úplný a NP-těžký problém

Třídy složitosti jsou základem teorie složitosti, která je ústředním tématem teoretické informatiky. Třída složitosti obsahuje sadu problémů, jejichž řešení vyžaduje podobný rozsah prostoru a času, například „všechny problémy řešitelné v polynomiálním čase s ohledem na velikost vstupu“, „všechny problémy řešitelné exponenciálním prostorem s ohledem na velikost vstupu“, a tak dále. Obvykle se prokáže, že problémy jsou v určité třídě složitosti spuštěním problému na abstraktním výpočetním modelu, obvykle na Turingově stroji. [49]

Některé z největších nevyřešených problémů v informatice mají co do činění s dokazováním, zda jsou dvě třídy složitosti ekvivalentní či nikoli, nejpozoruhodnějším příkladem toho je nechvalně známý problém P = NP – vědci dosud nebyli schopni dokázat ani vyvrátit tento problém. [49]

5.2.1 P problém

Třída P se skládá z těch úloh, které jsou řešitelné v polynomiálním čase, tj. tyto úlohy lze v nejhorším případě vyřešit v čase $O(nk)$, kde k je konstantní. Tyto problémy se nazývají ovladatelné, zatímco jiné se nazývají neřešitelné nebo super polynomiální. Formálně je algoritmus polynomiální časový algoritmus, pokud existuje polynom $p(n)$ takový, že algoritmus může vyřešit jakoukoli instanci velikosti n v čase $O(p(n))$. [40]

5.2.2 NP problém

NP složitý problém je kterýkoli z třídy výpočetních problémů, pro které nebyl nalezen žádný účinný algoritmus řešení. Nelze je vyřešit v polynomiálním čase, ale lze je ověřit

v polynomiálním čase. Očekáváme, že tyto algoritmy budou mít exponenciální složitost. Do této třídy patří mnoho významných problémů počítačové vědy – např. zde probíraný problém cestujícího obchodníka, problémy s uspokojením a problémy s pokrytím grafů. [38, 39, 40]

5.2.3 NP-úplný problém

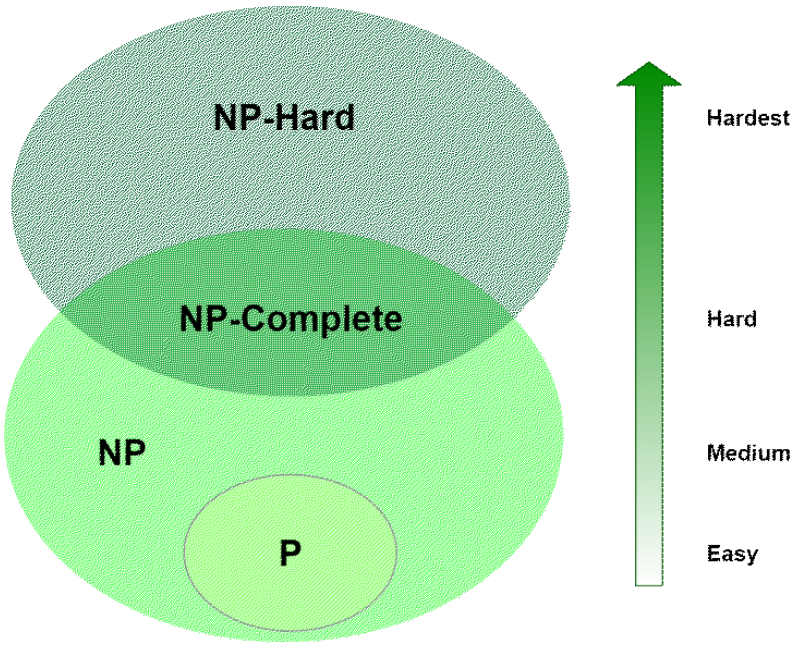
Takzvané snadné neboli zvládnutelné problémy mohou být řešeny počítačovými algoritmy, které běží v polynomiálním čase, tj. pro problém velikosti *n* je čas nebo počet kroků potřebných k nalezení řešení polynomiální funkcí *n*. Algoritmy pro řešení těžkých nebo neřešitelných problémů na druhé straně vyžadují časy, které jsou exponenciálními funkcemi velikosti problému *n*. Algoritmy s polynomiálním časem jsou považovány za účinné, zatímco algoritmy s exponenciálním časem jsou považovány za neefektivní, protože doba provádění druhého z nich roste mnohem rychleji, jak se zvětšuje velikost problémů. [38]

Problém se nazývá NP (nedeterministický polynom), pokud lze jeho řešení uhodnout a ověřit v polynomiálním čase. Nedeterministický znamená, že není dodržováno žádné konkrétní pravidlo pro odhad. Pokud je problém NP a všechny ostatní NP problémy jsou na něj redukovatelné v polynomiálním čase, je problém NP-úplný. Nalezení účinného algoritmu pro jakýkoli NP-úplný problém tedy znamená, že lze nalézt účinný algoritmus pro všechny takové problémy, protože jakýkoli problém patřící do této třídy může být přepracován do jakéhokoli jiného člena třídy. Není známo, zda budou někdy nalezeny nějaké polynomiální algoritmy pro NP-úplné problémy, a určení, zda jsou tyto problémy řešitelné nebo neřešitelné, zůstává jednou z nejdůležitějších otázek v teoretické informatice. Když musí být vyřešen NP-úplný problém, jedním přístupem je použití polynomiálního algoritmu k aproximaci řešení. Takto získaná odpověď nebude nutně optimální, ale bude přiměřeně blízko. [38]

5.2.4 NP-složitý problém

Obsahuje nejtěžší a nejsložitější problémy v informatice. Nejen, že je těžké je vyřešit, ale je také těžké je ověřit. Ve skutečnosti některé z těchto problémů ani nelze rozhodnout. Tyto algoritmy mají vlastnost podobnou těm v NP-úplných problémech – všechny lze redukovat na jakýkoli problém v NP. Kvůli tomu jsou v NP-Těžké třídě a jsou přinejmenším stejně těžké jako jakýkoli jiný problém v NP. Problém může být jak v NP, tak v NP-Hard, což je NP-Úplný problém. [39]

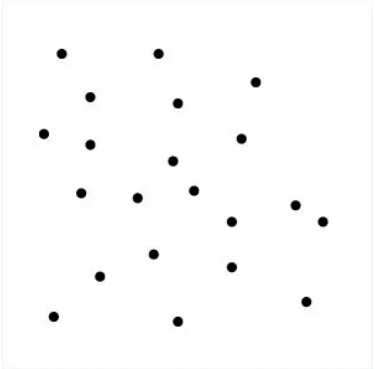
Takto charakteristika vedla k debatě o tom, zda problém cestujícího obchodníka je skutečně NP-Úplný problém. Vzhledem k tomu, že NP a NP-úplné problémy lze ověřit v polynomiálním čase, stačí pro umístnění algoritmu do NP-těžkých problémů také dokázat, že algoritmus nelze ověřit v polynomiálním čase. [39]



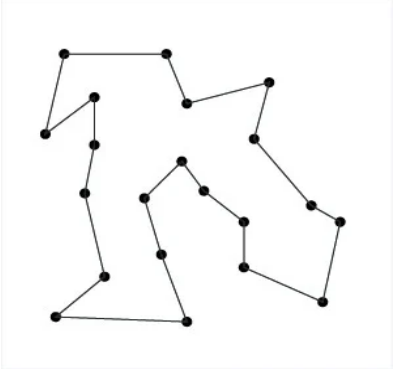
Obrázek 15 Třídy komplexnosti [39]

5.3 Řešení TSP

Problém lze vyřešit analýzou každé zpáteční trasy a určením nejkratší. S rostoucím počtem destinací však odpovídající počet zpátečních cest překonává možnosti i těch nejrychlejších počítačů. S 10 destinacemi může existovat více než 300 000 zpátečních permutací a kombinací. S 15 destinacemi by počet možných tras mohl přesáhnout 87 miliard. [41]



Obrázek 17 TSP graf řešení [41]



Obrázek 16 TSP graf [41]

5.3.1 Přístup hrubou silou

Přístup hrubou silou, také známý jako naivní přístup, počítá a porovnává všechny možné permutace tras nebo cest, aby určil nejkratší jedinečné řešení. Řešení TSP pomocí přístupu hrubou silou zahrnuje vypočítání celkového počtu tras a poté nakreslení a vypsání všech možných tras. Trasa s nejmenší vzdáleností je poté optimální řešení. [41]

5.3.2 Metoda „Branch and Bound“

Tato metoda rozděluje problém, který má být řešen, na několik dílčích problémů. Je to systém pro řešení řady dílčích problémů, z nichž každý může mít několik možných řešení a kde řešení zvolené pro jeden problém může mít vliv na možná řešení následných dílčích problémů. Na začátku se musí vybrat počáteční uzel a pak nastavit vazbu na velmi velkou hodnotu (třeba nekonečno). Dále je potřeba vybrat nejlevnější oblouk mezi nenavštíveným a aktuálním uzlem a poté přidat vzdálenost k aktuální vzdálenosti. Proces se opakuje, dokud je aktuální vzdálenost menší než hranice. Pokud je aktuální vzdálenost menší než hraniční, tak je řešení u konce. Nyní se sečtou vzdálenosti tak, aby se hranice rovnala aktuální vzdálenosti a tento postup se opakuje, dokud nebudou pokryty všechny oblouky. [41]

5.3.3 Metoda nejbližšího souseda

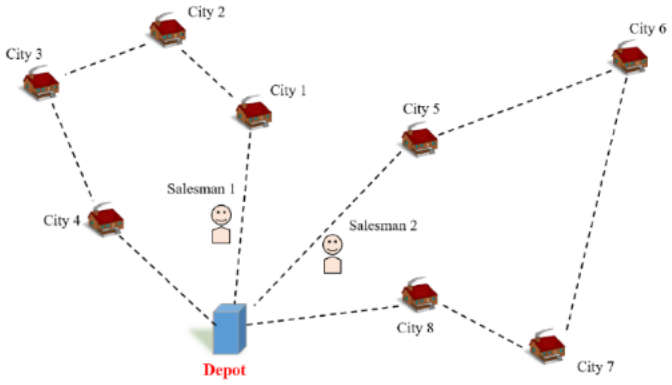
Toto je možná nejjednodušší heuristika TSP. Klíčem k této metodě je vždy navštívit nejbližší destinaci a vrátit se do původního města, až jsou navštíveny všechny ostatní města. Na začátku se vybere náhodné město a poté se vyhledá nejbližší nenavštívené město, do kterého se obchodní cestující přesune a tak dále dokud nejsou navštívena všechna města. Poté se obchodní cestující přesune zpět do prvního města. [41]

6 VARIACE CESTUJÍCÍCH OBCHODNÍKŮ

Tradiční model TSP byl modifikován několika výzkumníky, aby efektivně aplikoval tento problém na mnoho situací ve skutečném životě. Existují tři základní varianty TSP a to – symetrický TSP, asymetrický TSP a vícenásobný TSP. [50]

6.1 Problém více cestujících obchodníků

MTSP je zobecněním TSP. MTSP zahrnuje přidělení *m* obchodníků do *n* měst a každé město musí navštívit obchodník, přičemž vyžaduje minimální celkové náklady. Mnoho skutečných problémů lze modelovat jako MTSP, jako je problém s plánováním tiskového stroje, problém s distribucí nouzového materiálu, problém se směrováním vozidel, problém plánování UAV a problém plánování válcování za tepla. Ačkoli se dostalo TSP velké pozornosti, výzkum MTSP je omezený. [45, 46]

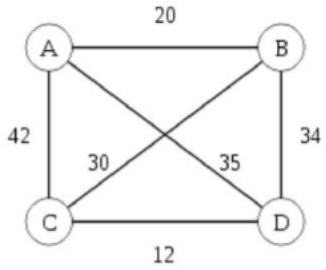


Obrázek 18 mTSP [54]

6.2 Symetrický problém cestujícího obchodníka – sTSP

Problém symetrického cestujícího obchodníka je problém nalezení nejkratšího hamiltonovského cyklu (nebo cesty) ve váženém konečném neorientovaném grafu bez smyček. Zdá se, že tento problém byl formulován asi před 45 lety a během posledních 25 let byl předmětem intenzivního zkoumání v kombinatorické optimalizaci. [51]

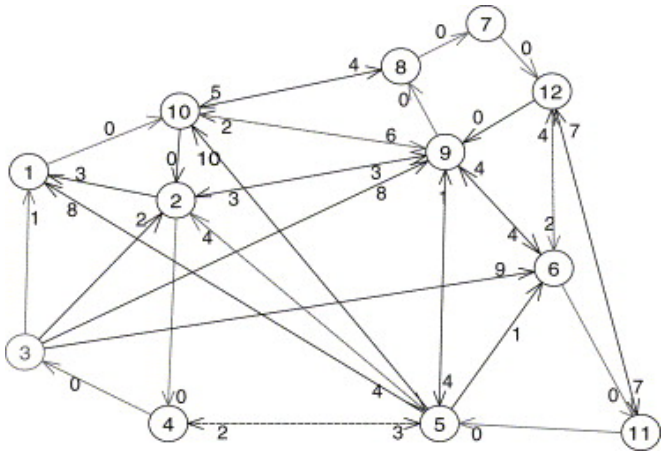
V nejběžnějším výkladu tohoto problému uzly grafu představují města, okraje grafu usměrňují cestovní trasy mezi městy a váhy vzdáleností mezi dvojicemi měst. V symetrickém TSP je vzdálenost mezi dvěma městy stejná v každém směru. Tato symetrie snižuje počet možných řešení na polovinu. Pro dvě lokace tedy platí $d_{ij} = d_{ji}$. [50, 52, 53]



Obrázek 19 Symetrický TSP [53]

6.3 Asymetrický problém cestujícího obchodníka

Klíčoví rozdíl mezi typickým TSP asymetrickým TSP je, že funkce vzdálenosti nemusí být symetrická. Tedy pro dvě lokace *u* a *v* je možné že $d(u, v) \neq d(v, u)$. Je zřejmé, že problém asymetrického TSP je NP-těžký (protože jde o striktní zobecnění symetrické verze. Problém asymetrického cestovního obchodníka se obvykle řeší pomocí algoritmů Branch-and-Bound a Branch-and-Cut. Obecné instance aTSP jsou často optimálně řešeny pomocí výčtových algoritmů, ve kterých je kontrolován pouze zlomek všech proveditelných řešení. [55, 56]



Obrázek 20 Asymetrický TSP [56]

6.4 Problém cestujícího obchodníka na základě zisku

Cílem variant TSP založených na zisku je identifikovat optimální cestu s efektivním kompromisem mezi inkasovaným ziskem (maximalizovaným) a cestovními náklady (minimalizovanými), kde není povinné navštěvovat všechny body a každý bod je spojen s pevným ziskem. Aplikace této kategorie problémů se objevují v kontextech, jako jsou problémy se směrováním vozidel a plánování úloh. Varianty TSP této kategorie se dělí na Problém selektivního cestujícího obchodníka (STSP), Problém cestujícího obchodníka

s kvótou (Quota TSP), Problém ziskových cest (PTP), Problém cestujícího kupujícího (TPP), Generalizovaný TSP (GTSP) [50]

6.4.1 Problém selektivního cestujícího obchodníka (STSP)

V STSP je cílem najít cestu, která maximalizuje inkasovaný zisk, a aby cestovní náklady nepřesáhly předem stanovenou hodnotu. Cílem je tedy najít optimální trasu, včetně depa v grafu, která maximalizuje inkasovaný zisk s cestovními náklady v dané minimální hranici. [50]

6.4.2 Problém cestujícího obchodníka s kvótou

TSP s kvótou se snaží najít cestu, která minimalizuje cestovní náklady a jejíž inkasovaný zisk není menší než přednastavená hodnota p_{min}. TSP s kvótou byl vytvořen vědcem B. Awerbuchem. Záměr kvótového TSP lze doložit příkladem identifikace optimální cesty včetně depa, která minimalizuje cestovní náklady a jejíž inkasovaný zisk by nikdy neměl být menší než p_{min}. [50]

Awerbuch představil problém kvóty TSP bez souvislosti s praktickou aplikací, ale navrhl následující scénář. Prodejce musí prodat kvótu p_{min} kartáčů. Prodejce má mapu s umístněním měst, zná vzdálenosti mezi městy a ví, kolik kartáčů by mohl v každém městě prodat. Jeho cílem je cestovat po městech, minimalizovat cestovní náklady a zároveň prodat požadovanou kvótu kartáčů. [50]

6.4.3 Problém ziskových cest (PTP)

Problém ziskových cest kombinuje STSP a TSP s kvótou. PTP spočívá v minimalizaci cestovních nákladů mínus zisk. Jeho objektivní funkce by se dala přirozeněji popsat jako maximalizovaný celkový zisk mínus celkové cestovní náklady. [50]

6.4.4 Problém cestujícího kupujícího (TPP)

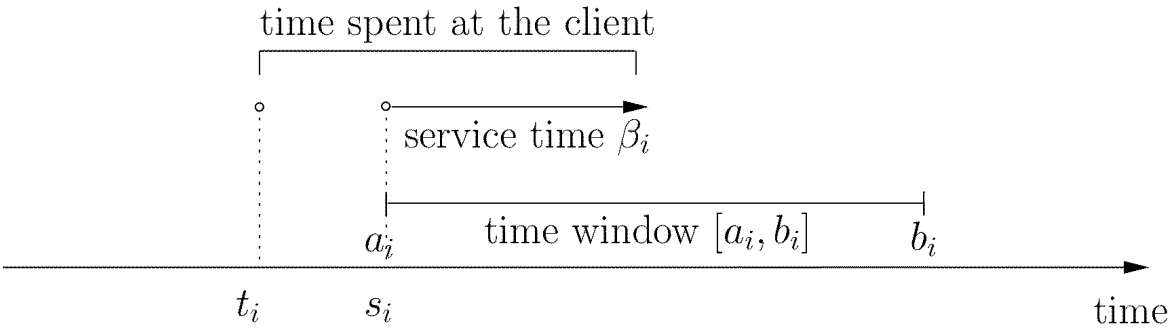
Problém cestujícího kupujícího je dobře známá zobecněná forma TSP. Uvažujme množinu produktů a množinu trhů, kde každý trh je dán s omezeným množstvím každého produktu za specifickou sazbu. TPP spočívá ve výběru podmnožiny trhů tak, aby bylo možné uspokojit danou poptávku po každém produktu, čímž se minimalizují náklady na směrování a nákupní náklady. [50]

6.4.5 Generalizovaný TSP (GTSP)

Generalizovaný problém cestujícího obchodníka je běžnou variantou klasického symetrického problému cestovního obchodníka, ve kterém jsou uzly rozděleny do shluků a obchodník musí navštívit alespoň jeden uzel pro každý shluk. [50]

6.5 Problém cestujícího obchodníka na základě časových oken

Problém cestovního obchodníka s časovými okny je podobný TSP s tím rozdílem, že města musí být navštíveny v daném časovém okně. Toto přidané časové omezení činí problém v praxi ještě obtížnějším. Pro tento problém neexistuje žádná společná definice, která by byla ve vědecké komunitě všeobecně přijímána. Základní myšlenkou je najít cestu s minimálními náklady, která navštíví každé ze sady měst přesně jednou, přičemž každé město musí být navštíveno v daném časovém okně. [58, 59]



Obrázek 21 Návštěva města v TSP na základě časových oken [58]

II. PRAKTICKÁ ČÁST

7 POPIS EXISTUJÍCÍCH APLIKACÍ PRO PLÁNOVÁNÍ NÁKUPU

V první kapitole praktické části diplomové práce jsou popsány vybrané současné aplikace pro plánování nákupu a popsány jejich nedostatky na operačním systému iOS.

Vybrané aplikace jsou

- Anylist - https://apps.apple.com/us/app/anylist-grocery-shopping-list/id522167641
- Our Groceries - https://apps.apple.com/us/app/our-groceries-shopping-list/id325851015
- Shoppka - https://apps.apple.com/cz/app/shoppka/id1535084412?l=cs
- Listonic - https://apps.apple.com/app/listonic-grocery-shopping-list/id331302745

7.1 Anylist

Autoři popisují aplikaci na AppStore takto

AnyList je nejlepší způsob, jak vytvořit nákupní seznamy potravin a shromažďovat a organizovat své recepty. Snadno a zdarma sdílejte seznam se svým partnerem nebo spolubydlícími. Změny se okamžitě projeví na každém iPhonu nebo iPadu. AnyList byl uveden v App Store jako „Nová a pozoruhodná“, „Skvělá bezplatná aplikace“ a jedna z „10 základních“ aplikací pro produktivitu. [47]

7.1.1 Popis aplikace

Aplikace obsahuje celkově 5 menu položek – Nákupní seznam, Recepty, Plánovač jídel, Obsáhlé nastavení aplikace a možnost koupit si lepší účet v aplikaci, který přináší různé výhody.

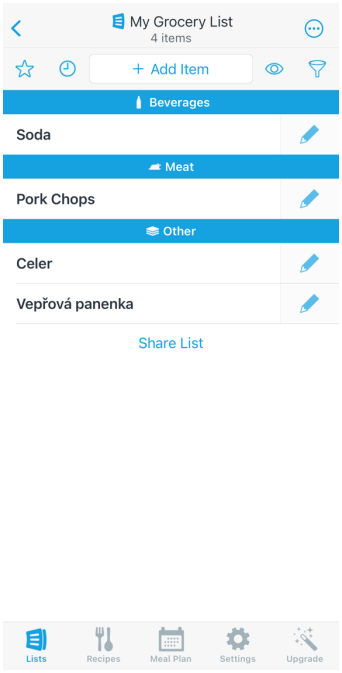
7.1.2 Funkce aplikace

Nákupní seznam

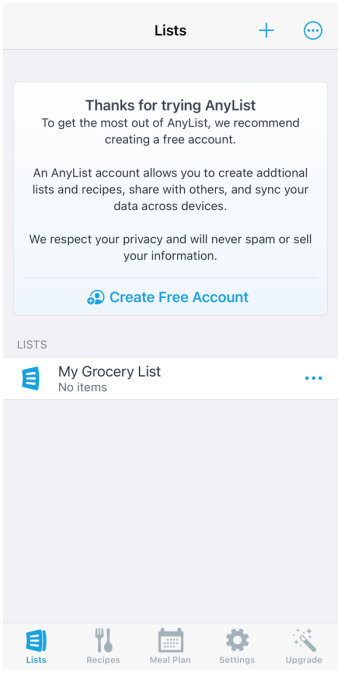
V nákupním seznamu lze přidávat a mazat položky, tuto funkci lze ovládat i přes hlasového asistenta Siri a k těmto položkám jde přidávat poznámky, cenu, počet, velikost balení, fotku a obchody, ve kterých lze zakoupit. Většina těchto atributů lze přidat pouze s premium účtem. Položky jdou odškrtávat jedním klepnutím. Aplikace taktéž podporuje funkci přidat více nákupních seznamů. Položky jsou rozděleny v kategoriích a aplikace podporuje

vytvoření vlastních kategorií. Pořadí kategorií lze libovolně přesouvat. Dále je zde možnost vytvořit si oblíbené položky seznamu.

Aplikace podporuje sdílení seznamů pomocí emailu. A daný uživatel, se kterým je seznam sdílen, by měl vidět položky u sebe na zařízení. Tato funkce je dostupná pouze s přihlášeným účtem. V aplikaci lze položky též přesouvat nebo kopírovat mezi seznamy.



Obrázek 22 Položky v seznamu



Obrázek 23 Nákupní seznamy

Recepty

Aplikace umožňuje vytvářet recepty nebo je importovat z externího zdroje. Přísady z těchto receptů jde poté přenášet do nákupního seznamu. Taktéž v nich je možno vyhledávat podle názvu nebo přísad. Recepty lze třídit do složek a sdílet je pomocí emailu. Aplikace obsahuje stejnou funkci jako u listů – sdílení receptů mezi účty v aplikaci.

Plánovač jídel

V plánovači jídel je zobrazený jednoduchý kalendář, do kterého jdou přidávat recepty z aplikace, tato funkce je ale pouze pro předplacený účet.

Zhodnocení aplikace

Základní funkce aplikace – správa listů, receptů a položek jsou i v bezplatné verzi, což postačí na jednoduché plánování nákupu každému běžnému uživateli. Nevýhodou je, že spousta užitečných funkcí je schováno za paywallem, jako například výše zmíněný plánovač jídel, filtrace produktů nebo import receptů z externích zdrojů. Taktéž obsáhlost aplikace z ní dělá vizuálně nepřehlednou a složitou. Aplikace je pouze v angličtině, takže pro české uživatele je nemožné používat spoustu doplňujících funkcí – jako jsou třeba kategorie z aplikace, pokud nechtějí udržovat svůj nákup v angličtině. Oproti vyvíjené aplikaci Šopák zde chybí jakékoliv plánování nákupu z pohledu nejkratší cesty strávené při nákupu.

7.2 Our Groceries

Popis aplikace na App Store

OurGroceries automaticky udržuje seznam potravin vaší domácnosti aktuální a s nejnovějšími změnami – na mobilním telefonu každého člena rodiny a na našem webu. Téměř bez mléka? Přidejte si jej nyní do telefonu a váš partner ho může koupit při příštím nákupu.

Každá změna ve vašem sdíleném nákupním seznamu je viditelná během několika sekund na jakémkoli jiném iPhonu, hodinkách, iPodu touch, iPadu nebo jiném smartphonu, který je synchronizovaný s aplikací ve vašem telefonu. Podívejte se, jak jsou položky odškrtávány ze seznamu hned jak je váš partner odškrtne!

Můžete také sledovat klíčové ingredience ve svých oblíbených receptech a přidat je všechny najednou do nákupního seznamu.

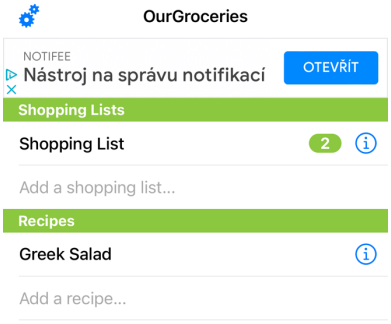
7.2.1 Popis aplikace

Aplikace je jednoduchá jak vzhledově, tak funkčně, obsahuje pouze jednu obrazovku, na které se spravují jak seznamy položek, tak recepty. Z této obrazovky lze též jít do obsáhlého nastavení aplikace.

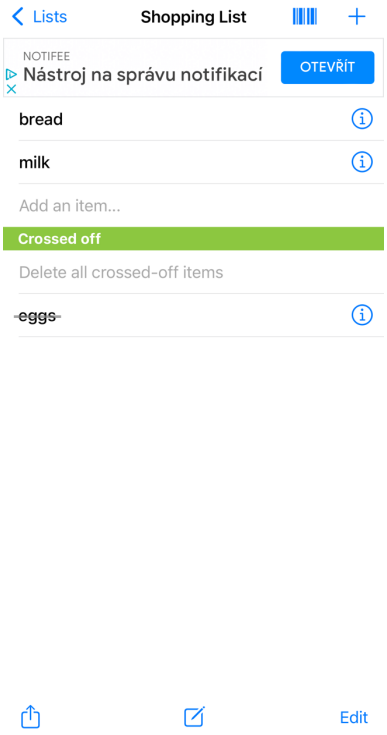
7.2.2 Funkce aplikace

V aplikaci lze vytvářet nákupní seznamy a do nich následně přidávat jednotlivé položky. Při přidávání těchto položek aplikace kontroluje předchozí nákupy a našeptává produkty, které již v minulosti byly přidány do některého ze seznamů. Tyto položky lze poté jedním kliknutím přesunout do spodní části, kde se zobrazují odškrtnuté.

Aplikace též podporuje kategorizaci položek, která se aktivuje po přidání první kategorie k produktu. Umí taktéž automaticky roztřídit zboží do kategorií. K položkám lze přidávat poznámky, množství a taktéž si uložit položku do oblíbených položek. Položky podporují i přidání fotografie produktu.



Obrázek 25 Úvodní obrazovka



Obrázek 24 Nákupní seznam

7.2.3 Zhodnocení aplikace

Aplikace je jednoduchá a veškeré funkce jsou zdarma. Je zde možnost zaplatit si premium, ale to slouží jen k odstranění reklam – které jsou hlavní nevýhodou – a jako podpora vývojářů. Vzhled aplikace je obyčejný a UX je trochu nepřehledné. Taktéž je pouze v angličtině, a tak automatická kategorizace pro česky mluvící uživatele nebude fungovat. Oproti vyvíjené aplikaci Šopák zde taktéž není žádné plánování nákupu z pohledu nejkratší cesty strávené při nákupu.

7.3 Shoppka

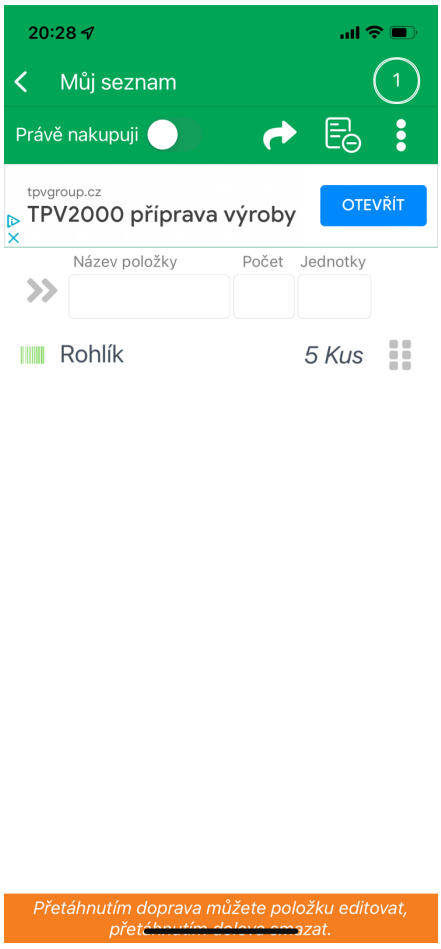
Aplikace je popsána na AppStore takto

Shopka je chytrý nákupní seznam. Umožní Vám snadno sdílet nákupní seznamy s členy vaší rodiny či domácnosti. Vytvořte si seznam na webu nebo ve svém mobilu a snadno jej sdílejte

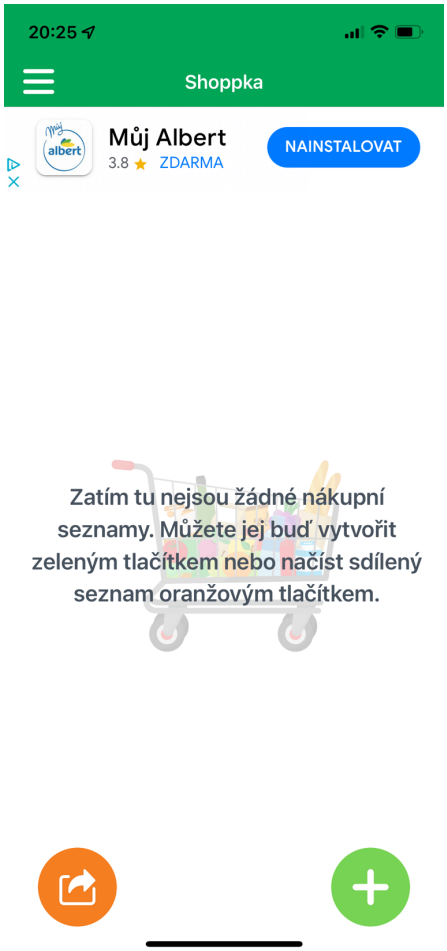
jedním kliknutím. Aplikace funguje na všech mobilních zařízeních nebo ve webovém prohlížeči. Všechny funkce jsou zdarma, za jednorázový poplatek se můžete zbavit reklam. [60]

7.3.1 Popis aplikace

Aplikace se skládá ze základní obrazovky, na které jdou vytvářet nákupní seznamy, načítat sdílené seznamy a postranního menu. V menu najdeme jednoduchou navigaci po aplikaci, šablony pro nákupní seznamy a jednoduché nastavení aplikace.



Obrázek 26 Shoppka – Nákupní seznam



Obrázek 27 Shoppka – úvodní obrazovka

Samotný seznam aplikace obsahuje přidávání položek, samotný nákupní seznam, tlačítko pro sdílení, tlačítko pro smazání všech položek, filtraci a přepínač právě nakupuji. Přidávání položek je pomocí 3 jednoduchých input polí, ve kterých se zadává název položky, počet položek a jednotka.

7.3.2 Funkce aplikace

V aplikaci lze vytvářet nákupní seznamy, které lze sdílet a přidávat do nich položky. Tyto položky jde mazat nebo editovat pomocí přetahu doleva nebo doprava. Nákupní seznamy lze sdílet pomocí kódu a načítat v jiném zařízení. Přepínač právě nakupuji omezí vypínání displeje. Šablony v aplikaci fungují obdobně jako nákupní seznam. Seznam taktéž obsahuje automatický našeptávač s předvolenými položkami – buď naposled použité nebo z databáze aplikace.

7.3.3 Zhodnocení aplikace

Aplikace je taktéž velmi jednoduchá, ale zařadil bych ji mezi nejhorší z popisovaných aplikací, protože je pro iOS velmi špatně navržené UX. Obsahuje plno vizuálních a UX bugů, které omezují při používání – věci se často překrývají přes sebe, a tak je nemožné se třeba dostat k určitému produktu. Je to pravděpodobně pouze port aplikace ze systému Android a jde poznat, že není orientována pro iOS systém. Veškerá navigace probíhá přes menu, takže je potřeba spousta kliků, než se někam uživatel dostane – iPhone nemá tlačítko zpět jako Android. Plusy této aplikace jsou, že obsahuje českou lokalizaci a je zdarma, ale s reklamou. Oproti vyvíjené aplikaci Šopák zde chybí též plánování nákupu z pohledu nejkratší cesty strávené při nákupu.

7.4 Listonic

Popis aplikace na AppStore

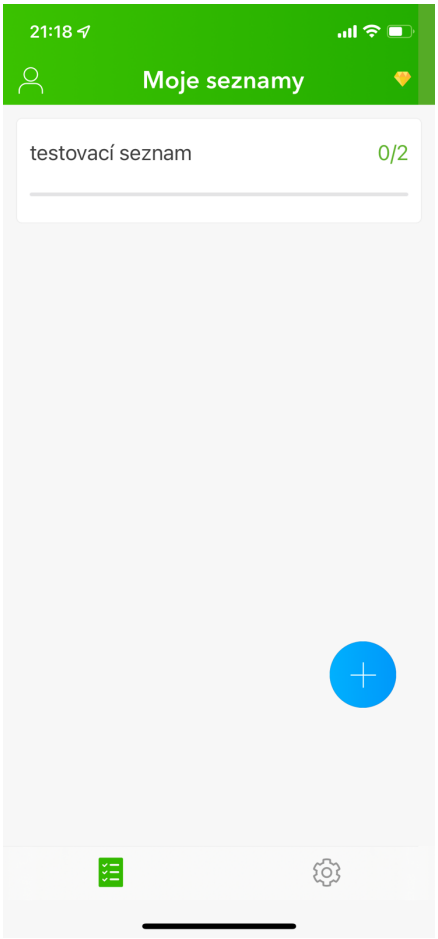
Nákupní seznam, který zlepšuje kvalitu vašeho nakupování potravin tím, že je jednodušší, rychlejší a hlavně chytřejší. To je vše, co můžete od nákupního seznamu chtít. Abych se ujistili, že vám Listonic šetří čas a peníze, zaměřili jsme se na rychlost, jednoduchost a uživatelsky přívětivé a praktické rozhraní. [61]

7.4.1 Popis aplikace

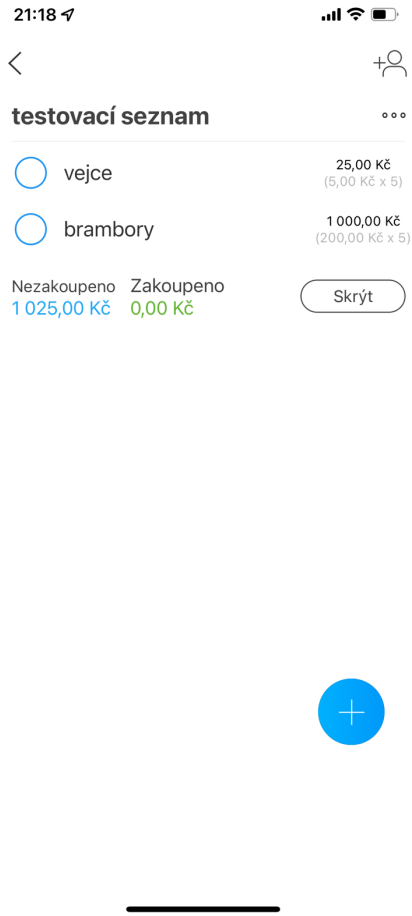
Aplikace se skládá z úvodní obrazovky, na které je list nákupních seznamů, obrazovka s informacemi o uživateli, nastavení aplikace a nákupního seznamu. Aplikace je jednoduchá, ale v dobrém smyslu.

7.4.2 Funkce aplikace

Aplikace umožňuje přidávat nákupní seznamy na úvodní obrazovce, ve kterých jdou přidávat produkty, které můžeme poté rozdělit do kategorií nebo jim přidělit cenu. Množství lze přidělit při přidávání produktu a aplikace následně dokáže spočítat cenu nákupu a jednotlivých položek podle množství. Nákupní seznamy lze poté sdílet mezi uživateli aplikace. Nastavení obsahuje obecné věci jako volba lokalizace aplikace, omezení vypínání obrazovky nebo správa notifikací.



Obrázek 29 Listonic – úvodní obrazovka



Obrázek 28 Listonic – nákupní seznam

7.4.3 Zhodnocení aplikace

Aplikace patří mezi nejlepší z hodnocených v této práci. Jednoduše se ovládá, je většinou přehledná a zobrazuje jen to co si uživatel zvolí. Má též českou lokalizaci. Mínus je že obsahuje chybu, kdy kategorie produktu se změní až po znovunačtení obrazovky seznamu. se a reklamu, které jde odstranit předplatným. Oproti vyvíjené aplikaci Šopák zde chybí jakékoliv plánování nákupu z pohledu nejkratší cesty strávené při nákupu.

8 DOKUMENTACE APLIKACE ŠOPÁK

Zadáním práce bylo vytvořit interaktivní aplikaci pro plánování nákupu. Pro tuto aplikaci byl zvolen název Šopák. Tato aplikace je vytvořena pomocí technologie Swift, SwiftUI a IDE Xcode, mocky aplikace byly vytvořeny v online aplikaci mockflow.com a loga a obrázky použité v aplikaci byly designovány v nástroji figma.com.

8.1 Content View

Content View je jádro celé aplikace. Je zde implementován Content View Model, který se stará o přepínání všech Views v aplikaci a taktéž obsahuje funkce pro operace s produkty v paměti telefonu. Skládá se ze dvou částí – první je samotné View, od kterého se odvíjí celá aplikace a druhá je třída ContentViewModel, která se stará o přepínání views a obsahuje funkce pro správu paměti.

8.2 View

View obsahuje Switch, který přepíná na ostatní View aplikace, podle toho, co mu Content Model pošle za hodnotu. Taktéž se v něm posílají funkce z Content View Modelu do nižších view v aplikaci.

```
struct ContentView: View {
    @StateObject var viewModel: ContentViewModel

    var body: some View {
        Group {
            switch viewModel.currentScreen {
            case .appView:
                AppView(
                    data: $viewModel.products,
                    displayView: $viewModel.currentScreen,
                    onDelete: viewModel.removeProduct,
                    onAdding: viewModel.addProduct,
                    onDeleteAll: viewModel.removeAllProducts
                )

            case .loginView:
                LoginView(
                    displayView: $viewModel.currentScreen
                )
            case .mapView:
                MapView(
                    displayView: $viewModel.currentScreen,
                    products: viewModel.products
                )
            }
        }
        .onAppear {
            viewModel.loadProducts()
        }
    }

}
```

Obrázek 30 Content View

8.3 Content View Model

Tato třída se stará, aby Content View dostalo vždy správné View, které má zobrazit pomocí výčtu obrazovek v aplikaci. Taktéž obsahuje funkce pro správu paměti v aplikaci, které využívají API pro komunikaci s pamětí. Funkce removeProduct slouží k mazání určitého prvku z paměti, addProduct naopak přidává produkt z obchodu do paměti, loadProducts načítá produkty z paměti do seznamu při zobrazení hlavního View a removeAllProducts smaže všechny produkty z paměti.

```
class ContentViewModel: ObservableObject {

    enum VisibleScreen: String {
        case appView
        case mapView
        case loginView
    }

    @Published var currentScreen: VisibleScreen = .loginView
    @Published var products: Products = []

    func removeProduct(_ onIndexSet: IndexSet) {
        self.products.remove(atOffsets: onIndexSet)
        ApiManager.store(products)
    }

    func addProduct(_ product: Product) {
        self.products.append(product)
        ApiManager.store(products)
    }

    func loadProducts() {
        self.products = ApiManager.get(Products()) ?? []
    }

    func removeAllProducts() {
        self.products.removeAll()
        ApiManager.store(products)
    }
}

struct ContentView_Previews: PreviewProvider {
    static var previews: some View {
        ContentView(viewModel: ContentViewModel())
    }
}
```

Obrázek 31 Content View Model

8.3.1 API Manager

Tento manager obsahuje pouze dvě funkce, a to pro získání a uložení hodnot do paměti. Funkce obsahují generiku, tak aby šla uložit do paměti jakákoliv hodnota. Tuto hodnotu následně dají nebo získají z UserDefaults – což je prostor pro ukládání dat do telefonu.

Aby tato generika fungovala, muselo být UserDefaults rozšířeno o dvě funkce s generikou save a load. Funkce save se pokusí zakódovat data do JSON a uložit je pod zvoleným klíčem a funkce load se pokusí zase dekódovat data z UserDefaults pod zvoleným klíčem.

```swift
struct ApiManager {

    static func get<T:PersistentData>(_ type: T) -> T? {

        return UserDefaults.load(type, key: T.storageKey)

    }

    static func store<T:PersistentData>(_ type: T) {

        UserDefaults.save(type, key: T.storageKey)

    }

}
```

Obrázek 32 API Manager

```swift
extension UserDefaults {

    static func save<T:Encodable>(_ data: T, key: DataType) {
        guard let value = try? JSONEncoder().encode(data) else {
            return
        }
        UserDefaults.standard.setValue(value, forKey: key.rawValue)
    }

    static func load<T:Decodable>(_ data: T, key: DataType) -> T? {

        guard let value = UserDefaults.standard.data(forKey: key.rawValue) else {
            return nil
        }
        return try? JSONDecoder().decode(T.self, from: value)
    }
}

```

Obrázek 33 UserDefaults rozšíření

8.4 Login View

Toto view je poskládáno ze tří komponent – Logo – což je komponenta, která byla vytvořena pro animované logo v aplikaci, Text – SwiftUI komponenta pro zobrazování textu a nakonec tlačítko, které nám umožňuje vstup do aplikace.

```swift
struct LoginView: View {

    @Binding var displayView: ContentViewModel.VisibleScreen

    var body: some View {
        VStack{
            Logo()
            Spacer()
            Text("Vítejte v Šopáku")
                .fontWeight(.bold)
                .padding()
                .font(.system(.title, design: .rounded))
            Spacer()
            Button(action: {
                displayView = .appView
            }) {
                Text("Pokračovat")

            }
            .padding(.top)
            .buttonStyle(GrowingButton())
                .frame(width: 280, height: 60, alignment: .center)
        }.background(
            Image("BackgroundIcons")
                .scaledToFill()
```

Obrázek 34 Login Obrazovka

Text komponenta má přidány atributy tak, aby se zobrazovala jako nadpis. Poté je tlačítko, které má vlastní styl, který je specifikován komponentou GrowingButton() a přepíná na další View ve Content View Modelu. Mezi textem a tlačítkem je komponenta Spacer(), která rozloží místo mezi komponentami. Nakonec je na celý VStack – což je pole, které skládá komponenty vertikálně – použit atribut .background, do kterého je vložena komponenta Image s obrázkem pozadí aplikace, které bylo též vytvořeno v aplikaci Figma.

Obrázek 35 Login View - aplikace

8.4.1 Growing Button

Tato komponenta je navržená pro stylování tlačítek v aplikaci. Obsahuje funkci, která vyrenderuje tlačítko podle zadaných atributů. Má taktéž nastavenou animaci, která zvětší a poté zmenší tlačítko, aby byl poznat klik na tlačítko.

```swift
import SwiftUI


struct GrowingButton: ButtonStyle {
    func makeBody(configuration: Configuration) -> some View {
        configuration.label
            .padding()
            .frame(width: 150)
            .background(Color.accentColor)
            .foregroundColor(.white)
            .cornerRadius(20)
            .scaleEffect(configuration.isPressed ? 1.2 : 1)
            .animation(.easeOut(duration: 0.2), value: configuration.isPressed)
    }
}
```

Obrázek 36 Growing Button

8.4.2 Logo

Logo je zobrazeno jako normální komponenta image, která je doplněna o atributy tak, aby gradient v jejím pozadí rotoval. Toto bylo dosaženo přidáním atributu .hueRotation(), nad kterým byla následovně zavolána animační funkce, která je nastavena, aby se opakovala do nekonečna.

```swift
import SwiftUI

struct Logo: View {
    @State private var animateGradient = false
    var body: some View {
        Image("ListImage")
            .resizable()
            .ignoresSafeArea()
            .hueRotation(.degrees(animateGradient ? 100 : 0))
            .onAppear {
                withAnimation(.linear(duration: 2.0).repeatForever(autoreverses: true)) {
                    animateGradient.toggle()
                }
            }
            .scaledToFill()
            .frame(height: 150)
    }
}
```

Obrázek 37 Komponenta Logo

Toto logo bylo designováno v aplikaci figma.com, použila se zde ikona nákupního košíku a mapového špendlíku, pod kterou se přidal gradient. Pro úvodní obrazovku aplikace byla vytvořena invertovaná verze.

Obrázek 38 Šopák Logo

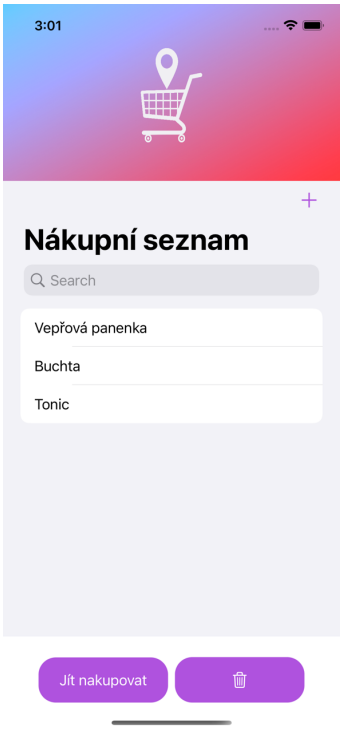
Obrázek 39 Šopák Splash

8.5 App View

App View je hlavní View v aplikaci, slouží jako rozcestník do dalších View – jako jsou mapa nebo obchod. Skládá se z komponenty Logo, poté obsahuje komponentu ProductChecklist a HStack (Swift UI komponenta, která řadí prvky v ní horizontálně), která obsahuje dvě tlačítka. První je DeleteAllButton pro mazání seznamu a druhé tlačítko Shopping Button pro přepínání na mapu – toto tlačítko je vypnuté pokud seznam neobsahuje žádný produkt.

```
3  struct AppView: View {
4
5      @Binding var data: Array<Product>
6      @Binding var displayView: ContentViewModel.VisibleScreen
7      let onDelete: (IndexSet) -> ()
8      let onAdding: (Product) -> ()
9      let onDeleteAll: () -> ()
10
11     var body: some View {
12         VStack {
13             Logo()
14             ProductChecklist(myList: data, onDelete: onDelete, onAdding: onAdding)
15             HStack {
16                 ShoppingButton(displayView: $displayView, myList: data)
17                 DeleteAllButton(onDeleteAll: onDeleteAll)
18             }
19
20         }
21     }
22 }
```

Obrázek 40 AppView



Obrázek 41 App View - aplikace

8.6 Product Checklist

Hlavní seznam produktů v aplikaci. Tento seznam obsahuje zvolené nebo vlastní produkty z obchodu, které poté předává mapě.

```
struct ProductChecklist: View {
    @State private var searchText = ""
    @State private var showingSheet = false
    let myList: [Product]
    let onDelete: (IndexSet) -> ()
    let onAdding: (Product) -> ()

    var body: some View {
        NavigationView {
            List {
                ForEach(searchResults, id: \.self) { product in
                    Text(product.name)
                }.onDelete { indexSet in
                    onDelete(indexSet)
                }
            }
            .navigationTitle("Nákupní seznam")
            .navigationBarItems(trailing: Button(action: {
                showingSheet.toggle()
            }) {
                Image(systemName: "plus")
            }
            )
            .sheet(isPresented: $showingSheet) {
                ProductView(isPresented: $showingSheet, myList: myList, onAdding: onAdding)
            }
        }.searchable(text: $searchText, placement: .navigationBarDrawer(displayMode: .always)) {
            ForEach(searchResults, id: \.self) { result in
                Text("Hledáte  \(result.name)?").searchCompletion(result.name)
            }
        }
    }
    var searchResults: [Product] {
        if searchText.isEmpty {
            return myList
        } else {
            return myList.filter { $0.name.contains(searchText)  }
        }
    }
}
```

Obrázek 42 Product Checklist

Seznam se skládá ze samotného seznamu, který je implementován pomocí komponenty NavigationView, která dále obsahuje komponentu List. Nad tímto seznamem lze vyhledávat, díky atributu .searchable. Tento list zobrazuje výsledek pomocné funkce searchResults, která kontroluje obsah proměnné searchText, pokud je v této proměnné uložen nějaký text, vrátí jen ty produkty, které obsahují tento text. Pokud je proměnná prázdná, vrátí celý seznam produktů. Hledání v seznamu obsahuje též našeptávač, který může automaticky doplnit hledaný produkt, toho je docíleno pomocí atributu .searchCompletion.

List obsahuje také atribut .navigationBarItems, který umožní v hlavičce listu zobrazit tlačítko +, pomocí kterého lze přidávat produkty z obchodu do seznamu. Po kliknutí na toto tlačítko se uživateli zobrazí modální okno, které je docíleno pomocí atributu .sheet. Toto modální okno zobrazuje samotný obchod.

8.7 Struktura Product

Tato struktura je Model v našem CVM, obsahuje všechny informace o produktu – automatické ID, které se generuje pomocí hash hodnoty jména produktu, jméno, bod x a bod y. Taktéž obsahuje statickou funkci pro získávání produktů z obchodu, který je vytvořen v json souboru Data. Tato funkce získá cestu k souboru a následně dekóduje data pomocí funkce JSONDecoder(). Pokud vše proběhlo bez chyby, tak vrátí data, jinak vypíše chybu.

Následně obsahuje proměnnou point, která se skládá z hodnot X a Y a tvoří CGPoint – což je datový typ pro souřadnice ve Swift. Nakonec vytváří pole pro produkty a rozšiřuje ho o storageKey, pomocí kterého poté aplikace hledá produkty v paměti.

```swift
public struct Product: Equatable, Identifiable, Hashable, Encodable, Decodable {

    public var id: String {
        String(describing: name.hashValue)
    }
    let name: String
    let x: Float
    let y: Float

    static func getProducts() -> Products {
        do {
            guard let path = Bundle.main.url(forResource: "Data", withExtension: "json") else {
                return []
            }

            let data = try Data(contentsOf: path)

            let json = try JSONDecoder().decode([Product].self, from: data)

            return json

        } catch let error {
            print(error)
            return []
        }
    }

    var point: CGPoint {
        CGPoint(x: CGFloat(x), y: CGFloat(y))
    }
}

public typealias Products = Array<Product>

extension Products: PersistentData {
    static var storageKey: DataType { .products }

}
```

Obrázek 43 Struktura Product

8.8 Product View

Product View je jedno z nejobsáhlejších View v aplikaci Šopák. Funguje jako virtuální obchod, ze kterého se přidávají předvolené nebo vlastní produkty do uživatelského seznamu produktů. Je vytvořený jako modální okno, které zobrazuje produkty ze souboru Data.json a vykresluje je jako tlačítka v LazyVGrid (Komponenta, která vykresluje obsah vertikálně, ale jen až je tento obsah potřeba). Tyto tlačítka obsahují pouze název produktu.

```
16      var body: some View {
17          VStack {
18              Text("Vítejte v našem obchodě")
19                  .padding()
20                  .font(.system(.title, design: .rounded))
21              Text("Zvolte položku")
22                  .font(.system(.title2, design: .rounded))
23              ZStack {
24                  Rectangle()
25                      .foregroundColor(Color("LightGray"))
26                  HStack {
27                      Image(systemName: "magnifyingglass")
28                      TextField("Hledat..", text: $searchText)
29                  }
30                  .foregroundColor(.gray)
31                  .padding(.leading, 13)
32              }
33                  .frame(height: 40)
34                  .overlay(
35                      RoundedRectangle(cornerRadius: 13)
36                      .stroke(Color.black, lineWidth: 2))
37                  .cornerRadius(13)
38                  .padding()
39              ScrollView {
40                  LazyVGrid(columns: twoColumnGrid, spacing: 20) {
41                      ForEach (searchResults) { product in
42                          Button(action: {
43                              onAdding(product)
44                              isPresented = false
45                          }) {
46                              Text(product.name)
47
48                          }
49                          .buttonStyle(GrowingButton())
50                      }
51                  }.padding()
52              }.background(
53                  Image("BackgroundIcons")
54                      .resizable()
55                      .aspectRatio(contentMode: .fill)
56              )
57              Button(action: {
58                  showingSheet.toggle()
59              }) {
60                  Text("Vlastní produkt")
61              }
62              .buttonStyle(GrowingButton())
```

Obrázek 44 Product View

Nad tímto seznamem lze též vyhledávat, funkce je stejná jako v ProductChecklist View. Toto vyhledávání je ale vytvořeno pomocí TextFieldu, který byl nastylován tak, aby vypadal jako systémové pole vyhledávání, protože implementace tohoto seznamu nepodporuje atribut .searchable. View je taktéž ošetřeno proti přidání stejného prvku dvakrát, a to filtrovací funkcí, která odebere přidaný prvek z obchodu.

```
104        var filteredProducts: [Product] {
105            Product.getProducts()
106                .filter { product in
107                    !myList.contains(product)
108                }
109
110        }
```

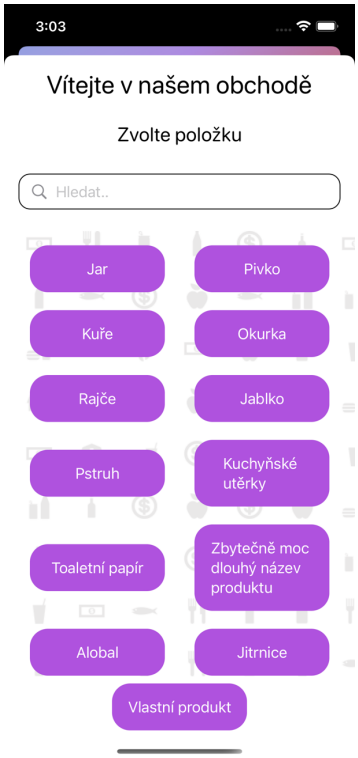
Obrázek 45 Filtrovací funkce

Obsahuje též tlačítko pro přidání vlastního produktu, které zobrazí další modální okno sheet. V tomto okně může uživatel vyplnit název a souřadnice produktu a kliknout na tlačítko Přidat produkt a produkt se přidá zrovna do uživatelského nákupního seznamu.

```
63              .sheet(isPresented: $showingSheet) {
64                  VStack{
65                      Text("Přidat vlastní produkt")
66                          .font(.system(.title, design: .rounded))
67                      HStack {
68                          Text("Název: ")
69                              .padding()
70                          TextField("", text: $name, prompt: Text("Název"))
71                              .padding()
72                              .textFieldStyle(.roundedBorder)
73                      }
74                      HStack {
75                          Text("X: ")
76                              .padding()
77                          TextField("", value: $x, formatter: NumberFormatter(), prompt: Text("X"))
78                              .padding()
79                              .keyboardType(.decimalPad)
80                              .textFieldStyle(.roundedBorder)
81                      }
82                      HStack {
83                          Text("Y: ")
84                              .padding()
85                          TextField("", value: $y, formatter: NumberFormatter(), prompt: Text("Y"))
86                              .padding()
87                              .keyboardType(.decimalPad)
88                              .textFieldStyle(.roundedBorder)
89                      }
90                      Button(action: {
91                          onAdding(Product(name: name, x: x, y: y))
92                          isPresented.toggle()
93                          showingSheet.toggle()
94                      }) {
95                          Text("Přidat produkt")
96
97                      }
98                      .frame(width: 300)
99                      .buttonStyle(GrowingButton())
00                  }
01              }
02          }
03      }
```

Obrázek 46 Vlastní produkt – modal

Toho je docíleno pomocí 3 textových polí, první pro jméno je klasické String textové pole, které má nastavený prompt a styl pole. Souřadnice jsou taktéž textová pole, ale mají povolenou pouze číselnou klávesnici pomocí atributu .keyboardType.



Obrázek 47 Product View - aplikace

8.9 Map View

Map View je základní rozcestník pro mapu, obsahuje pouze dvě komponenty a to TSPView – které řeší samotnou mapu a tlačítko Zpět, na kterém je opět použit styl GrowingButton() a přepíná CVM na hodnotu .appView – což znamená že vrací zpět na ProductChecklist View.

```
3  struct MapView: View {
4
5      @Binding var displayView: ContentViewModel.VisibleScreen
6      let products: Products
7
8      var body: some View {
9          VStack {
10             TSPView(products: products)
11             Button(action: {
12                 displayView = .appView
13             }) {
14                 Text("Zpět")
15             }
16             .buttonStyle(GrowingButton())
17         }
18     }
19 }
```

Obrázek 48 Map View

8.10 TSP View

TSP View je hlavní komponentou v Map View, do této komponenty se předává seznam uživatelských produktů a toto View je zobrazí na mapě obchodu a ukáže cestu. V tomto View je použitý ZStack (pole, které vykresluje produkty i s osou Z, takže můžou být přes sebe), ve kterém je ForEach, který zobrazí uživatelské produkty a rozmístí je na mapě, podle souřadnic produktu. ForEach zobrazuje pro každý prvek kolečko – které je vytvořeno jako barva, co má masku Circle() a název produktu v komponentě text. Vstup do obchodu je udělán stejně, ale mimo ForEach. Čára, která spojuje prvky je vykreslena pomocí atributu .background, který volá pomocnou funkci makePath, ta očekává dva atributy, a to start čáry a konec čáry. Pro start čáry je použit aktuální prvek a pro konec čáry se vypočítá nejbližší prvek pomocí funkce findClosest.

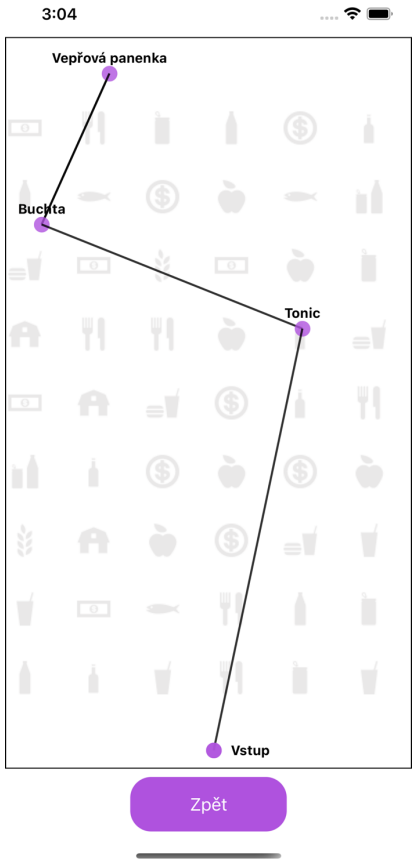
```
struct TSPView: View {

    @State var products: Products
    var body: some View {
        ZStack {
            ForEach(products) { item in
                ZStack {
                    Color.accentColor
                        .mask(Circle())
                        .frame(width: 15, height: 15)
                        .position(x: CGFloat(item.x), y: CGFloat(item.y))
                        .background(makePath(start: item, end: findClosest(product:item)))
                        .opacity(0.8)
                    Text(item.name)
                        .fontWeight(.bold)
                        .padding()
                        .font(.footnote)
                        .truncationMode(.tail)
                        .cornerRadius(20)
                        .frame(width: 150)
                        .position(x: CGFloat(item.x), y: CGFloat(item.y))
                        .offset(y: -15)
                }
            }
            Color.accentColor
                .mask(Circle())
                .frame(width: 15, height: 15)
                .position(x: 200.0, y: 685.0)
                .opacity(0.8)
                .task {
                    startingPoint()
                }
            Text("Vstup")
                .fontWeight(.bold)
                .padding()
                .font(.footnote)
                .truncationMode(.tail)
                .cornerRadius(20)
                .frame(width: 150)
                .position(x: 200.0, y: 685.0)
                .offset(x: 35)
        }
        .border(Color.black, width: 1)
        .background(Image("BackgroundIcons"))

    }
```

Obrázek 49 TSP View

Bod u vstupu obchodu obsahuje ještě atribut .task, který spustí funkci startingPoint, která pouze přidá vstup do seznamu produktů, ale neuloží ho do paměti, aby ho pak neměl uživatel v seznamu.



Obrázek 50 TSP View - aplikace

8.10.1 Funkce makePath

Jednoduchá funkce, která vždy přesune start čáry na aktuální produkt a konec čáry na další nejbližší produkt, pokud existuje. Pokud neexistuje použije jako koncový bod startovní bod.

```
52      @ViewBuilder
53      func makePath(start: Product, end: Product?) -> some View {
54          Path() { path in
55              path.move(to: start.point)
56              path.addLine(to: end?.point ?? start.point)
57          }
58          .stroke(Color.black, lineWidth: 2)
59      }
```

Obrázek 51 Funkce makePath

8.10.2 Funkce findClosest

Funkce provede filtr elementů v poli na základě podmínky, kde id produktu nesmí být stejné. Poté pro každý prvek pole vytvoří dvojici (Tuple), kde na nulté pozici je prvek z pole a na první pozici je vzdálenost od bodu, která je vypočítaná pomocí funkce distance. Následně prvky seřadí od nejmenší vzdálenosti a vybere nultý prvek z přetransformovaného pole, který vrátí.

```
61      func findClosest(product: Product) -> Product? {
62          self.products
63              .filter { p in
64                  p.id != product.id
65              }
66              .map { p in
67                  (p, product.point.distance(from: p.point))
68              }
69              .sorted { p1, p2 in
70                  p1.1 < p2.1
71              }
72              .first?.0
73      }
```

Obrázek 52 Funkce findClosest

8.10.3 Funkce distance

Tato funkce vypočítá vzdálenosti dvou bodů pomocí funkce max(), která vybere nejvyšší prvek z daných argumentů. Poté vrátí maximální hodnotu z hodnot nebo Pythagorovu větu těchto hodnot.

```
81      func distance(from point: CGPoint) -> CGFloat {
82          let dx = max(point.x - x, x - point.x, 0)
83          let dy = max(point.y - y, y - point.y, 0)
84          return dx * dy == 0 ? max(dx, dy) : hypot(dx, dy)
85      }
86  }
```

Obrázek 53 funkce distance

ZÁVĚR

V rámci práce byly popsány vybrané technologie pro vývoj mobilních aplikací. Dále zde byl popsán problém obchodního cestujícího a jeho mutace a varianty. V praktické části byly popsány vybrané aplikace pro plánování nákupu a jejich nedostatky. Poslední část práce popisuje technickou dokumentaci interaktivní aplikace Šopák, která byla vytvořena na základě zadání.

Vybrané technologie k popisu byly Swift, Swift UI, což jsou technologie, které jsou zaměřené na vývoj pro zařízení od firmy Apple. Zde bylo popsáno, co je to Swift, jaké má funkce, výhody a nevýhody jazyka Swift. Dále byl jazyk Swift srovnán s jazykem Objective-C, který byl předchůdce Swiftu. Poté bylo popsáno nastavení vývojového prostředí a možnosti vývoje. Nakonec byly popsány nejvíce používané návrhové vzory v tomto jazyce. V další části této kapitoly bylo popsáno SwiftUI, kde bylo shrnuto, co je to SwiftUI, jaká je jeho syntaxe, nástroje a srovnání s jeho předchůdci UIKit a Appkit. V závěru SwiftUI kapitoly byly shrnuty výhody a nevýhody této technologie.

Další popisovanou technologií byl React Native, který je orientovaný jak na Android, tak na iOS, v této části byly zhodnoceny výhody a nevýhody tohoto frameworku, rozdíly mezi Swift/SwiftUI. Poté byly popsány nejpoužívanější návrhové vzory v JavaScriptu, což je jazyk frameworku React Native.

V poslední části popisu technologií byly popsány PWA – progresivní webové aplikace. Pomocí kterých lze vyvíjet prakticky pro jakoukoliv platformu, která podporuje podporovaný webový prohlížeč. Zde bylo popsáno, jak vytvořit PWA a výhody i nevýhody této technologie.

Následně se práce zabývala Obchodním cestujícím a jeho variantami. Kde bylo popsáno samotné TSP, poté Hamiltonovský cyklus, třídy obtížnosti algoritmu a možné řešení TSP. Následně byly shrnuty variace cestujících obchodníků – jako problém více cestujících obchodníků, symetrický a asymetrický TSP, problém cestujícího obchodníka na základě zisku a jeho varianty, a nakonec TSP na základě časových oken.

V první kapitole praktické části práce byly popsány vybrané aplikace, jejich funkce a následně byly zhodnoceny. Vybrané aplikace pro tuto práci byly – Anylist, Our Groceries, Shoppka a Listonic.

Závěrečná kapitola práce obsahuje technickou dokumentaci Šopák, která byla vyvinuta na základě zadání této diplomové práce. Je to interaktivní aplikace pro plánování nákupu, která dokáže plánovat nákup i z pohledu nejkratší cesty. Aplikace umožňuje přidávat předvolené produkty z obchodu nebo přidávat vlastní, následně poté vykreslí mapu. Mapa obsahuje jednu chybu, a to že když dvojice produktů vyhodnotí druhý produkt jako nejbližší tak se spojí mezi sebou, a ne s dalším produktem. Tato chyba ale není možná opravit bez změny architektury modelu, což by znamenalo předělat celé jádro aplikace.

SEZNAM POUŽITÉ LITERATURY

[1] SCHAFFER, Erin. What is Swift? Features, advantages, and syntax basics. In: Educative [online]. 27.10.2021 [cit. 2022-05-09]. Dostupné z: https://www.educative.io/blog/swift-programming

[2] Swift enviroment setup. W3 schools [online]. [cit. 2022-05-09]. Dostupné z: https://www.w3schools.in/swift/environment-setup

[3] Swift: The powerful programming language that is also easy to learn. Developer Apple [online]. [cit. 2022-05-09]. Dostupné z: https://developer.apple.com/swift/

[4] VIGNESHWARAN, P. Swift Vs Objective-C: Which Is Ideal For IOS App Development In 2021. W2ssolutions [online]. 3.8.2021 [cit. 2022-05-09]. Dostupné z: https://www.w2ssolutions.com/blog/swift-vs-objective-c-2021/

[5] C, Radislav a Gleb B. Top 5 Design Patterns in Swift for iOS App Development. Rubygarage [online]. 11.1.2020 [cit. 2022-05-09]. Dostupné z: https://rubygarage.org/blog/swift-design-patterns

[6] SUNDELL, John. Using the builder pattern in Swift. Swift by Sundell [online]. 8.4.2018 [cit. 2022-05-09]. Dostupné z: https://www.swiftbysundell.com/articles/using-the-builder-pattern-in-swift/

[7] Adapter. Refactoring Guru [online]. [cit. 2022-05-09]. Dostupné z: https://refactoring.guru/design-patterns/adapter

[8] LEAL, Vini. Design Patterns in Swift - Decorator. Vini Leal [online]. 8.11.2020 [cit. 2022-05-09]. Dostupné z: https://vinileal.com/design%20patterns/design-patterns-swift-decorator/

[9] Facade. Refactoring Guru [online]. [cit. 2022-05-09]. Dostupné z: https://refactoring.guru/design-patterns/facade

[10] GEEKSFORGEEKS. Template Method Design Pattern. Geeks for Geeks [online]. 18.10.2021 [cit. 2022-05-09]. Dostupné z: https://www.geeksforgeeks.org/template-method-design-pattern/

[11] SwiftUI. Developer Apple [online]. [cit. 2022-05-09]. Dostupné z: https://developer.apple.com/xcode/swiftui/

[12] JACOBS, Bart. SwiftUI Fundamentals: What Is SwiftUI. Cocoacasts [online]. [cit. 2022-05-09]. Dostupné z: https://cocoacasts.com/swiftui-fundamentals-what-is-swiftui

[13] UIKit. Developer Apple [online]. [cit. 2022-05-09]. Dostupné z: https://developer.apple.com/documentation/uikit

[14] Appkit. Developer Apple [online]. [cit. 2022-05-09]. Dostupné z: https://developer.apple.com/documentation/appkit

[15] BUTTON, Joshua. SwiftUI or UIKit: Which is Right For You?. Big Nerd Ranch [online]. 2.3.2021 [cit. 2022-05-09]. Dostupné z: https://bignerdranch.com/blog/learning-apples-swiftui-or-uikit-which-one-is-right-for-you-right-now/

[16] ZAITSEVA, Alina. SwiftUI vs UIKit: Benefits and Drawbacks. Steel Kiwi [online]. [cit. 2022-05-09]. Dostupné z: https://steelkiwi.com/blog/swiftui-vs-uikit/

[17] PASQUIER, Benoit. SwiftUI - What has changed in your MVVM pattern implementation. Benoit Pasquier [online]. 12.1.2020 [cit. 2022-05-09]. Dostupné z: https://benoitpasquier.com/swiftui-what-has-changed-in-mvvm-pattern-swift/

[18] THE TECH PLATFORM. Decorator Design Pattern in ASP.NET Core. The Tech Platform [online]. 19.7.2021 [cit. 2022-05-09]. Dostupné z: https://www.thetechplatform.com/post/decorator-design-pattern-in-asp-net-core

[19] Builder Design Pattern. Source making [online]. [cit. 2022-05-09]. Dostupné z: https://sourcemaking.com/design_patterns/builder

[20] Learning React Native: building mobile applications with JavaScript. Prosinec 2015. Beijing: O'Reilly, [2016]. ISBN 978-149-1929-001.

[21] MARCAK, Mariusz. REACT NATIVE PROS AND CONS – 2022 UPDATED. Page Pro [online]. 4.5.2022 [cit. 2022-05-16]. Dostupné z: https://pagepro.co/blog/react-native-pros-and-cons/

[22] OSADCHUK, Serhii. React Native vs Swift in 2022: Which One is Better for Your Project?. Do It Software [online]. 17.4.2022 [cit. 2022-05-16]. Dostupné z: https://doit.software/blog/react-native-vs-swift#screen17

[23] DEVEN. 7 JavaScript Design Patterns Every developer should know. Codesource [online]. [cit. 2022-05-16]. Dostupné z: https://codesource.io/javascript-design-patterns/

[24] Constructor Pattern. Educative.io [online]. [cit. 2022-05-17]. Dostupné z: https://www.educative.io/collection/page/5429798910296064/5725579815944192/5920633608208384

[25] PATEL, Devan. Prototype Design Pattern in JavaScript. Digital Ocean [online]. [cit. 2022-05-17]. Dostupné z: https://www.digitalocean.com/community/conceptual_articles/prototype-design-pattern-in-javascript

[26] Prototype Design Pattern. Geeks for Geeks [online]. 22.9.2022 [cit. 2022-05-17]. Dostupné z: https://www.geeksforgeeks.org/prototype-design-pattern/

[27] GUPTA, Mayank. Module Pattern in JavaScript. Javascript plain english [online]. [cit. 2022-05-17]. Dostupné z: https://javascript.plainenglish.io/data-hiding-with-javascript-module-pattern-62b71520bddd

[28] The Module Pattern. O'Reilly [online]. [cit. 2022-05-17]. Dostupné z: https://www.oreilly.com/library/view/learning-javascript-design/9781449334840/ch09s02.html

[29] Singleton Pattern. Educative.io [online]. [cit. 2022-05-17]. Dostupné z: https://www.educative.io/collection/page/5429798910296064/5725579815944192/4890148815765504

[30] PRASANNA, Vijay. JavaScript Object Oriented Patterns: Factory Pattern [online]. 23.1.2019 [cit. 2022-05-17]. Dostupné z: https://www.digitalocean.com/community/tutorials/js-factory-pattern

[31] SINGH, Amrinder. Factory Design Pattern. Dev.to [online]. 11.9.2019 [cit. 2022-05-17]. Dostupné z: https://dev.to/amrindersinghdev/factory-design-pattern-3pah

[32] RICHARD, Sam a Pete LEPAGE. What are Progressive Web Apps?. Web.dev [online]. 24.2.2020 [cit. 2022-05-17]. Dostupné z: https://web.dev/what-are-pwas/

[33] WARDELL, Lindsay. Progressive Web Apps and Mobile Apps. Thisdot [online]. 22.7.2021 [cit. 2022-05-17]. Dostupné z: https://www.thisdot.co/blog/progressive-web-apps-and-mobile-apps

[34] DOMES, Scott. How to build a Progressive Web App. Creative bloq [online]. 26.10.2018 [cit. 2022-05-17]. Dostupné z: https://www.creativebloq.com/how-to/build-a-progressive-web-app

[35] PLUSZCZEWSKA, Bianka, Matt WARCHOLINSKI a Krystian KOŚCIELNIAK. Progressive Web Apps: Advantages and Disadvantages. Brainhub [online]. 10.3.2021 [cit. 2022-05-17]. Dostupné z: https://brainhub.eu/library/progressive-web-apps-advantages-disadvantages/

[36] TECHTARGET CONTRIBUTOR. Traveling salesman problem (TSP). Tech Target [online]. [cit. 2022-05-17]. Dostupné z: https://www.techtarget.com/whatis/definition/traveling-salesman-problem

[37] Travelling Salesman Problem | Set 1 (Naive and Dynamic Programming). Geeks for Geeks [online]. 6.8.2018 [cit. 2022-05-17]. Dostupné z: https://www.geeksforgeeks.org/travelling-salesman-problem-set-1/

[38] GREGERSEN, Erik. NP-complete problem. Britannica [online]. [cit. 2022-05-17]. Dostupné z: https://www.britannica.com/science/NP-complete-problem

[39] BAELDUNG. P, NP, NP-Complete and NP-Hard Problems in Computer Science. Baeldung [online]. 25.8.2021 [cit. 2022-05-17]. Dostupné z: https://www.baeldung.com/cs/p-np-np-complete-np-hard

[40] Design and Analysis P and NP Class. Tutorials point [online]. [cit. 2022-05-17]. Dostupné z: https://www.tutorialspoint.com/design_and_analysis_of_algorithms/design_and_analysis_of_algorithms_p_np_class.htm

[41] MA, Suzanne. Solving the Travelling Salesman Problem for deliveries. Routific [online]. 2.1.2020 [cit. 2022-05-17]. Dostupné z: https://blog.routific.com/travelling-salesman-problem

[42] TSP [online]. 2018 [cit. 2022-05-17]. Dostupné z: https://www.csd.uoc.gr/~hy583/papers/ch11.pdf

[43] What is the Traveling Salesman Problem (TSP)?. Route 4 me [online]. 4.3.2022 [cit. 2022-05-17]. Dostupné z: https://blog.route4me.com/traveling-salesman-problem/

[44] , Stephanie. Hamiltonian Cycle: Simple Definition and Example. Statistics how to [online]. 12.11.2017 [cit. 2022-05-17]. Dostupné z: https://www.statisticshowto.com/hamiltonian-cycle/

[45] Yang Shuai, Shao Yunfeng & Zhang Kai (2019) An effective method for solving multiple travelling salesman problem based on NSGA-II, Systems Science & Control Engineering, 7:2, 108-116, DOI: 10.1080/21642583.2019.1674220

[46] BEKTAS, Tolga. The multiple traveling salesman problem: an overview of formulations and solution procedures. Omega [online]. 2006, 34(3), 209-219 [cit. 2022-04-03]. ISSN 03050483. Dostupné z: doi:10.1016/j.omega.2004.10.004

[47] PURPLE COVER. AnyList: Grocery Shopping List. App Store [online]. [cit. 2022-05-17]. Dostupné z: https://apps.apple.com/us/app/anylist-grocery-shopping-list/id522167641

[48] OURGROCERIES. Our Groceries Shopping List. App Store [online]. [cit. 2022-05-17]. Dostupné z: https://apps.apple.com/us/app/our-groceries-shopping-list/id325851015

[49] MOORE, Karleigh a Alex CHUMBLEY. Complexity Classes. Brilliant [online]. [cit. 2022-05-17]. Dostupné z: https://brilliant.org/wiki/complexity-classes/

[50] K. Ilavarasi and K. S. Joseph, "Variants of travelling salesman problem: A survey," International Conference on Information Communication and Embedded Systems (ICICES2014), 2014, pp. 1-7, doi: 10.1109/ICICES.2014.7033850.

[51] Grötschel, Martin & Padberg, Manfred. (1978). On the Symmetric Travelling Salesman Problem: Theory and Computation. 10.1007/978-3-642-95322-4_12.

[52] Grötschel, Martin & Padberg, Manfred. (1979). On the symmetric travelling salesman problem I: Inequalities. Mathematical Programming. 16. 265-280. 10.1007/BF01582116.

[53] The traveling salesman problem. Ppt online [online]. [cit. 2022-05-17]. Dostupné z: https://en.ppt-online.org/31503

[54] Lee, Meng-Tse & Chen, Bo-Yu & Lu, Wen-Chi. (2019). Failure-Robot Path Complementation for Robot Swarm Mission Planning. Applied Sciences. 9. 3756. 10.3390/app9183756.

[55] GILBERT, Seth. The Asymmetric Traveling Salesman [online]. 2015 [cit. 2022-05-17]. Dostupné z: https://www.comp.nus.edu.sg/~gilbert/CS4234/2015/lectures/06.ATSP.pdf

[56] Marcel Turkensteen, Diptesh Ghosh, Boris Goldengorin, Gerard Sierksma, Iterative patching and the asymmetric traveling salesman problem, Discrete Optimization, Volume 3, Issue 1, 2006, Pages 63-77, ISSN 1572-5286

[57] The Travelling Salesman Problem with Time Windows (TSPTW). Acrogenesis [online]. [cit. 2022-05-17]. Dostupné z: https://acrogenesis.com/or-tools/documentation/user_manual/manual/tsp/tsptw.html

[58] BOLAND, Natashia, Mike HEWITT, Vu Duc MINH a Martin SAVELSBERGH. Solving the Traveling Salesman Problem with Time Windows Using Time-Expanded Networks. Research gate [online]. [cit. 2022-05-17]. Dostupné z: https://www.researchgate.net/publication/292251491_Solving_the_Traveling_Salesman_Problem_with_Time_Windows_Using_Time-Expanded_Networks

[59] MATĚJKA, Jiří. Shoppka - nákupní seznam. App Store [online]. [cit. 2022-05-17]. Dostupné z: https://apps.apple.com/cz/app/shoppka/id1535084412?l=cs

[60] LISTONIC. Grocery Shopping List Listonic. App Store [online]. [cit. 2022-05-17]. Dostupné z: https://apps.apple.com/app/listonic-grocery-shopping-list/id331302745

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

LLDB low-level debugger

REPL read-eval-print loop

LLVM Low Level virtual Machine

API Application programming interface

IDE Integrated Development Environment

UX User Experience

MVP Minimum viable product

PWA Progressive Web App

JS JavaScript

MD Man Day

DRY Don’t Repeat Yourself

HTML Hypertext Markup Language

CSS Cascading Style Sheets

HTTPS Hypertext Transfer Protocol Secure

SSL Secure Sockets Layer

RAIL Response Animation Idle and Load

NP Nondeterministic polynomial

TSP Traveling Salesman Problem

MTSP Multiple Traveling Salesman Problem

UAV Unmanned Aerial Vehicle

sTSP Symmetric Traveling Salesman Problem

aTSP Asymmetric Traveling Salesman Problem

mTSP multiple Traveling Salesman Problem

TSPTW Traveling Salesman with Time Windows

STPS Selective Traveling Salesman Problem

PTP Profitable Tour Problem

TPP Travelling Purchaser Problem

GTSP Generalized Travelling Salesman Problem

CVM Content View Model

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: CD S OBSAHEM DIPLOMOVÉ PRÁCE A ZDROJOVÉ SOUBORY APLIKACE ŠOPÁK