VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

KNIHOVNA FREKVENTOVANÝCH KOMPONENT PRO VÝVOJ APLIKACÍ V IOS

LIBRARY OF UI COMPONENTS FOR DEVELOPING APPLICATIONS IN IOS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **ADAM SALIH**
AUTHOR

VEDOUCÍ PRÁCE **Ing. MARTIN HRUBÝ, Ph.D.**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Salih Adam**
Program: Informační technologie
Název: **Knihovna frekventovaných komponent pro vývoj aplikací v iOS**
Library of UI Components for Developing Applications in iOS
Kategorie: Uživatelská rozhraní

Zadání:

1. Prostudujte programování aplikací pro iOS. Prostudujte podobu uživatelského rozhraní populárních aplikací z různých oborů.
2. Navrhněte sadu často používaných komponent UI, které nejsou přímo implementovány v knihovnách UIKit. Zaměřte se na výsuvné panely, formáty buněk tabulky a podobně. Dále se zaměřte efektivní specifikaci tzv. datasource pro komponenty.
3. Implementujte knihovnu komponent pro konvenční UIKit (Storyboard) a SwiftUI.
4. Sadu komponent prezentujte formou ukázkových aplikacích.

Literatura:

- Dle pokynů vedoucího.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Hrubý Martin, Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 3. listopadu 2021

Abstrakt

V této práci se zabývám analýzou a následnou implementací sady často používaných komponent, které nejsou součástí standardní sady uživatelských rozhraní. Tyto komponenty následně implementuji pomocí knihoven *UIKit* a *SwiftUI*. V této práci se dále zabývám implementací silně typových datových modelů a ověřuji implementaci sady komponent na demonstrační aplikaci.

Abstract

In this thesis I analyse and implement a set of frequently used components that are not part of the standard set of user interfaces. Then I implement these components using the UIKit and SwiftUI libraries. In this work I also focus on the implementation of strongly typed data models. In the end I verify the implementation of the set of components on a demonstration application.

Klíčová slova

Apple, iOS, iPhone, iPad, UIKit, SwiftUI, BottomSheet, DynamicContent, InheritableEnum, Uživatelské rozhraní, Xcode

Keywords

Apple, iOS, iPhone, iPad, UIKit, SwiftUI, BottomSheet, DynamicContent, InheritableEnum, User interface, Xcode

Citace

SALIH, Adam. *Knihovna frekventovaných komponent pro vývoj aplikací v iOS*. Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Martin Hrubý, Ph.D.

Knihovna frekventovaných komponent pro vývoj aplikací v iOS

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Martina Hrubého Ph.D.. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Adam Salih
11. května 2022

Poděkování

Rád bych poděkoval svému kolegovi Jiřímu Veverkovi za konzultace při psaní technické zprávy. Chtěl bych také poděkovat mojí přítelkyni Žanetě Muselíkové za neobyčejnou podporu v posledních dnech úsilí na této bakalářské práci. Na závěr bych chtěl poděkovat svým rodičům za podporu a důvěru při mém studiu.

Obsah

Kapitola 1

Úvod

Uživatelská rozhraní jsou velmi důležitou součástí vývoje softwaru, protože vytváří vrstvu mezi světem počítačů a světem běžných lidí. Tato vrstva s sebou nese velkou zodpovědnost. Pokud se udělá dobře, výsledek vytváří intuitivní nástroj, který je schopen ovládat každý. Pokud se ale udělá špatně, výsledek vytváří umělé překážky, které uživatele může odradit od používání výsledného nástroje, ať je jakkoliv užitečný.

Tato práce se zabývá uživatelskými komponenty, analýzou často používaných komponent, které nejsou součástí standardních komponent pro vytváření uživatelského rozhraní, definicí jejich chování a následnou implementací za pomocí knihoven *UIKit* a *SwiftUI*.

Druhá kapitola se zabývá základními prostředky pro práci s uživatelským rozhraním definovaných v knihovnách UIKit a SwiftUI.

Třetí kapitola se zabývá podrovnou analýzou komponent uživatelského rozhraní populárních aplikací, které vyhrály ocenění Apple Design Awards[2], které poslouží jako základ pro návrh výsledných komponent.

Čtvrtá kapitola se zabývá definicí jednotlivých komponent, jejich chováním a popisuje případy užití, které se dané komponenty snaží řešit.

Pátá a Šestá kapitola se zabývá implementaci těchto komponent za použití knihoven UIKit a SwiftUI, popisuje rozdíly přístupů knihoven a jejich výhody a nevýhody.

Sedmá kapitola popisuje implementaci knihovny, která umožňuje emulovat silně typovou dědičnost nad výčtovými typy, implementovanou v protokolově orientovaném paradigmatu.

Poslední, osmá, kapitola popisuje implementaci těchto komponent v demonstrační aplikaci, vytvořenou pouze pro účely této práce.

Kapitola 2

Teoretická východiska práce

Pro pochopení analýzy a implementací je potřeba si definovat používané uživatelských komponent implementované v knihovně *UIKit* a *SwiftUI*, které dohromady vytvářejí kompozici uživatelského rozhraní aplikací. Pro každou z komponent vyberu unikátní barvu a pro každou aplikaci při analýze vytvořím barevný obrázek, znázorňující možnou implementaci kompozice uživatelského rozhraní pomocí komponent z *UIKitu*.

2.1 UIKit komponenty

UIViewController

Je jednou z elementárních tříd reprezentující scénu v aplikaci. Každá scéna v navigační hierarchii obrazovek, implementované v UIKitu, musí z této třídy dědit. *UIViewController* může ve své struktuře obsahovat další *UIView*, či *UIViewControllery*, tvořící společně kompozici uživatelského rozhraní dané scény. [21]

UINavigationController

Je kontejner dědící z třídy *UIViewController* jejíž hlavní funkcí je řešení postupného přechodu navigace mezi scénami. Tyto scény jsou uložené za pomocí datového modelu stack a tedy obsahuje i stejné operace. Operace push zobrazí novou scénu, obdobně operace pop opustí aktuálně prezentovanou scénu a zobrazí scénu předchozí.[15]

UISplitViewController

S představením iPadu získal iOS podporu pro zařízení výrazné větším obsahem displeje. *UISplitViewController*, který dědí ze třídy *UIViewController* a je, podobně jako *UINavigationController*, kontejnerem dalších *UIViewControllerů*.[17] *UISplitViewController* ale narozdíl od *UINavigationControlleru*, který zobrazuje pouze jednu scénu v daný moment (pokud nepočítáme přechody mezi scénami), může zobrazovat více *UIViewControllerů* najednou. Zobrazované scény se nazávají *Primary* a *Secondary*, přičemž *UISplitViewController* v posledních verzích iOS podporuje zobrazení dalšího sloupce nazávaný, *Suplementary*. *Primary UIViewController* se zobrazuje jako boční lišta, která většinou umožňuje zobrazovat *detail* v *Secondary UIViewControlleru*.

Zda má *UISplitViewController* zobrazit více čí méně *UIViewControllerů* zaleží na tzv. *SizeClass* displaye, což je kategorizace šířky a výšky displeje, která může být buď *compact* či *regular*.

UIView

Je další z elementárních tříd pro tvorbu uživatelského rozhraní. Každá renderovatelná *UIKit* komponenta musí dědit z této třídy.[20]

UIStackView

Je komponenta dědící ze třídy *UIView* zobrazující kolekci dalších vnořených *UIView* zobrazených vedle sebe a to buď vertikálně, nebo horizontálně.[18]

UILabel

UI komponenta zobrazující formátovaný text.[14]

UIImageView

Je komponenta dědící ze třídy *UIView*, která zobrazuje statické fotky či obrázky.[13]

UIControl

Je přímým potomkem třídy *UIView*, s robustním rozhraním pro události uživatelských interakcí. Definuje širokou škálu událostí pro dotyk a posun prstu pro danou komponentu.[12]

UIButton

Je potomek třídy *UIControl*, reprezentující klikatelné tlačítko.[10]

UIScrollView

UIScrollView je podtřída *UIView*. Je to komponenta, která může zobrazovat větší obsah než jsou jsou jeho rozměry. Uživatel se pak může posouvat v obsahu pomocí gest, jako posun, nebo zoom.[16]

UITableView

UITableView je podtřídou *UIScrollView*, která je zároveň její nejpoužívanější podtřídou. *UITableView* je uživatelská komponenta, která zobrazuje seznam buněk pod sebou. Tyto buňky musejí být typu *UITableViewCell*, která je podtřídou *UIView*. *UITableView* potom zobrazuje tyto buňky v obsahu *UIScrollView* líně, čímž dosahuje dobré paměťové a výpočetní náročnosti.[19]

UICollectionView

Je další instancí podtřídy *UIScrollView*, která, narozdíl od *UITableView*, zobrazuje svoje buňky *UICollectionViewCell* v mřížce namísto vertikálního seznamu.[11]

2.2 SwiftUI komponenty

View

View je protokol definující část uživatelského rozhraní pro SwiftUI. Tento protokol vyžaduje implementaci vlastnosti body, které obsahuje vlastní definici uživatelského rozhraní komponenty.[8]

VStack, HStack, ZStack

VStack, *HStack* a *ZStack* komponenta zobrazující kolekci vnořených pod-struktur implementující protokol *View*. *VStack* zobrazuje pod-struktury vertikálně, HStack je zobrazuje horizontálně a ZStack je zobrazuje nad sebou. [3]

GeometryReader

Je komponenta, která svému tělíčku předá strukturu typu *GeometryProxy* obsahující informace o svých rozměrech v rodičovské komponentě.[7]

Text

UI komponenta zobrazující jeden či více řádků textu. [9]

Button

Button je komponenta reprezentující klikatelné tlačítko. [6]

Kapitola 3

Analýza populárních aplikací

Každá aplikace je ve svém řešení uživatelského rozhraní pro danou uživatelskou funkci unikátní. Některá jsou efektivnější, než ostatní; některá prioritizují vlastní business logiku nad uživatelskou přívětivostí; některá přicházejí s vlastním řešením některých uživatelských problémů a některá přenášejí uživatelská paradigmata z jiných systémů.

Najdou se nicméně mezi těmito řešeními určité parallely, které bych se chtěl v této práci zaměřit, definovat a následně vytvořit vlastní implementaci takovýchto uživatelských komponent pro budoucí využití v mobilním vývoji.

Postup výběru aplikací

Pro sadu aplikací k analýze jsem si vybral pár aplikací, které v minulosti získaly ocenění Apple Design Awards, které Apple každý rok oceňuje na své vývojářské konferenci WWDC.

Popis vybraných aplikací

Pro analýzu jsem si vybral dvě velmi populární aplikace a to aplikace *Darkroom* - jednoduchý, ale mocný editor fotek a videí dostupnou pro iOS a iPadOS a aplikace *Calzy*, velmi pěkně vyřešenou aplikaci pro iOS a iPadOS.

Apple Design award 2020

- Darkroom: Photo & Video Editor[1]

Apple design awards app 2018

- Calzy[2]

3.1 Calzy

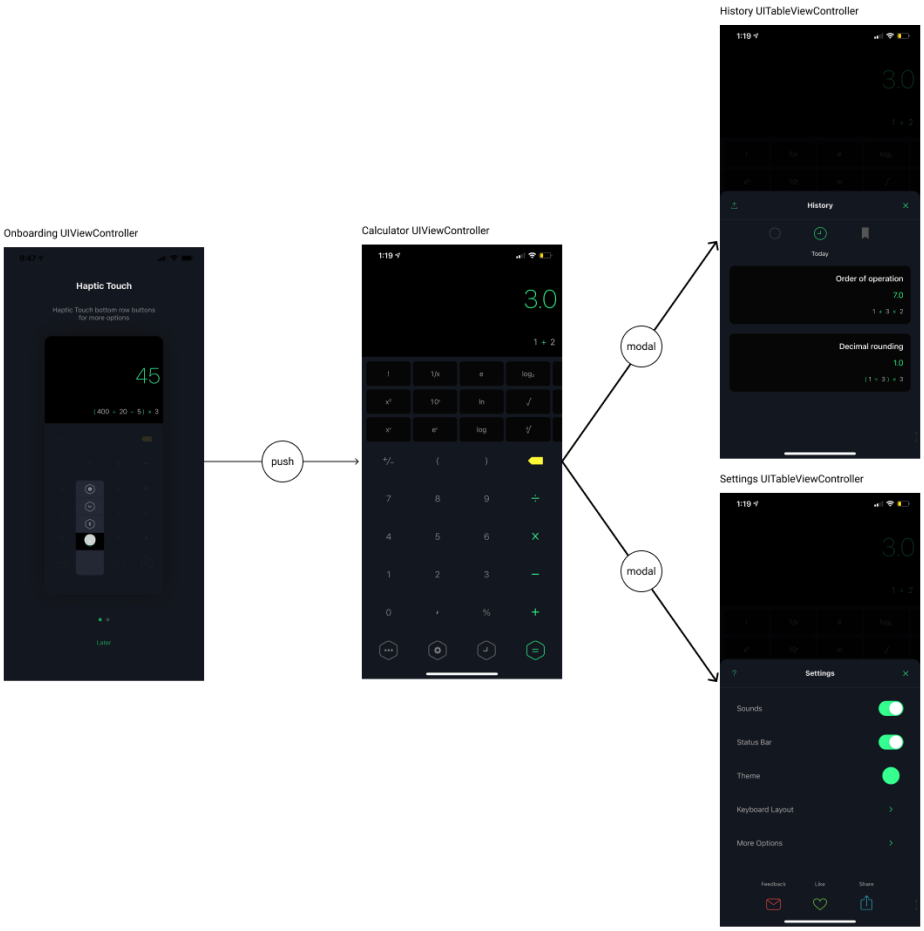
Calzy je krásná a jednoduchá kalkulačka, která se snaží nahradit vychozí kalkulačku na vašem iPhonu a iPadu, kde kalkulačka chybí úplně. Kromě běžných funkcí kalkulačky, které nemohou chybět žádné kalkulačce, tato aplikace obsahuje synchronizovanou historii výpočtů s iCloud, funkce *drag and drop*, *force touch* a haptické vibrace v celé aplikaci.

[1] https://apps.apple.com/us/app/darkroom-photo-video-editor/id953286746#?platform=ipad
[2] https://apps.apple.com/us/app/calzy-3/id623690732?ign-mpt=uo%3D4

3.1.1 Navigace

Při prvním spuštění aplikace vás *Calzy* přivítá *UIViewController* s onboaridngem, který vás provede základní navigací a funkcí *drag and drop*, poté *Calzy* přejde na hlavní obrazovku obsahující rozhraní kalkulačky. *UIViewController* s onboardingem se zobrazí pouze jednou, poté aplikace při dalších spuštěních již zobrazí scénu kalkulačky. *UIViewController* se scénou kalkulačky obsahuje hlavní funkce aplikace. Obsahuje numerickou klávesnici, číselný displej a custom tab bar navigačních tlačítek ve spodní části obrazovky, která dokonale ladí s numerickou klávesnicí.

Na scéně s kalkulačkou může uživatel modálně zobrazovat *UIViewController* s historií a *UINavigationController* s tokem nastavení. Kompletní storyboard (plán navigace mezi scénami) je zobrazen na obrázku 3.1



Obrázek 3.1: Calzy storyboard.

3.1.2 Dekompozice scén

Pojďme projít konkrétní scény v aplikaci *Calzy* a analyzovat jejich hierarchii, začínající s počáteční scénou kalkulačky, zabalenou v *UISplitViewControlleru*.

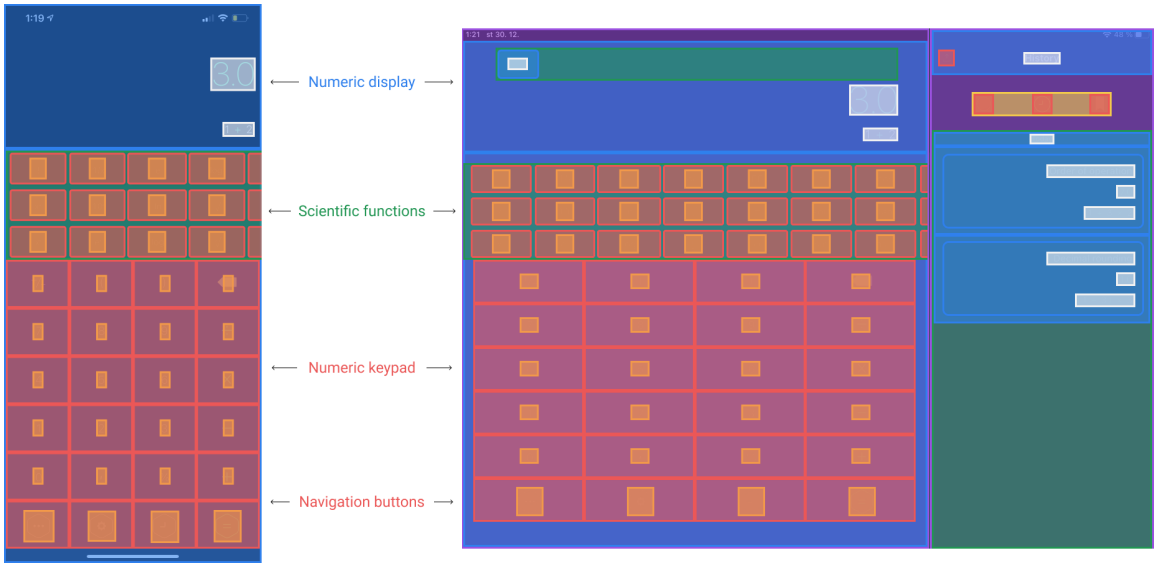
Main UISplitViewController

Calzy je aplikace vytvořená jak pro iOS, tak i pro iPadOS, a proto dává smysl, že jejím hlavním *UIViewControllerem* je *UISplitViewController* se scénou kalkulačky jako *Primary UIViewController* a scénou s historii výpočtů jako *Primary UIViewController*. *UISplitViewController* má vlastnost *primaryEdge* nastavenou na *.trailing*, takže *Primary UIViewController* s historií výpočtů je zobrazen na pravé straně obrazovky, když je prezentován na obrazovkách s *regular SizeClass*. Při kompaktním zobrazení se *UISplitViewController* rozpadne a zobrazuje *Primary UIViewController*, takže *UIViewController* s rozhraním kalkulačky je prezentován jako první *UIViewController*.

Scéna kalkulačky

UIViewController se scénou kalkulačky je rozdělen do čtyř částí:

1. Číselný display s počítaným výrazem a hodnotou tohoto výrazu
2. Posuvná klávesnice s vědeckými funkcemi
3. Číslicová klávesnice
4. Spodní řádek s navigačními tlačítky



Obrázek 3.2: Kalkulačka

Číselný displej může být tvořen *UIView* kontejnerem obsahujícím dva *UILabely* s textem výrazu a jeho vypočtenou hodnotou. Tento kontejner obsahuje v zobrazení pro iPad i posuvnou oblast, kam může uživatel přetahovat číselné výsledky pro pozdější použití. Tato oblast by se mohla implementovat dvěmi způsoby - může být tvořena z *UIScrollView* obsahující *UIStackView* s vlastními buňkami *UIView* nebo by mohla být tvořena *UICollectionView* s buňkami *UICollectionViewCell* s horizontálním směrem posunu.

Posuvná klávesnice s vědeckými funkcemi by se měla skládat z *UICollectionView* a *UICollectionViewCells*. Klávesnice by také mohla být implementována pomocí *UIScrollView*, obsahující horizontální *UIStackView* uvnitř vertikálního *UIStackView*. Takové řešení by ale

bylo značně neefektivní jelikož by struktura musela být statická a jakékoliv rozšíření by mohlo být složitější, než v případě implementace pomocí *UIScrollView*.

Numerická klávesnice je roztáhnutelná a neměnná tabulka tlačítek. Protože tato sekce nemusí být rolovatelná, může se skládat ze statického horizontálního *UIStackView* uvnitř vertikálního *UIStackView* obsahující tlačítka *UIButtons*.

Navigační tlačítka mohou být umístěna ve stejném nebo samostatném *UIStackView* pod tlačítky numerické klávesnice.

Navigační tlačítka reagují na gesto, které aktivuje zobrazení s více možnostmi. Uživatel může vybrat tyto možnosti tak, že na ně posune prst. Gesto se deaktivuje při události *touch up*, takže gesto vyberete dotykem a posunutím, což je originální přístup k řešení více možností.

Tlačítka používají force touch gesto na zařízeních s force touch displeji nebo používají gesto dlouhého podržení.

History UITableViewController

Smyslem tohoto *UIViewController* je zobrazit historii výpočtů uživatele či výpočty uložené v jeho záložkách. Uživatel se pak může dostat ke svým předchozím výpočtům, které může dále sdílet jako text s jinými aplikacemi prostřednictvím systémové Share extensiony.

Protože je tato obrazovka s největší pravděpodobností podtřídou *UITableViewController*, její struktura je vcelku jednoduchá. V horní části obrazovky je navigační lišta s tlačítkem sdílení na levé straně, titulem názvu obrazovky uprostřed a tlačítkem pro zrušení na pravé straně. Zbytek scény zabírá *UITableView* s buňkou *UITableViewCell* pro položky v seznamu, rozděleného do sekcí podle data a záhlavím se třemi přepínacími tlačítky implementovaného pomocí *UIView*.

UIView se záhlavím se v *UITableView* objeví jako první buňka bez opakování. Pravděpodobně obsahuje *UIStackView* tří tlačítek *UIButton*, která po události doteku změní obsah tabulky z historie na záložky a naopak. Třetí tlačítko způsobí, že buňky budou označitelné, jak je uvedeno níže. Tato buňka nahrazuje *UITabBarController*, který by byl pro tento případ použití přehnaný z hlediska efektivního využití paměti a výkonu.

Buňka s výpočtem se skládá ze tří textů *UILabel* zobrazené pod sebou. První *UILabel* zobrazuje (volitelný) název, druhý zobrazuje vypočtenou hodnotu výpočtu a třetí zobrazuje výraz výpočtu. Buňka výpočtu reaguje na gesto dlouhého stisknutí, které přináší list akcí s možnostmi sdílet výpočet jako text, upravit jeho název, použít výraz, hodnotu výrazu v aktuálním výpočtu na hlavní scéně, nebo přidat zámek k výrazu, který potom uživatel může odemknout pomocí biometrického zabezpečení, tedy pomocí FaceID či TouchID.

Buňku výpočtu lze také vybrat, když je vybráno tlačítko výběru v záhlaví. Uživatelé pak mohou sdílet vybrané buňky výpočtu prostřednictvím systémové Share extensiony.

Modal segues

Calzy v zobrazení pro iOS nepoužívá výchozí modální segue, místo toho *Calzy* navrhl svůj vlastní styl modální prezentace podobný komponentě nazývané BottomSheet, kterou můžete najít v aplikaci jako je např. Apple Maps. Na iPadOS prezentace modálních scén připomíná spíše styl popover. Obě zobrazení lze zavřít kliknutím mimo prezentující UIViewController.

3.2 DarkRoom

DarkRoom je aplikace na úpravu fotek a videí, která je určena nejen pro laické uživatele, ale i profesionály. *DarkRoom* umožňuje svým uživatelům transformovat své fotografie, nastavovat filtry, upravovat barvy a další. Každá úprava se v aplikaci aplikuje nedestruktivně, díky čemuž DarkRoom zvládne uchovávat celou historii úprav a vracení změn je pro něj velmi snadné.

DarkRoom je postavená na modelu předplatného, což znamená, že abyste mohli aplikaci používat, tak vám aplikace účtuje měsíční nebo roční poplatek. *DarkRoom* také nabízí možnost, která vám umožní mít aplikaci neomezeně dlouho, cena této možnosti je 4 krát vyšší než roční poplatek.

3.2.1 Navigace

Aplikace potřebuje pro své správné fungování souhlas od uživatele do knihovny obrázků. Pokud uživateli ještě nebyl zobrazen dialog pro přístup do uživatelské knihovny obrázků, aplikace po startu přechází do první scény *Permission UIViewController*, kde uživatel musí poskytnout aplikaci přístup do oné knihovny, jinak ho aplikace nepustí dál.

Po kliknutí na velké červené tlačítko se uživateli zobrazí systémový dialog pro přístup do uživatelské knihovny obrázků, kde má uživatel na výber ze tří možností - povolit přistup do célé knihovny obrázků, vybrat určité obrázky, nebo zakázat přístup. Dialog pro přístup k citlivým udájům a funkcím, se uživateli na iOS může zobrazit pouze jednou. Pokud uživatel zakáže aplikaci přístup, *Permission* scéna se změní a zobrazí pouze tlačítko s odkazem do systémového nastavení, kde uživatel může tyto přístupy dále upravit.

Poté, co uživatel udělí aplikaci přístup, přejde aplikace do scény *Finder UIViewController* s mřížkou uživatelských fotografií. Z *Finder UIViewController* může uživatel přidávat další fotografie ze své knihovny obrázků, zobrazit modálně *Nastavení UIViewController* a nebo vybrat jednu ze svých fotografií pro přechod na scénu *Edit UIViewController*.

V *Edit UIViewController* může uživatel upravovat své fotografie nebo videa, lajkovat je anebo je sdílet s ostatními prostřednictvím systémové *Share* extensiony.

3.2.2 Dekompozice scén

Finder UIViewController

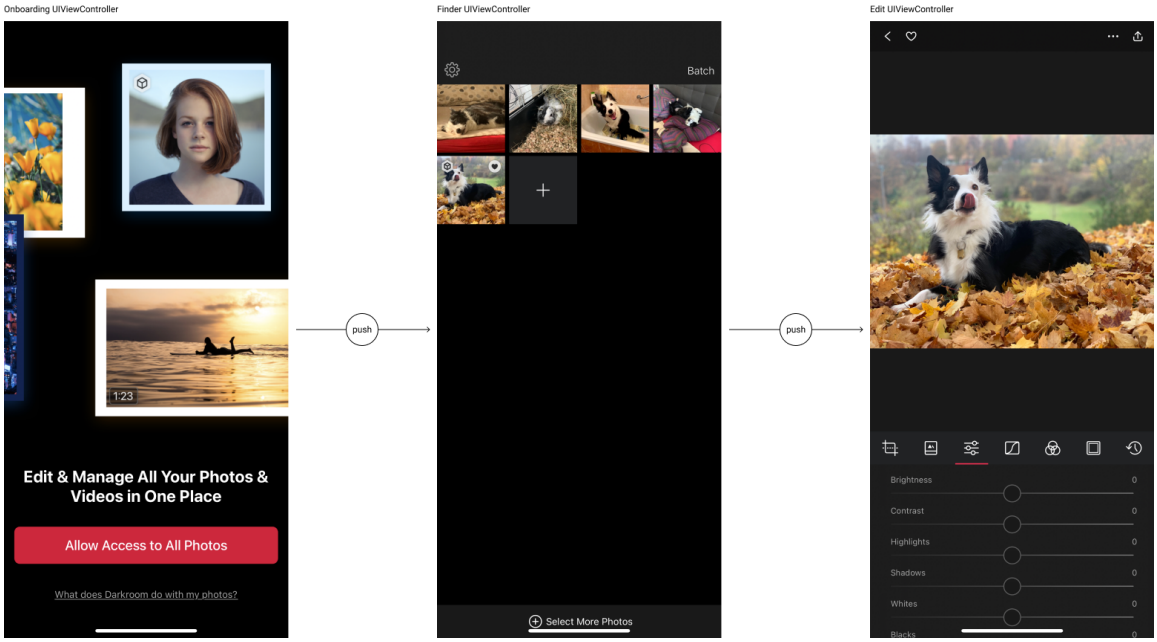
Finder UIViewController obsahuje horní lištu, *UICollectionView* s uživatelskými fotografiemi a spodní lištu.

Na horní liště je tlačítko, které přepíná vlastnost *contentMode UIImage* uvnitř buněk *UICollectionViewCell* mezi *.scaleAspectFit* a *.scaleAspectFill*. Hned vedle přepínacího tlačítka je tlačítko nastavení, které zobrazuje *Settings UINavigationController* obsahující celý navigační strom nastavení pro všechna nastavení.

Spodní lišta obsahuje pouze jedno tlačítko, které zobrazí uživatelskou knihovnu obrázků, kam může uživatel přidávat další fotografie do seznamu ve *Finder UIViewController*.

Zbytek *Finder UIViewController* zabírá *UICollectionView* s *UICollectionViewCell* obsahující *UIImageView* s fotografiemi uživatele.

Při výběru buňky aplikace přejde na *Edit UIViewController*.



Obrázek 3.3: Storyboard

Edit UIViewController

Edit UIViewController je hlavní scénou aplikace DarkRoom, je to obrazovka, kde probíhají uživatelské akce pro úpravu fotek a videí. *Edit UIViewController* zdůrazňuje nejdůležitější aspekt scény – obsah – tím, že roztahuje zobrazení obsahu až k okrajům obrazovky. Zbytek možností scény je schovaný ve dvou tenkých panelech po bocích.

Oba panely jsou si z hlediska chování velmi podobné – oba jsou výsuvné, oba mají okrajové i středové *UIStackView* obsahující talčítka *UIButton* a oba mohou prezentovat vysouvací detail vybrané možnosti.

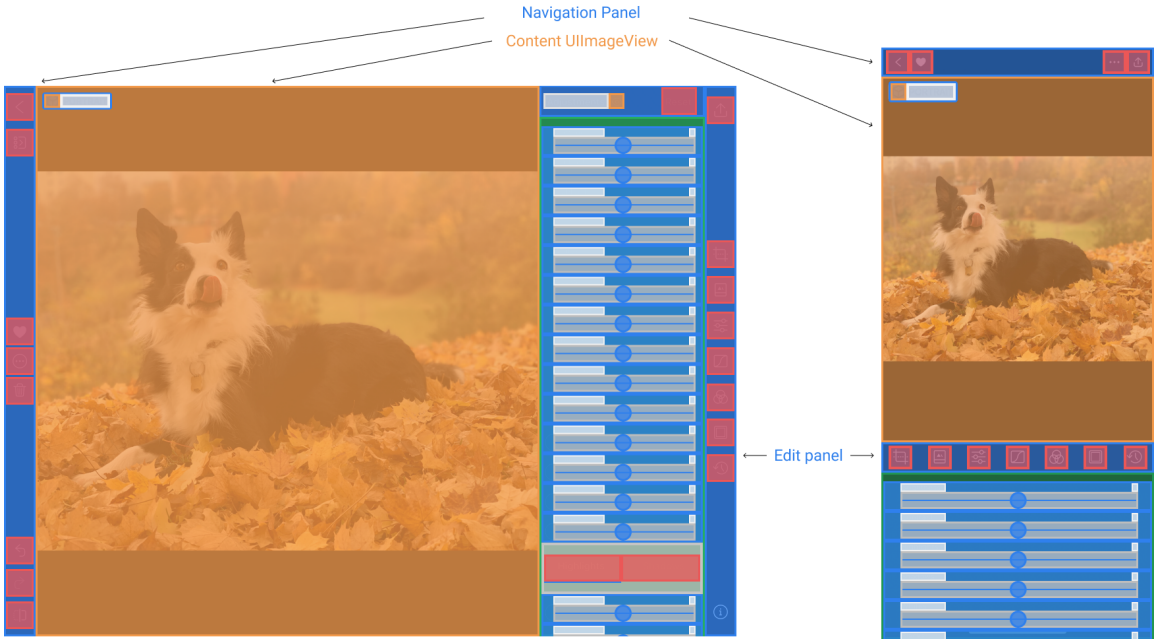
Panely na obrazovce s *regular SizeClass* se chovají jinak než na displejích s *compact SiceClass*.

Jeden z panelů je panel je pro navigaci. Navigační panel se zobrazuje na levé straně při zobrazení obrazovky na displeji s *regural SizeClass* a obsahuje tlačítko zpět, které uživatele vrátí na scénu Finder UIViewController. Vedle tlačítka zpět je tlačítko Preview, které na větších displejích zobrazí náhled obsahu z *UICollectionView* ve *Finderu*. Středová řada tlačítek obsahuje tlačítko Líbí se mi, Odstranit a tlačítko, které vyvolá *ActionSheet* s dalšími možnostmi pro skrytí histogramu, správu úprav a redundantní možností pro export, Líbí se mi a smazání aktuálně zobrazené fotografie. Na druhém okraji navigačního panelu jsou tlačítka vrátit akci a vrátit zpět vrácenou akci a tlačítko, které po dotyku zobrazí původní originální fotografii bez úprav.

Navigační panel je zobrazen vertikálně na displejích s *regular SizeClass* a horizontálně na *compact SizeClass*.

Druhý panel – editační panel – obsahuje prvky uživatelského rozhraní, které uživatel používá pro úpravu svých fotografií. Obsahuje detailní pohled na oříznutí, nastavení filtrů, úpravy, nástroj křivek histogramu, úpravy barevných kanálů a chronologickou historii změn.

Panel úprav je zobrazen vertikálně na displeji s *regural SizeClass* s výsuvným detailem, které se vysune vedle editačního panelu. Na displejích s *compact SizeClass* se panel zobrazí



Obrázek 3.4: Edit UIViewController

jako horizontální spodní výsuvný panel, který lze táhnout nahoru a dolů pomocí gesta posunu a odhalit tak detail pod editačním panelem.

Kapitola 4

Návrh sady komponent

Uložím pro výběr komponent pro téma této práce několik cílů.

Jednoduchost

Flexibilní povaha komponent uživatelského rozhraní definovaných knihovnou *UIKit* umožňuje vývojářům přizpůsobit si komponentu tak, aby vyhovovala potřebám projektu, a/nebo poskládat vlastní komponentu z více prvků a vytvořit tak zcela novou komponentu uživatelského rozhraní. Nemá tedy smysl soustředit se na komponenty, které lze implementovat během několika hodin. Místo toho se zaměřím na složitější komponenty, jejichž implementace vyžaduje kumulativně více času, díky čemuž je řešení vhodné pro použití v projektech s omezeným časovým budgetem. Taková komponenta pak ušetří čas vývojářů a umožní dokončit projekt bližším časovém termínu.

Přizpůsobitelnost

Komponenta uživatelského rozhraní musí dostatečně univerzální, aby se dala využít pro širokou škálu aplikací s různými designovými i návrhovými vzory. Pokud je účel komponenty uživatelského rozhraní vhodný pouze pro úzkou sadu aplikací, mělo by se na ni pohlížet jako na komponentu pro speciální účel, a proto takovou komponentu nepovažují za vhodnou pro účel této práce.

4.1 BottomSheet

BottomSheet je navigační komponenta podobná například *UISplitViewController*, která se skládá ze dvou scén – hlavní scény, zobrazení obvykle obsahujícího hlavní obsah a doplňkového překryvného zobrazení, které se obvykle používá jako doplněk k obsahu hlavního zobrazení.

4.1.1 Chování

BottomSheet je schopen ve svém rozložení prezentovat jakýkoli druh scén UIViewController, včetně dalších navigačních komponent, jako je *UINavigationController*. Toto umožní vývojářům, pracující s touto komponentou, mít úplnou kontrolu nad vzhledem prezentovaného obsahu.

4.1.2 Řízení

BottomSheet by měl reagovat na gesto posunu uživatele, díky čemuž se panel pohybuje nahoru nebo dolů plynulým pohybem. Když uživatel gesto pustí, spodní list by se měl animovaně ukotvit k nejbližšímu předdefinované koncovému bodu . Animace by měla být uživatelsky přerušitelná pro případ, že uživatel změní názor. Rozsah možných koncových bodů (pozic, ke kterým se může *BottomSheet* ukotvit) může být kdekoliv ve vertikálním intervalu uživatelského displeje.

4.1.3 Gesta

Algoritmus řešící gesta pro posun dolního panelu by si měl být vědom možného UIScrollView uvnitř obsahu v prezentovaném panelu. Pokud by měl UIScrollView posuvný obsah ve vertikálním směru a byl by na jednom z konců, měl by BottomSheet nabízet plynulý přechod mezi posouváním panelu a obsahem v UIScrollView. BottomSheet by dále neměl kolidovat s žádným jiným gestem prezentujícího UIViewControler.

4.1.4 Případy užití

BottomSheet září nejlépe při použití v kompozici uživatelského rozhraní se silným důrazem na obsah, což umožňuje vývojářům využít celou plátno celé scény naplno. Tyto typy aplikací zahrnují aplikace, jako jsou mapy nebo navigace, aplikace pro videohovory, fotogalerie, editory fotografií, vyhledávač fotoaparátu nebo dokonce audio aplikace (jako výsuvné ovládací prvky přehrávače).

4.2 DynamicContent

Jednou z repetitivních rutin vývojáře je implementovat muli-stavovou komponentu pro načitatelný obsah, která bude podporovat stav pro načítání dat, zobrazení obsahu, zobrazení prázdného obsahu a případně chybový stav. Načítatelný obsah je takový obsah, který není přítomen na zařízení v počátečním stavu a je třeba jej získat ze třetí strany. U načítatelným obsahu není získávání dat zaručenou operací. Tato situace způsobuje explozi možných stavů, a každý stav by měl být prezentován jinak.

Mezi tyto stavy mohou patřit následující příklady z předchozí analýzy - stav prázdného obsahu pro obsah historie uživatelských výpočtů (součást *History UITableViewController* v aplikaci Calzy), stav pro autorizaci aplikace pro soukromého obsahu (*Onboarding UIViewController* v aplikaci DarkRoom) nebo stav načítání a chyb pro jakýkoli obsah ke stažení z internetu. To se pro vývojáře ukazuje jako docela běžná vyčerpávající práce, která kumulativně vývojáři může zabrat až dny práce, a je to přesně tento problém, který se *DynamicContentView* snaží řešit.

4.2.1 Chování

DynamicContentView by měl mít uživatelsky definovatelnou sadu možných stavů. Komponenta by měla nabízet předdefinované systémové *UIViews* pro běžné dynamické stavy, včetně stavu načítání, prázdného stavu a chybového stavu, avšak by měla obsahovat volitelnou možnost si uživatelsky definovat vlastní *UIView* pro každý stav.

Použití komponenty by mělo být snadné, měla by zpřístupňovat pouze vlastnosti, které mají být veřejné, a vyžadovat implementaci svého delegáta pouze v případě, že je to ne-

zbytně nutné. *DynamicContentView* by mělo využívat nejnovější knihovny od Apple, jako je *Combine*.

Komponenta by měla být také paměťově efektivní, protože ji lze využít vícekrát na jedné scéně. *DynamicContentView* by měl instanci svých stavových *UIView* vytvářet pouze v případě potřeby, měl by je tedy instanciovat líně. Uvolňovat by je měl ihned poté, nebudou vyžadovány.

4.2.2 Případy užití

Každá aplikace, která nemá obsah v zařízení a potřebuje jej načíst z třetích stran a/nebo s obsahem, který může být prázdný. Například stav načítání a chybový stav pro načítání předplatných v DarkRoom, prázdný stav pro prázdný obsah historie minulých výpočtů uživatele, v Calzy's History *UITableViewController*, stav pro autorizaci aplikace pro obsah ochrany soukromí v Onboarding *UIViewController* aplikace DarkRoom.

Kapitola 5

Implementace UIKit

5.1 BottomSheet

BottomSheet pro *UIKit* je sada několika tříd, které umožní uživateli implementovat posuvný překryvný panel v projektu využívající *UIKit*. *BottomSheet* se skládá z následujících tříd a protokolů:

- *BottomSheetControlleru* - scéna, ve které se realizuje zobrazení dvou dalších pod-scén ve formě hlavního obsahu a posuvného panelu.
- *BottomSheetPosition* - Třída starající se o kategorizaci a nastavení stylu prezentace rozložení v BottomSheetControlleru.
- *BottomSheetAnchor* - Protokol pro implementaci datového modelu v podobě výčtového typu.
- *BottomSheetAnchorDelegate* - Protokol pro delegáta kontrolujícího chování posuvného panelu.

5.1.1 Implementace

BottomSheet je pro účel této implementace koncipován jako *Container UIViewController*, tedy scéna, která zaobaluje vícero dalších pod-scén (tzv. *Child View Controllery*[4]). Tato dekompozice dává *BottomSheet* komponentě několik výhod.

Díky tomuto zapouzdření získává *BottomSheet* úplnou kontrolu nad prezentací výsledného rozložení scén na displeji v rámci svého ohraničení. Toto umožní *BottomSheetControlleru* zobrazovat scény volně v prostoru svojí scény. Tímto *BottomSheet* docílí oddělení logiky prezentace pro posuvný panel a umožní abstrahovat definici API rozhraní operací nad posuvným panelem.

Další výhodou této kompozice je flexibilnost využití scén definovaných uživatelem knihovny. *BottomSheet* díky tomu vyžaduje splnit pouze jednu specifikaci, a to aby scéna byla typu *UIViewController*. Veškeré další chování *BottomSheetu* je dále nastaveno dle výchozích hodnot této knihovny. Uživatel díky tomu získává kontrolu nad veškerým prezentovaným obsahem v *BottomSheetu* a umožňuje mu svůj obsah dále zaobalit do dalších *UIViewControllerů*, jako je např. *UINavigationController* používaný pro strukturovanou navigaci. *BottomSheet*, pro úplnost, umožňuje bližší specifikaci výchozích hodnot implementací *BottomSheetAnchorDelegate*, či nastavení instančních vlastností přímo v *BottomSheetControlleru*.

5.1.2 Struktura BottomSheetControlleru

BottomSheetController je *UIViewController* obsahující dvě pod-scény nazývané v kontextu knihovny jako *master* a *overlay*. *Master* je scéna obsahující primární obsah, která je v *BottomSheetu* roztáhlá do všech okrajů, ignorující pomocné odsazení *safe area*. *Overlay* je potom scéně zobrazující se jako posuvný panel na spodu obrazovky.

5.1.3 Renderování zobrazovaných scén

BottomSheet počítá se dvěma přístupy implementace, které uživatelé knihovny mohou použít. Pomocí konstruktoru a využitím storyboardu. Ačkoliv *BottomSheet* podporuje tyto dvě možnosti instanciování, implementuje rozložení svoji vnitřní struktury striktně v kódu za použití *NSLayoutConstraints*, využívá tedy technologie *AutoLayout*.

Pokud uživatel zvolí instanciovat BottomSheet ručně pomocí konstruktoru, uživatel sám instancuje BottomSheetController, kterému předá referenci na master a overlay scénu. BottomSheet v takovém případě zavolá vnitřní metody *showMaster(_:)* a *showBottomSheet(_:,_:,_:)*, které korektně registrují Child View Controller pomocí metod *addChild(_:)* a *didMove(_:)*, které jsou popsané v dokumentaci od Apple o tom jak správně přidávat Child View Controllery[3].

Druhá možnost pro uživatele knihovny je využít storyboard. Při použití storyboardu uživatel definuje scénu, dědící ze třídy BottomSheetController, přiřadí mu dva přechody (segueu) na scény. Tyto přechody musejí být typu *Show detail* a musejí obsahovat identifikátory *master* nebo *overlay*. Příklad kompozice storaboardu zobrazuje obrázek 5.1. *BottomSheetController* po instanciování zkontroluje v metodě *loadView()* zda byl sám instanciován pomicí storyboardu tím, že zkontroluje instanční vlastnost "storyboard". Pokud tato vlastnost není null, zavolá metodu *performSegueu(_:,_:)* s identifikátory pro master a overlay, která tyto scény instanciuje. BottomSheet poté v metodě ‘prepare(for segue:)‘, která předá BottomSheetu referenci na instance pod-scén, naváže na postup pro instancování pomocí kódu. Tedy zavolá metody *showMaster(_:)* a *showBottomSheet(_:,_:,_:)*.

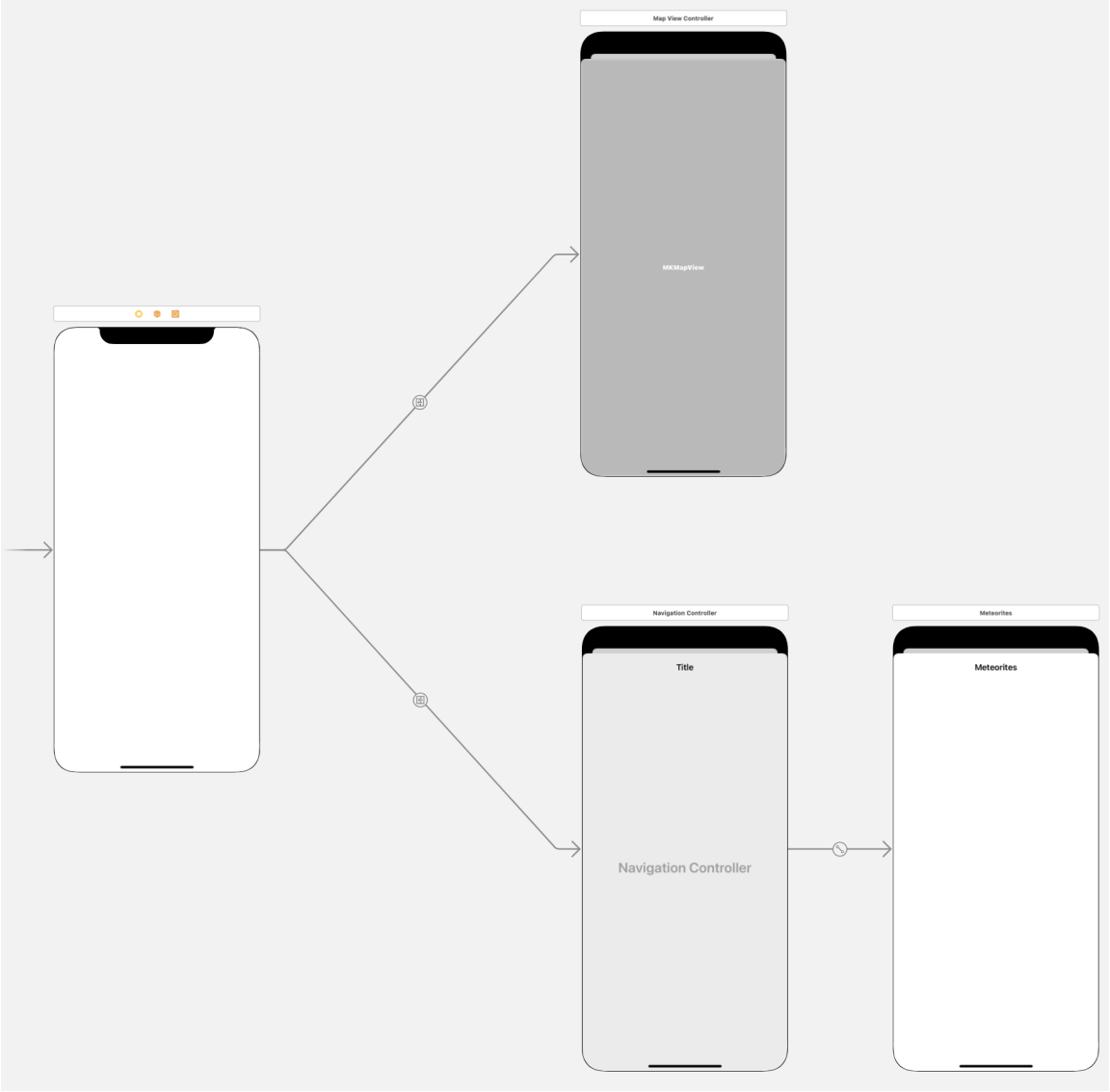
5.1.4 Chování posuvného panelu

Posuvný panel se scénou overlay je po prvním zobrazení scény zobrazeno schovaně na spodu BottomSheetControlleru. Toto chování je implementováno úmyslně, aby si jej uživatel knihovny mohl upravit dle svých potřeb.

Uživatel může gesty tento panel posouvat po vertikální ose nahoru a dolu, přičemž nejníže může uživatel panel posunout na spodní odsazení s hodnotou 0, kdy je panel ve stavu *schovaný*, a nejvýše do úrovně vertikální velikosti obsahu scény overlay.

Posuvný panel má uživatelsky definované mezi-úrovně, do kterých se může panel ukotvit.

Chování posouvání panelu jsem implementoval podle techniky, prezentované na online přednášce celosvětové vývojářské konference pořádané firmou Apple z roku 2018, pojmenované *Designing Fluid Interfaces*, tedy navrhování fluidního rozhraní[5]. V této přednášce Chan Karunamuni definuje fluidní rozhraní jako nástroj rozšiřující mysl člověka, tedy nástroj, který dokáže odhadnout úmysl uživatele na základě mikrogest, které uživatel může použít. Tento velmi zajímavý pohled navrhování interakcí dále na přednášce prezentoval Nathan de Vries na příkladu implementace dynamického pohybu komponenty Picture in Picture, kde popisuje uživatelskou interakci pro přesun okna Picture in Picture mezi záchytnými body. Princip této interakce spočívá ve výpočtu konečné pozice okna. Když začne uživatel přesouvat okno, udělá většinou hbitý pohyb směrem ke koncovému bodu, kam chce



Obrázek 5.1: Implementace BottomSheetControlleru ve storyboardu

okno přesunout. Nathan de Vries v této části prezentace sdílí formuli pro výpočet projektovaného konečného bodu v prostoru zohledňující počáteční rychlost a konstantní lineární zpomalení. V jeho prezentaci poté vypočítali nejbližší koncový bod od projektované koncové pozice, kam okno Picture in Picture animovaně přesunuli.

$$projectedPosition = \frac{initialVelocity}{1000} * \frac{decelerationRate}{1 - decelerationRate} \tag{5.1}$$

Výsledkem je systém, který dokáže na základě uživatelského pohybu určit výsledný koncový bod využívající úmyslu uživatele. Tento systém je mnohonásobně lepší, než například systém, kde se uživatel musí přesunout objekt přes určitou vzdálenost (threshold), který může být při nesprávném zvolení hodnoty pro threshold, náchylný na chyby.

Tuto techniku jsem, po vzoru přednášky navrhování fluidního rozhraní, aplikoval pro vertikální posouvání posuvného panelu *BottomSheetu*, kdy *BottomSheet* vyhledá vertikálně

projektovanou pozici a poté *BottomSheet* vybere od té pozice nejbližší koncový bod pro ukotvení.

5.1.5 Gesta

BottomSheet implementuje pouze jedno vlastní gesto, nazývané v kontextu knihovny jako *slideGesture*, které se využívá pro posun posuvného panelu. Toto gesto je implementované pomocí UIPanGestureRecognizer, které umožňuje sekvenční sledování uživatelského dotyku začínající v kontejneru UIView obsahující scénu overlay.

SlideGesture je natvrdo nastavené tak, aby nekolidovalo s gesty jiného typu, jako jsou *UITapGestureRecoginer*, *UISwipeGestureRecognizer* a další. SlideGesture může kolidovat poze s dalším gestem typu *UIPanGestureRecognizer* a to pouze za podmínky, že je druhé gesto součástí komponenty *UIScrollView* a uživatel jej dopředu registroval do komponenty pomocí metody *func registerScrollViewDelegate(scrollView: UIScrollView)*.

Provázání s UIScrollView

Aby bylo možné provázat *slideGesture* s kompnentou *UIScrollView* (a tedy i její podtřídy *UITableView* či *UICollectionView*), BottomSheetController dokáže po registraci naslouchat gestu *UIPanGestureRecognizer* používané v této komponentě. Pokud uživatel začne gesto v hranicích komponenty *UIScrollView*, BottomSheet svoje *SlideGesture* gesto po dobu přijímání událostí UIScrollView deaktivuje. Pokud se uživatel dostane k při posunu k vertikálním hranicím obsahu *UIScrollView*, *BottomSheet* zakáže posun obsahu v *UIScrollView* a plynule naváže pohyb uživatele na posun panelu.

5.1.6 DataSource

BottomSheet potřebuje ke svému správnému fungování množinu bodů definujících úrovně odsazení, ke kterým se může panel ukotvit. Úrovně odsazení si knihovna definuje pomocí výčtového typu *BottomSheetOffset*, který definuje dva způsoby definice - relativní a specifický.

Specifická možnost výčtového typu *BottomSheetOffset* definuje své odsazení v bodech od spodní části scény *BottomSheetControlleru*. Tato možnost se hodí, pokud uživatel knihovny zná přesné velikosti zobrazovaného obsahu a chce mít kontrolu nad přesným zobrazením velikosti panelu v posuvném panelu. Tento typ odsazení nicméně nepokrývá všechny případy použití, které by uživatelé mohli potřebovat, jako například přesné zobrazení v půlce displeje, které na různých zařízení má různé specifické hodnoty odsazení.

Relativní datový typ *BottomSheetOffset* definuje úroveň odsazení relativně k displeji, vyžaduje tedy specifikovat procentuální hodnotu v intervalu od nuly do jedné, kde procentuální hodnota 0, představuje schovaný stav posuvného panelu (stejně jako specifická hodnota 0), hodnota 0.5 představuje odsazení v polovině displeje a hodnota 1, která definuje zobrazení panelu přes celou obrazovku. Hodnoty mimo tento interval *BottomSheet* přemapuje buď na hodnotu 0, nebo 1, podle toho, kterému konci tohoto intervalu má daná hodnota blíže. Relativní datový typ podporuje volitelné bodové odsazení od své relativní úrovně.

Pro předání množiny úrovní odsazení se používá protocol *BottomSheetAnchorDelegate*, který stačí implementovat do scény overlay v posuvném panelu. Předání reference na tohoto delegáta není potřeba, jelikož si *BottomSheetController* již referenci na tuto scénu

má a dokáže si ji najít, dokonce i kdyby byla scéna vnořená v dalším *UIViewController*. BottomSheet totiž prochází celou strukturu *UIViewControllerů* v hierarchii overlay.

Pokud uživatel nespecifikuje svého delegáta typu *BottomSheetAnchorDelegate*, BottomSheetController využije výchozí množinu definovanou v typu *BottomSheetDefaultAnchor*.

První iterace

Při první iteraci implementace předávání množiny úrovní odsazení předával *BottomSheetAnchorDelegate BottomSheetControlleru* v podobě seřazeného pole. Tento přístup byl dostatečný, ale nechal prostor pro nekonzistentní stav. Popíši připad užití, kdy by k nekonzistentnímu stavu mohlo dojít.

Uvažujme, že si uživatel knihovny předovlil dvě úrovně odsazení *.specific(offset: 200)* a *.relative(percentage: 0.6)*. Knihovna správně převezme tyto dvě hodnoty a ukotví panel k jedné z nich. Problém nastává v případě, kdy uživatel programově zavolá metodu pro manuální změnu úrovně voláním metody "setOffset(_:)". V tento moment nemá knihovna žádný způsob, kterým by explicitně uživatelovi knihovny omezila výběr možností na specifikovanou hodnotu z množinu datového modelu. Jinými slovy si uživatel uživatel mohl vybrat jakoukoliv hodnotu mimo rozsah uvedený v datovém modelu. Toto způsobilo zvláštní chování při uživatelském gestu pro posunu, kdy uživatel posunul panel z úrovně, na kterou se již nemohl dostat.

Druhá iterace

Abych tento případ užití vyřešil, bylo zapotřebí referencovat hodnoty v poli datového modelu v metodě setOffset(_:). Pro tento účel jsem vytvořil protokol pro výčtový typ nazývaný v kontextu knihovny jako *BottomSheetAnchor*. V tomto typu si uživatel knihovny definuje výčet pojmenovaných hodnot, na které pak namapuje konkrétní úrovně *BottomSheetOffset*.

Aby bylo možné vynutit typovou kontrolu, přidal jsem asociovaný typ implementující nově definovaný protokol *BottomSheetAnchor* do protokolu *BottomSheetAnchorDelegate*. *BottomSheetAnchorDelegate* poté přidává objektu, který tento protokol implementuje vlastnost *hook*, vytvářející generický wrapper, který poskytuje silně typované metody pro manuální nastavení úrovně odsazení posuvného panelu.

Toto řešení má i další výhodu. Když bude uživatel přidávat novou úroveň, bude mu stačit vědět, že ji stačí přidat do výčtového typu *BottomSheetAnchor*. Díky typové kontrole kompilátor uživatele upozorní na všechny části kódu, kde se využívá příkaz *switch*, které potřebují upravit.

Ačkoliv se tato druhá iterace zdá téměř ideální, v praxi tento přístup nutí uživatele vytvářet redundantní datové typy s velmi malými rozdíly. Tento problém jsem adresoval v poslední části této práce, věnující se knihovně InheritableEnum.

5.1.7 Podpora zobrazení pro iPad

BottomSheet pro *UIKit* navíc ke své specifikaci implementuje podporu zobrazení pro regular SizeClass, díky čemuž je schopný prezentovat obsah posuvného panelu na zařízení s malým, tak i velkým displejem. Pro zobrazení na displejích s velkým formátem, zobrazí *BottomSheet* posuvný panel na boční straně. Výsledné zobrazení tak připomíná zobrazení komponenty *UISplitViewController* s tím rozdílem, že je panel zobrazený nad primárním obsahem, namísto vedle něj.

5.2 DynamicContent

Knihovna *DynamicContent* funguje jako *wrapper* jednoho konkrétního primárního obsahu, který může přepínat mezi tímto primárním obsahem a dalšími zobrazeními, které jsou typicky definované pro celý projekt.

Knihovna se skládá ze dvou částí. První část je základní implementace stejnojmenné komponenty *DynamicContent* skládající z generické třídy UIView a protokolů pro datový model. Druhá část se skládá z výchozí implementace často používaných systémových komponent *UICollectionView*, *UITableView*, *UIImageView* a *UIStackView* využívajících tříd z první části.

Uživatel knihovny tak může v uživatelském rozhraní ve svém projektu použít výchozí implementace pro rychlejší a snazší použití, nebo si může implementovat vlastní komponenty využívající generické třídy *DynamicContent*.

5.2.1 Implementace

Generická třída *DynamicContent* obsahuje dva generické typy, potřebné pro správné fungování této třídy. Generický typ *ConfigurationState: DynamicContentState* obsahující datový model a generický typ *ContentViewType: UIView*, který specifikuje třídu *UIView* definující referenci pro primární obsah.

Uživatel si definuje datový model pro *DynamicContent* nad výčtovým typem, kde implementuje protokol *DynamicContentState*. Tento model bude obsahovat množinu stavů, které chce uživatel ve svém uživatelském prostředí podporovat. Tato definice je obecná a může obsahovat jakékoliv stavy, které uživatel může potřebovat.

Ve výchozí implementaci tato knihovna podporuje 4 nejčastější stavy, se kterými se běžně setkávám při vývoji aplikací pro iOS. Jsou to stavy pro načítání, chybový stav, stav pro prázdný obsah a stav pro primární obsah.

Uživatel pak definici pro tento datový model specifikuje jako generických typ *ConfigurationState* ve třídě *DynamicContent*, druhý generický typ *ContentViewType* specifikuje třídu dědící z UIView obsahující primární obsah komponenty.

5.2.2 Memory management

Komponenta *DynamicContent* klade velký důraz na optimalizaci výpočetní náročnosti. Vzhledem k tomu, že stavový prostor této komponenty není nijak omezen, mohlo by při vícenásobném použití generické třídy *DynamicContent* v jedné scéně dojít k drahému načítání všech objektů *UIView*, čímž by se zbytečně prodloužilo čekání uživatele na načtení příslušné scény.

Abych tento problém vyřešil, implementoval jsem do této knihovny načítání objektů *UIView* líně. Generická třída *DynamicContent* tedy načítá objekty jenom v případě, že že si uživatel vyžádá zobrazení konkrétního stavu pro příslušný objekt. Tímto způsobem komponenta docílí, že při prvotním vykreslení nedojde ke zpomalení načítání scény. Komponenta si navíc drží již načtené instance v paměti, čímž se snaží zamezit redundantnímu instancování tříd *UIView* pro stavy, které se budou za běhu programu opakovat.

5.2.3 DataSource

DynamicContent potřebuje ke svému správnému fungování datový model specifikující množinu stavů s referencí na příslušné instance třídy *UIView*. Datová struktura pro *Dynamic-*

Content je realizována pomocí výčtového typu *enum*, ve kterém si uživatel definuje výčet stavů pro příslušný dynamický obsah.

Použitím výčtového typu jsem docílil řešení obdobného problému s referencí, jako u komponenty *BottomSheet*.

V datovém modelu je uživatel povinen implementovat protokol *DynamicContentState*, který obsahuje povinnou metodu *func instantializeView(contentView: UIView) -> UIView* pro mapování příslušného stavu na příslušnou instanci třídy *UIView*, kterou bude v komponenta *DynamicContent* volat za běhu programu pro líné instancování objektů *UIView*.

Aby uživatel knihovny mohl v této mapovací metodě použít příkaz *switch* pro všechny definované stavy, přidal jsem do této metody argument *contentView* obsahující uživatelův primární obsah. Uživatel pak tímto způsobem naváže tuto instanci na příslušný stav pro primární obsah.

Kapitola 6

Implementace SwiftUI

6.1 BottomSheet

BottomSheet pro *SwiftUI* implementuje stejnou komponentu uživatelského rozhraní jako u *BottomSheetu* pro *UIKit*. *SwiftUI* komponenta přejímá od své sesterské knihovny terminologii a datový model *BottomSheetAnchor*, implementace části uživatelského rozhraní je ale od začátku nová a využívá přístupu deklarativní knihovny *SwiftUI*.

6.1.1 Implementace

BottomSheet se skládá ze tří struktur *View* - *BottomSheet*, která zaobaluje scénu podobně jako *BottomSheetController* v knihovně pro *UIKit*, *Sheet*, instance struktury *View*, která představuje posuvný panel a protokol pro datový model *BottomSheetAnchor*, který je převzatý z knihovny pro *UIKit*. Komponenty *BottomSheet* a *Sheet* pro oddělení business logiky a logiky pro renderování definují své vlastní modely *BottomSheetModel* a *SheetModel*.

BottomSheet implementuje svoji hierarchii pomocí komponenty *ZStack*, která umožňuje zobrazení komponent v ose z, čímž docílí komponenta *BottomSheet* zobrazení primárního obsahu a posuvného panelu přes sebe. *ZStack* ale sám o sobě pro implementací nestačí. *BottomSheet* potřebuje mít informaci o velikosti svého okna, aby mohl rozhodovat o intervalu posunu v rámci zobrazované scény. Pro tento účel *BottomSheet* navíc zaobaluje *ZStack* v komponentě *GeometryReader*, která svému tělíčku předá strukturu typu *GeometryProxy* obsahující informace o svých preferovaných rozměrech v rodičovské komponentě[7]. *BottomSheet* potom při vykreslení předá informaci o souřadnicovém prostoru celé scény *BottomSheet* svému modelu.

Sheet je implementovaný jako wrapper uživatelského obsahu. Úkolem tohoto wrapperu je vypočítat rozměry renderovaného obsahu a předat tuto informaci svému modelu. Tímto komponenta *BottomSheet* docílí, že společně s informací o rozměrech jejího matčiného kontejneru bude model vědět o všech potřebných rozměrech potřebných pro definici validního intervalu posuvného panelu pro gesto posunu.

6.1.2 Gesta

Implementace logiky pro posun posuvného panelu je definovaná ve třídě view modelu *BottomSheetModel*. Tento model instanciuje strukturu *DragGesture*, reprezentující gesto tažení, s odkazem na metody *slide(gesture:)* a *endSlide(gesture:)*, které gesto volá sekvenčně stejně při uživatelské interakci, jako je tomu u *UIPanGestureRecognizer* z implementace knihovny

pro *UIKit*. Toto gesto potom struktura *BottomSheet* vloží do struktury *Sheet* pomocí operátoru *.gesture(_: Gesture)*.

Tažení a ukončení gesta je z pohledu funkčnosti implementováno ekvivalentně s implementací v knihovně pro *UIKit*. Formule 5.1, kterou jsem použil pro nalezení koncového bodu v implementaci komponenty *BottomSheet* pro "UIKit, je již ve *SwiftUI* součástí gesta *DragGesture.Value* a tudíž již nebylo nutné tuto funkcionalitu implementovat znova.

Při pokusu implementovat provázání gesta tažení obsahu v *Listu* a gesta pro *Sheet*, jsem nenašel žádný způsob, které by se dalo pro tento účel využít. Zkoušel jsem tuto funkci implementovat odposloucháváním aktuálního odsazení obsahu v listu, ale při uživatelském tažení až k hranicím obsahu, se odsazení přestane aktualizovat a tudíž tento přístup nebylo možné použít. *SwiftUI* dále neposkytuje žádný operátor, který zpřístupnil interní gesto používané uvnitř komponenty *List*.

6.2 Dynamic Content

DynamicContent pro *SwiftUI* implementuje stejnojmennou komponentu implementovanou v kapitole implementace komponent pro *UIKit*. Varianta pro SwiftUI přejímá z knihovny pro *UIKit* pouze datový model.

Případ implementace ekvivalentní funkcionální parity pro knihovnu *DynamicContent* bylo pro *SwiftUI* velmi jednoduché. Výsledná implementace se vleze pouze do dvou datových typů - protokolu pro datový model *ContentState*, převzaté z knihovny pro *UIKit* a struktury *DynamicContent* implementující protokol *View*

DynamicContent si definuje dva generické typy - *ContentStateType*, obsahující typ datového modelu, který musí implementovat protokol *ContentState* a *ContentViewType*, který reprezentuje uživatelský obsah, čímž docílí flexibilní imeplementace umožňující zapouzdřit jakýkoliv primární obsah, který si uživatel definuje.

Tím, že tato komponenta implementuje pouze uživatelské rozhraní, je implementace pomocí *SwiftUI* velmi efektivní. Tělíčko definující obsah struktury *DynamicContent* obsahuje pouze volání metody "func viewForState(or content: AnyView) -> some View" obsaženou v datovém modelu.

6.3 Rozdíly přístupu implementací SwiftUI a UIKit

SwiftUI je velmi mocná knihovna umožňující velmi rychlou implementaci uživatelského rozhraní, u které je zapotřebí minimum kódu, a která umožňuje jednoduchou dekompozici komponent do vlastních zapouzdřených celků. Díky operátorům je možné výslednou hierarchii stuktur dále deklarativně upravovat bez nutnosti znát chování postupu vykreslování výsledného uživatelského rozhraní.

Vývoj knihoven ve *SwiftUI* byl přibližně daleko efektivnější a potřeboval k implementaci ekvivalentních funkcí méně než polovinu řádků kódu.

Komponenta	SwiftUI (počet řádků kódu)	UIKit (počet řádků kódu)
BottomSheet	310	961
DynamicContent	62	130

Tabulka 6.1: Tabulka s počty řádků implementace ekvivalentních funkcí

Deklarativní podstata *SwiftUI* spojená s jednoduchou syntaxí a faktem, že je SwiftUI dostupný relativně krátkou chvíli, se ale promítl ve flexibilitě této knihovny. *SwiftUI* stále nepokrývá všechny případy užití (zejména pak těch nestandardních) vývoje aplikací pro iOS, iPadOS a macOS, jako je např. implementace provázání gest pro posuvný panel a vnořeného scrollovatelného *Listu* v knihovně *BottomSheet*.

Po třech letech je, dle mého uvážení, *SwiftUI* natolik zralý, že se stává, díky jeho efektivnosti, dostatečným nástrojem pro tvorbu jednoduchých až středně pokročilých aplikací v agilním vývoji produkčních řešení.

Kapitola 7

InheritableEnum

Knihovna *InheritableEnum* umožnuje svým uživatelům implementovat dědičnost nad výčtovými typy pomocí protokolu *Inheritable*, čímž umožní rozšíření výčtových hodnot o hodnoty děděných výčtových typů.

7.1 Limitace jazyku swift

Jazyk Swift je multi-paradigmatický jazyk, implementující paradigmata pro objektově orientované programování[1], protokolově orientované programování[22] a součásti vzoru pro funkcionálně orientované programování[1].

Jazyk swift v rámci objektově orientovaného paradigmatu definuje třídy jako referenční datový typ umožňující zapouzdřit a abstrahovat vlastní implementaci, implementovat dědičnost a polymorfismus. Jazyk Swift dále implementuje hodnotové datové typy, které se za běhu programu předávají jako hodnoty v paměti, nikoliv jako ukazatel na část v paměti jako je tomu u referenčních datových typů. Instance referenčních a hodnotových datových typů si jsou v jazyce swift ve smyslu funkcionalit velmi podobné. Oba tyto typy mohou obsahovat vlastnosti, metody, mohou implementovat protokoly a tak podobně. Hodnotové datové typy jsou v jazyce Swift reprezentovány dvěma datovými typy - strukturami a výčtovými typy. Obecně se doporučuje při vytváření nových typů začínat se strukturou, tedy hodnotovým datovým typem a podle potřeby případně změnit strukturu na třídu, tedy na referenční datový typ.

Jsou tu ale rozdíly, které omezují použití hodnotových datových typů. Mám na mysli dědičnost, která je součástí pouze referenčních datových typu. V rámci této práce jsem se rozhodl implementovat dědičnost nad výčtovými datovými typy. Knihovna InheritableEnum je nicméně aplikovatelná na všechny hodnotové datové typy, ale s určitými limitacemi.

7.2 Motivace

V průběhu prace na implementaci sady komponent pro UIKit jsem narazil na zajímavý případ užití výčtových typů. Použití výčtových typu při implementaci obout komponent umožnilo silně typovanou referencovatelnou definici stavů, která je uživatelsky velmi přívětivá, a která při rozšíření těchto stavů umožní kompilátoru upozornit uživatele na nekompletní implementace jím definovaných typů.

V praktické imlementaci těchto knihoven do projektů, ale díky absenci dědičnosti hodnotových datových typů docházelo k redundantní definici stavů s velmi malými rozdíly.

Tento přístup nechává uživateli možnost implementovat redundantní stavy s nekonzistentními hodnotami.

Dědičnost nad výčtovými typy tento problém řeší a umožní uživateli definovat pro redundantní skupinu stavů referencovatelný vlastní datový typ, ze kterého potom uživatel může dědit.

7.3 Popis implementace

Implementace dědičnosti je psána využitím protokolově orientovaného paradigmatu jazyka Swift. Knihovna *InheritableEnum* tedy abstrahuje dědičnost nad výčtovými typy pouze za použití protokolů a struktur. Implementaci dědičnosti nad výčtovými typy je postavená na dvou nápadech - zřetězení dědičnosti do struktury *Chain*, vytvářející lineárně provázaný řetězec dědících typů, a funkcionalitě jazyku Swift pro přesměrovat volání vlastností *dynamic member lookup*, která umožňuje silně typovou kontrolu pro práci se zapisováním hodnot výčtových typů.

Základním protokolem pro využití dědičnosti je protokol *Inheritable*. Tento protokol obsahuje asociovaný typ *Inherites*, ze kterého chce uživatel knihovny dědit. *Inheritable* si definuje další dva asociované typy pro definici řetězce - *ChainType*, označující aktuální článek řetězce a *NextWrapper*, označující následující článek v řetězci.

Články mohou být dvojího typu - *Chaining* nebo *Wrapping*. Tyto dvě struktury, implementující protokol stejný *TypeWrapper*, mají různé účely. *Wrapping* je článek obsahující pouze jeden generický typ a zaobaluje tedy jeden dědící typ, většinou řetězcový terminátor *NoneType*. Řetězcový článek *Chaining* naproti tomu obsahuje generické typy dva - jeden pro dědící typ, a druhý pro následující matčinný typ *Chaining* či *Wrapping*.

Pokud výslednou implementaci implementujeme nad volitelným počtem výčtových typů, kompilátor v dokáže syntetizovat výsledný typ řetězce v asociovaném typu *ChainType* v protokolu *Inheritable*.

Pro instanciaci řetězce jsem rozšířil protokol *Inheritable* o implementací statické metody *static func chain(with delegate: ChainingDelegate) -> ChainType*, která zvládne její asociovaný typ *ChainType* instanciovat. Tato metoda očekává jeden parametr typu *ChainingDelegate*, který bude dostávat notifikace o přiřazených hodnotách.

Výsledný řetězec obsahuje implementaci *@dynamicMemeberLookup*, který implementuje generickou metodu *subscript<L>(dynamicMember keyPath: WritableKeyPath<NextWrapperType, L>) -> L*. Tato metoda umožní směrovat volání getter a setter metod pro všechny vlastnosti z podřetězců do matčiného článku, čímž efektivně docílí přetížení všech vlastností *Chaining*. Každý ze článků děděného výčtového typu řetězce obsahuje vlastnost *value* s getterem a setterem pro generování notifikací pro delegáta. Výsledný řetězec *Chaining* tedy neimplementuje pouze jednu heterogenní vlastnost, ale více homogenních přetížených vlastností, které používá k emulaci dědičnosti.

7.4 Limitace implementace

Specifikace pro *InheritableEnum* je navrhnutá pro uživatelské případy užití knihoven *BottomSheet* a *DynamicContent*, které pro své datové model využívají hodnotové datové typy *enum* a u kterých není potřeba číst aktuální zapsanou hodnotu.

Implementace *InheritableEnum* tedy implementuje emulaci dědičnosti pouze nad výčtovými typy a poskytuje silně typovou kontrolu pouze pro nastavování hodnot množiny výčtových typů.

Případné rozšíření pro emulaci dědičnosti nad strukturami a silně typové kontroly čtení aktuálně zapsaných hodnoty, není vyloučeno.

Kapitola 8

Ověření implementace

Pro ověření implementace knihoven jsem vytvořil dva projekty, implementující komponenty pro *SwiftUI* a *UIKit*. Projekt pro demonstraci by měl efektivně využívat jak *BottomSheet*, tak *DynamicContent*. Pro tento účel jsem vymyslel aplikaci *Meteorites*.

Meteorites je prostá aplikace zobrazující místa dopadů meteoritů na mapě. Mapu meteoritů komplementuje abecedně seřazený seznam v posuvném panelu komponenty *BottomSheet*. Při uživatelské akci vybrání konkrétního meteoritu, aplikace schová posuvný panel se seznamem, změní obsah seznamu na obsah zobrazující detailní informace vybraného meteoritu a zobrazí panel znovu uživateli.

Data v aplikaci *Meteorites* se načítají dynamicky z internetu. Pro tento účel jsem využil volně dostupný dataset meteoriteoritů z databáze NASA[23]. Aplikace Meteorites pro tuto část využívá komponentu knihovny DynamicContent, kterou zobrazí v posuvném panelu. Tato komponenta pro svůj dynamický obsah implementuje výchozí stavy pro načítání, prázdný stav, chybový stav a zobrazení primárního obsahu.

Kapitola 9

Závěr

Výsledekm mé práce jsou dvě komponenty *BottomSheet* a *DynamicContent*, implementované pomocí knihoven pro práci s uživatelským rozhraním *UIKit* a *SwiftUI*; knihovna *InheritableEnum* umožňující emulovat dědičnost nad výčtovými typy; a dvě demonstrační aplikace.

BottomSheet implementuje navigační komponentu pro zobrazení primárního obsahu se spodním posuvným panelem obsahující komplementární scénu. Tato komponenta je vhodná pro aplikace, které chtějí dát důraz na obsah, ale které zároveň potřebují zobrazit uživateli komplexní doplňkové možnosti.

DynamicContent implementuje komponentu zobrazující opakující se stavy pro dynamicky načítaný obsah. Komponenta se snaží řešit každodenní rutinní práce spojené s řešením explozí stavů způsobené nemožností zajistit spolehlivé načítání obsahu.

Knihovna *InheritableEnum* umožňuje emulovat silně typovou dědičnost nad výčtovými typy. Konstrukce, umožňující dědičnost, je implementovaná v protokolově orientovaném paradigmatu. Tuto knihovnu využívají komponenty pro rozšíření možností datových modelů definovatelné uživatelem knihovny.

Výsledné knihovny jsou demonstrovány na aplikacích *Meteorites*, které tyto knihovny implementují. *Meteorites* je jednoduchá aplikace zobrazující dynamicky načítané místa dopadů meteoritů na mapě, s posuvným panelem obsahující list meteoritů ve stavové komponentě *DynamicContent*.

V neposlední řadě jsem se v této práci zaměřil na rozdílnost přístupu nové deklarativní knihovny *SwiftUI*, ve které byly implementovány ekvivalentní funkcionality komponent pro *SwiftUI* a *UIKit*.

Výsledkem je 5 knihoven a dvě demonstrační aplikace.

Literatura

[1] Apple. *About the Swift programming language* [online]. [cit. 2022-05-10]. Dostupné z: https://developer.apple.com/swift/.

[2] Apple. *Apple Design Awards* [online]. [cit. 2022-05-10]. Dostupné z: https://developer.apple.com/design/awards/.

[3] Apple. *Building Layouts with Stack Views* [online]. [cit. 2022-05-10]. Dostupné z: https://developer.apple.com/documentation/swiftui/building-layouts-with-stack-views.

[4] Apple. *Creating a Custom Container View Controller* [online]. [cit. 2022-05-10]. Dostupné z: https://developer.apple.com/documentation/uikit/view_controllers/creating_a_custom_container_view_controller.

[5] Apple. *Designing Fluid Interfaces* [online]. [cit. 2022-05-10]. Dostupné z: https://developer.apple.com/videos/play/wwdc2018/803/.

[6] Apple. *Dokumentace Button z knihovny SwiftUI* [online]. [cit. 2022-05-10]. Dostupné z: https://developer.apple.com/documentation/swiftui/button.

[7] Apple. *Dokumentace GeometryReader z knihovny SwiftUI* [online]. [cit. 2022-05-10]. Dostupné z: https://developer.apple.com/documentation/swiftui/geometryreader.

[8] Apple. *Dokumentace protokolu View z knihovny SwiftUI* [online]. [cit. 2022-05-10]. Dostupné z: https://developer.apple.com/documentation/swiftui/view.

[9] Apple. *Dokumentace struktury Text z knihovny SwiftUI* [online]. [cit. 2022-05-10]. Dostupné z: https://developer.apple.com/documentation/swiftui/text.

[10] Apple. *Dokumentace UIButton z knihovny UIKit* [online]. [cit. 2022-05-10]. Dostupné z: https://developer.apple.com/documentation/uikit/uibutton.

[11] Apple. *Dokumentace UICollectionView z knihovny UIKit* [online]. [cit. 2022-05-10]. Dostupné z: https://developer.apple.com/documentation/uikit/uicollectionview.

[12] Apple. *Dokumentace UIControl z knihovny UIKit* [online]. [cit. 2022-05-10]. Dostupné z: https://developer.apple.com/documentation/uikit/uicontrol.

[13] Apple. *Dokumentace UIImageView z knihovny UIKit* [online]. [cit. 2022-05-10]. Dostupné z: https://developer.apple.com/documentation/uikit/uiimageview.

[14] Apple. *Dokumentace UILabel z knihovny UIKit* [online]. [cit. 2022-05-10]. Dostupné z: https://developer.apple.com/documentation/uikit/uilabel.

[15] Apple. *Dokumentace UINavigationController z knihovny UIKit* [online]. [cit. 2022-05-10]. Dostupné z: https://developer.apple.com/documentation/uikit/uinavigationcontroller.

[16] Apple. *Dokumentace UIScrollView z knihovny UIKit* [online]. [cit. 2022-05-10]. Dostupné z: https://developer.apple.com/documentation/uikit/uiscrollview.

[17] Apple. *Dokumentace UISplitViewController z knihovny UIKit* [online]. [cit. 2022-05-10]. Dostupné z: https://developer.apple.com/documentation/uikit/uisplitviewcontroller.

[18] Apple. *Dokumentace UIStackView z knihovny UIKit* [online]. [cit. 2022-05-10]. Dostupné z: https://developer.apple.com/documentation/uikit/uistackview.

[19] Apple. *Dokumentace UITableView z knihovny UIKit* [online]. [cit. 2022-05-10]. Dostupné z: https://developer.apple.com/documentation/uikit/uitableview.

[20] Apple. *Dokumentace UIView z knihovny UIKit* [online]. [cit. 2022-05-10]. Dostupné z: https://developer.apple.com/documentation/uikit/uiview.

[21] Apple. *Dokumentace UIViewController z knihovny UIKit* [online]. [cit. 2022-05-10]. Dostupné z: https://developer.apple.com/documentation/uikit/uiviewcontroller.

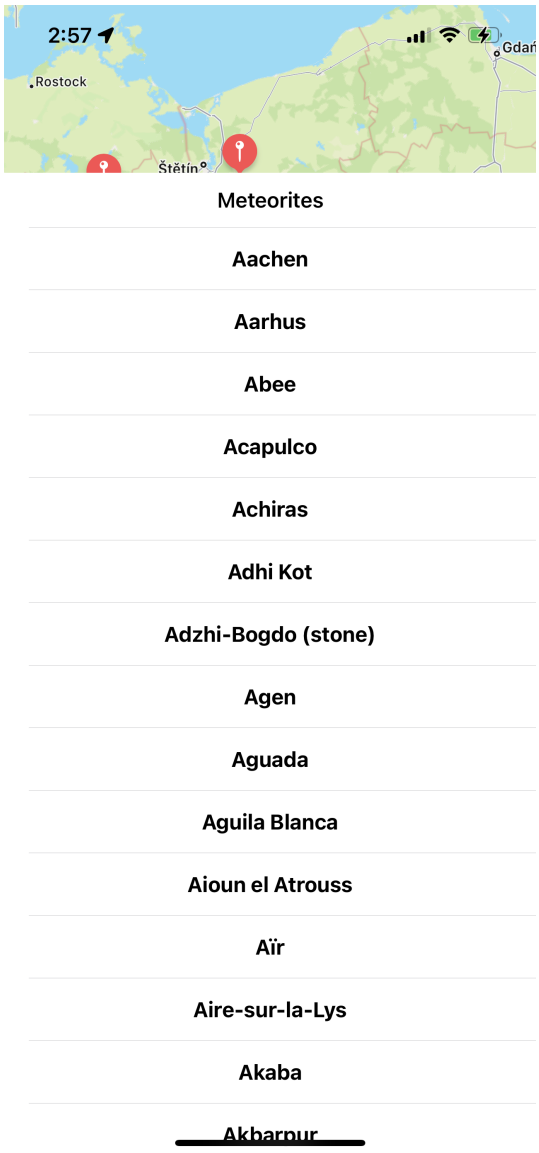
[22] Apple. *Protocol-Oriented Programming in Swift* [online]. [cit. 2022-05-10]. Dostupné z: https://developer.apple.com/videos/play/wwdc2015/408/.

[23] NASA. *Meteorite Landings* [online]. [cit. 2022-05-10]. Dostupné z: https://dev.socrata.com/foundry/data.nasa.gov/y77d-th95.

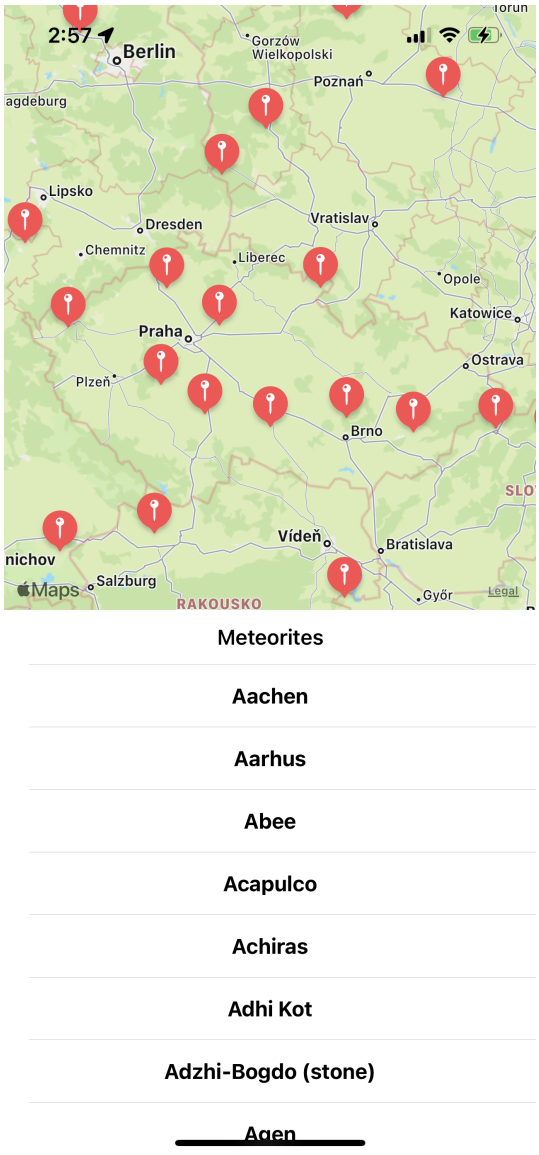
Příloha A

Obrazovky demonstrační aplikace UIKit

Obrázek A.1: Demonstrace dynamického obsahu se stavem načítání



Obrázek A.2: Demonstrace dynamického obsahu se stavem primárního obsahu



Obrázek A.3: Demonstrace posuvného panelu BottomSheet

Aachen

Rok dopadu	1880
Hmotnost	21 g
Třída	L5
Zeměpisná šířka	50.775000
Zeměpisná délka	6.083330

Obrázek A.4: Demonstrace posuvného panelu BottomSheet s detailem meteoritu

Příloha B

Obrazovky demonstrační aplikace SwiftUI

Obrázek B.1: Demonstrace dynamického obsahu se stavem načítání

Meteorites

Aachen

Aarhus

Abee

Acapulco

Achiras

Adhi Kot

Adzhi-Bogdo (stone)

Agen

Aguada

Aguila Blanca

Aioun el Atrouss

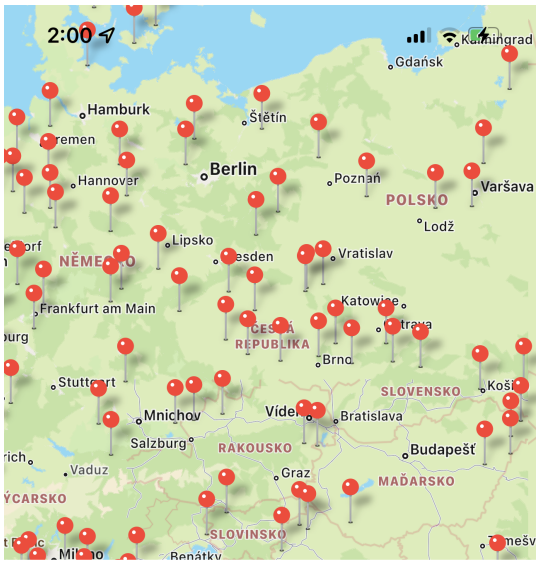
Aïr

Aire-sur-la-Lys

Akaba

Akbarpur

Obrázek B.2: Demonstrace dynamického obsahu se stavem primárního obsahu



Meteorites

Aachen

Aarhus

Abee

Acapulco

Achiras

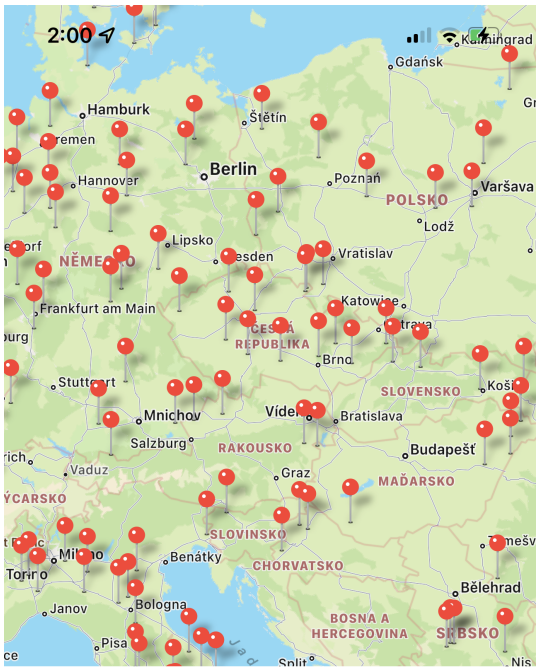
Adhi Kot

Adzhi-Bogdo (stone)

Agen

Aguada

Obrázek B.3: Demonstrace posuvného panelu BottomSheet



Aachen

Rok dopadu	1880-01-01T00:00:00.000
Hmotnost	21 g
Třída	L5
Zeměpisná šířka	50.775000
Zeměpisná délka	6.083330

Obrázek B.4: Demonstrace posuvného panelu BottomSheet s detailem meteoritu