Automatické rozpoznání textu a jeho digitalizace za použití mobilních platforem od společnosti Apple

Dominik Vanduch

Bakalářská práce
2023

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

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

Akademický rok: 2022/2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Dominik Vanduch**
Osobní číslo:	**A20027**
Studijní program:	**B0613A140020 Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Automatické rozpoznání textu a jeho digitalizace za použití mobilních platforem od společnosti Apple**
Téma práce anglicky:	**Automatic Text Recognition and Digitization Using Apple Mobile Platforms**

Zásady pro vypracování

1. Popište základní principy a problematiku vývoje pro mobilní platformy od firmy Apple, včetně nástrojů pro vývoj aplikací k rozpoznání a digitalizaci textu.
2. Vyhledejte a porovnejte konkurenční aplikace, řešící obdobný případ užití.
3. Sestavte funkcionální a nefunkcionální požadavky na aplikaci a vytvořte návrh prototypu aplikace.
4. Implementujte prototypovou aplikaci a popište její důležité části.
5. Otestuje vytvořený prototyp a zhodnoťte možnosti využití zabudovaných knihoven pro rozpoznání a digitalizaci textu.

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

Seznam doporučené literatury:

1. MARQUES, Oge. Image Processing and Computer Vision in iOS. Springer, 2020. ISBN 978-3-030-54030-2.
2. Apple Developer Documentation: Browse the latest developer documentation, including tutorials, sample code, articles, and API reference. Documentation [online]. Apple, 2022 [cit. 2022-12-01]. Dostupné z: https://developer.apple.com/documentation/
3. Vision: Apply computer vision algorithms to perform a variety of tasks on input images and video. Documentation [online]. Apple, 2022 [cit. 2022-12-01]. Dostupné z: https://developer.apple.com/documentation/vision/
4. Documentation: Swift language. Documentation [online]. Apple, 2022 [cit. 2022-12-01]. Dostupné z: https://www.swift.org/documentation/
5. SwiftUI: Declare the user interface and behavior for your app on every platform. Documentation [online]. Apple, 2022 [cit. 2022-12-01]. Dostupné z: https://developer.apple.com/documentation/swiftui/

Vedoucí bakalářské práce: **Ing. Radek Vala, Ph.D.**
Ústav informatiky a umělé inteligence

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

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

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

Ve Zlíně dne 7. prosince 2022

Prohlašuji, že

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

Prohlašuji,

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

Ve Zlíně, dne 25. 5. 2023

Dominik Vanduch, v.r.
podpis diplomanta

ABSTRAKT

Bakalářská práce se zabývá vývojem mobilních aplikací pro rozpoznání textu a jeho digitalizaci v rámci mobilních platforem od společnosti Apple. V teoretické části jsou popsány principy rozpoznávání textu, jeho dělení a použití. Následně jsou prezentovány základní informace o mobilních platformách od společnosti Apple. Závěr teoretické části se věnuje nástrojům a knihovnám, které se používají k vývoji aplikací pro rozpoznání a digitalizaci textu. Praktická část práce je zaměřena na návrh, implementaci a validaci prototypu aplikace pro rozpoznání a digitalizaci textu. Součástí práce je zhodnocení využití zabudovaných knihoven pro rozpoznání a digitalizaci textu a zhodnocení vytvořeného prototypu.

Klíčová slova: rozpoznávání textu, digitalizace textu, mobilní aplikace, iOS vývoj, Vision framework

ABSTRACT

This bachelor's thesis focuses on the development of mobile applications for text recognition and digitization within the mobile platforms of Apple Inc. The theoretical part of the thesis provides a comprehensive explanation of text recognition principles, along with the segmentation and use cases of the text recognition process. Subsequently, it presents fundamental information about Apple's mobile platforms. The conclusion of the theoretical part is devoted to the tools and libraries used in the development of applications for text recognition and digitization. The practical section of the thesis is centered on the design, implementation, and validation of a prototype application for text recognition and digitization. The thesis includes an evaluation of the utilization of embedded libraries for text recognition and digitization, as well as an evaluation of the created prototype.

Keywords: text recognition, text digitization, mobile application, iOS development, Vision framework

Rád bych poděkoval vedoucímu mé bakalářské práce panu Ing. Radku Valovi, Ph.D. za odborné vedení, ochotu vždy zodpovědět všechny mé dotazy a celkově za skvělý přístup. Dále bych rád poděkoval panu Ing. Romanu Manďákovi za pomoc při návrhu tématu a cenné rady během vypracovávání práce. Na závěr bych rád poděkoval všem ostatním, kteří aplikaci testovali, nebo se jinak podíleli na jejím vývoji.

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

OBSAH

ÚVOD

V dnešní době se často setkáváme se situacemi, kdy různá zařízení či senzory mají omezenou nebo zcela žádnou konektivitu. I v případech, kdy by bylo technologicky možné zařízení připojit k síti a sbírat z nich data, může toto připojení být složité nebo finančně náročné. To následně komplikuje sběr a zpracování dat z těchto zařízení. V situacích, kdy je potřeba data ze zařízení uchovávat, se obvykle používá optický odečet hodnot, ruční přepis a následné ruční zadání do informačního systému. Tento proces je však časově náročný a náchylný na chyby. Z toho důvodu je tato práce zaměřena na to, zda a případně v jaké míře mohou ve výše zmíněných případech pomoci zabudované nástroje v mobilních systémech společnosti Apple, jako je Vision framework.

V teoretické části bude obecně popsáno rozpoznání textu, jeho dělení a také využití. Dále bude popsán vývoj pro mobilní platformy společnosti Apple a nástroje využívané pro tvorbu software na tyto platformy, jako jsou například Swift programovací jazyk, SwiftUI pro tvorbu uživatelského rozhraní nebo Vision framework pro samotné rozpoznávání textu.

V praktické části bude navržen a implementován prototyp aplikace, která zobrazuje možnosti využití technologií popsaných v teoretické části. Po implementaci návrhu bude prototypová aplikace otestována, aby byla ověřena její funkčnost a spolehlivost. Následně bude zhodnocena možnost využití mobilních platforem Apple pro digitalizaci textu. V závěru praktické části bude vyhodnocena celková funkčnost a účinnost vytvořeného prototypu aplikace.

Výsledkem práce bude přiblížení problematiky využití mobilních platforem společnosti Apple pro digitalizaci textu a přiblížení praktických přístupů k digitalizaci dat, které budou aplikovány do vytvořeného prototypu aplikace.

I. TEORETICKÁ ČÁST

1 ROZPOZNÁVÁNÍ TEXTU

Rozpoznávání textu je proces, který umožňuje rozpoznávat znaky abecedy v obrázcích, dokumentech, formulářích nebo fakturách.

Dle společností IBM [1], Microsoft [2] a Adobe [3] jsou rozpoznávání textu (anglicky *text recognition*) a optické rozpoznávání znaků (anglicky *Optical Character Recognition*, zkráceně OCR) ekvivalentní pojmy. Společnost Google však vnímá rozpoznávání textu jako pojem zastřešující několik podkategorií, mezi které patří i samotné rozpoznávání textu z obrázků nebo dokumentů, ale také analýza textu a rozpoznání jazyka [5]. V této práci budou názvy rozpoznávání textu a optické rozpoznávání znaků vnímány ekvivalentně a dále bude využíváno zkratky OCR.

Význam OCR v současné digitální době nelze přehlédnout. Technologie využívající OCR hrají klíčovou roli v mnoha oblastech, které sahají od podnikání a vzdělání až po vládní a zdravotnické sektory. Tyto technologie pomáhají šetřit čas a zvyšují efektivitu tím, že automatizují procesy, které by jinak vyžadovaly náročné manuální práce. Například, místo ručního přepisování tisknutých dokumentů do digitální formy, OCR umožňuje rychlé a přesné převedení těchto dokumentů do upravitelného a vyhledatelného textu. To značně urychluje zpracování dat a umožňuje efektivnější správu a archivaci dokumentů. OCR také umožňuje přístup k informacím pro osoby se zrakovým postižením, čímž jim zvyšuje dostupnost textových materiálů. Tímto způsobem OCR technologie přispívají k digitální transformaci a podporují inovace v mnoha oblastech. [3]

V dalších podkapitolách je podrobněji popsán princip OCR, jeho dělení a praktické využití v různých odvětvích.

1.1 Princip OCR

Prvním krokem pro použití OCR je vytvoření digitálního dokumentu nebo snímku, na který bude proces aplikován. Nejčastěji se jedná o skenování skenerem nebo pořízení fotografie fotoaparátem.

Před samotnou detekcí znaků je potřeba snímek zpracovat do podoby, která zaručí nejvyšší efektivitu detekce. Obraz každého snímku je tvořen konečným počtem bodů, zvaných pixely, které jsou uspořádány do dvourozměrné mřížky. Každý pixel v sobě uchovává informaci o své barvě. Jedním z prvních kroků bývá obvykle převedení snímku, tedy jednotlivých pixelů, na dvoubarevnou verzi (často černobílou). [1]

Další kroky zpracování snímku zahrnují jeho vyrovnání nebo naklonění, odstranění skvrn a vyhlazení čar v obraze. Z takto upravené verze je následně detekováno pozadí, které je u běžného papíru bílé, a znaky, které bývají černé. Tvary detekované černou barvou jsou zpracovány pro nalezení jednotlivých znaků. Tato detekce znaků se může být prováděna po jednotlivých znacích, slovech nebo celých blocích textu. Detekované znaky jsou identifikovány jedním ze dvou algoritmů – *pattern matching* nebo *feature extraction.* [1]

Pattern matching vychází z rozpoznání jednotlivého znaku, který se označuje jako glyf. Algoritmus porovná detekovaný glyf s glyfy uloženými, které vykazují určitou shodu. Tato metoda funguje pouze pokud mají uložené glyfy podobný font a velikost jako ty vstupní. Z toho důvodu je algoritmus vhodný tehdy, pokud je font, kterým je text ve snímku psán, předem známý. [5]

Feature extraction rozkládá jednotlivé glyfy na čáry, smyčky, směry a křížení čar. Tyto jednotlivé prvky jsou potom využity k nalezení nejvyšší shody mezi uloženými glyfy. [5]

1.2 Dělení OCR

V této podkapitole je podrobně popsáno dělení OCR do různých aspektů, zahrnující jeho závislost na internetu, rozdílné typy písma a různé metody a techniky využívané pro rozpoznávání textu.

1.2.1 Dělení dle závislosti na internetu

a) **Online** – Tento způsob využívá cloudových služeb, které provedou detekci a uživateli je dodán až výsledek detekce. Znamená to tedy, že nejsou požadavky na výkonnost nebo softwarové vybavení zařízení uživatele. Hlavní podmínkou pro využití této metody je přístup k internetu.

b) **Offline** – OCR prováděné offline využívá plně zařízení uživatele a není potřeba přístup k internetu. Tento způsob je náročnější na software a hardware zařízení, na kterém je rozpoznávání prováděno, ale přináší lepší ochranu a soukromí dat, jelikož vstupní snímek i výstupní data neopouštějí zařízení uživatele.

1.2.2 Dělení dle typu písma

a) **Ruční písmo**

b) **Strojové písmo**

1.2.3 Dělení dle použití

a) **Jednoduchý OCR** – software ukládá fonty a obrazové vzory jako šablony a porovnává je s interní databází znak po znaku pomocí dříve zmíněného algoritmu *pattern matching*. Tento typ řešení má omezení v zachycení různých stylů písma a rukopisu. [3]

b) **Inteligentní rozpoznávání znaků** (známé také pod anglickou zkratkou ICR – *Inteligent Character recognition*) – využívá moderní OCR systémy, u kterých je využito strojového učení. Tyto systémy analyzují text podobně jako lidé, a to znak po znaku pomocí algoritmu *feature extraction*. [3]

c) **Inteligentní rozpoznávání slov** (známé také pod anglickou zkratkou IWR – *Inteligent Word Recognition*) – funguje na stejných principech jako inteligentní rozpoznávání znaků, ale místo toho zpracovává celé obrazy slov. [3]

d) **Optické rozpoznávání značek** – identifikuje loga, vodoznaky a další textové symboly v dokumentu. [3]

1.3 Využití OCR

OCR má široké možnosti využití v mnoha odvětvích. Případy užití sahají od běžného života až po vysoce specializované průmyslové využití. Následující podkapitoly se věnují nejběžnějším a zajímavým případům využití OCR.

1.3.1 Digitalizace dokumentů

Digitalizace dokumentů je jeden z nejčastějších případů využití OCR. Díky této technologii je možné převést skenované dokumenty, PDF soubory nebo obrázky s textem na přizpůsobitelný a vyhledatelný text. Výhodou digitalizace dokumentů je snížení potřeby fyzického úložného prostoru, snadné hledání a sdílení informací. Digitalizované dokumenty mohou být pro zvýšení bezpečnosti chráněny například heslem nebo jiným způsobem zabezpečení.

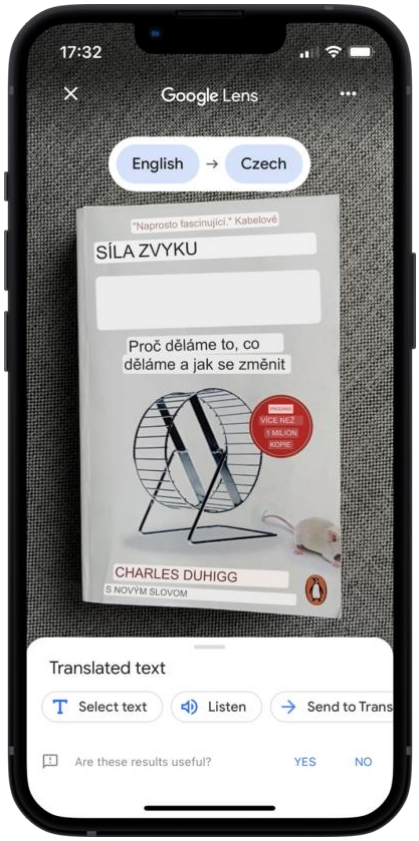
Například platforma iOS od společnosti Apple nabízí uživatelům několik zabudovaných nástrojů a aplikací, které umožňují digitalizaci dokumentů s vysokou úrovní efektivity. Jednou z těchto aplikací je zabudovaná aplikace Poznámky (viz obrázek číslo 1), která umožňuje přidávat, upravovat a ukládat digitální kopie dokumentů. Tato integrace umožňuje

uživatelům snadný a přístupný způsob, jak digitalizovat a ukládat dokumenty na svém zařízení s minimálním úsilím a maximální kvalitou.

Obrázek 1 Rozhraní iOS aplikace Poznámky
při digitalizaci dokumentů [6]

1.3.2 OCR technologie v reálném čase

Oblíbenou aplikací využívající OCR v reálném čase je Google Translate. Tato aplikace používá sofistikovanou OCR technologii ke snímání a rozpoznání textu prostřednictvím kamery mobilního zařízení. Jakmile je text rozpoznán, je pomocí technologie strojového učení okamžitě přeložen do předem vybraného jazyka. Tato aplikace implementující technologii OCR pro okamžitý překlad textu se prokázala jako užitečná v širokém spektru situací, včetně cestování, při studiu jazyků, nebo kdykoliv je vyžadován rychlý a přesný překlad cizího textu.



Obrázek 2 Rozhraní iOS aplikace Google Translate

1.3.3 Rozpoznávání registračních značek silničních vozidel

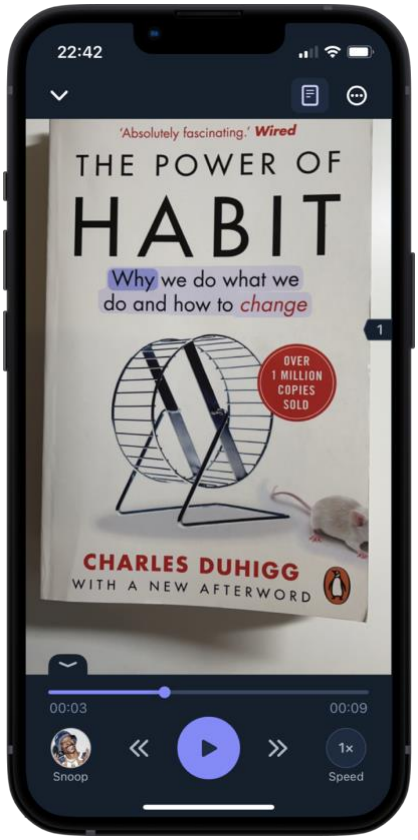
Jedním z případů užití OCR, se kterým je každý řidič dopravních prostředků v kontaktu, je rozpoznávání registračních značek silničních vozidel. Takto rozpoznaná data slouží například k měření rychlosti pomocí radaru, detekci kradených vozidel nebo automatické otevírání parkovacích závor.

1.3.4 Převod textu na řeč

OCR ve spojení s technologií pro převod textu na řeč (anglicky *Text To Speech*, zkráceně TTS) představuje specifický přístup k interakci s textovými zdroji. Toto spojení umožňuje rozpoznávat text z digitálních materiálů a následně pomocí uměle vygenerovaného hlasu dojde k přečtení textu obsaženého ve fotografiích, dokumentech nebo jiných zvolených zdrojích. Jedno z důležitých využití této technologie je jeho integrace přímo do systému iOS, které na úrovni platformy nabízí funkcionalitu zpřístupnění, včetně detekce textu na obrázcích a jeho následného přečtení.

Speechify představuje další příklad, jež využívá obou technologií – OCR i TTS. Tato aplikace, zkoumána v kontextu průzkumu aplikací využívajících OCR, nabízí zpracování rozmanitých typů dat, včetně fotografií, dokumentů či webových stránek. Uživatelské rozhraní aplikace (viz obrázek číslo 3) poskytuje možnost modifikace hlasu pro čtení vybraného textu, včetně volby hlasů několika známých osobností. Kromě toho umožňuje přizpůsobení rychlosti čtení a zobrazování přepisu čteného textu.

Obecně je software sloužící pro převod textu na řeč užitečný například při naraci videí, tvorbě audioknih nebo jako technologie zpřístupnění pro lidi s vadou zraku nebo jinými poruchami. Zakladatel společnosti Speechify Inc., jež je vývojářem stejnojmenné aplikace, uvádí, že aplikaci vytvořil z důvodu, že má kvůli dyslexii problém číst a tato aplikace mu čtení usnadňuje. [7]



Obrázek 3 Rozhraní iOS aplikace Speechify

1.3.5 Využití v průmyslu

Při použití OCR v průmyslu je nejčastěji využíváno cloud technologií. Všechny tři největší platformy pro poskytování cloud computingu, kterými jsou Amazon Web Services,

Microsoft Azure a Google Cloud Platform, mají ve svém portfoliu také služby OCR [8]. Níže budou popsány produkty využívající OCR technologie, které tyto tři společnosti nabízí.

Amazon Web Services, Inc

Nejvýznamnější společností v této oblasti je Amazon Web Services, Inc (zkráceně AWS), která má největší podíl trhu [8]. OCR služba, která je nabízená zákazníkům AWS, se nazývá Amazon Textract a slouží k extrakci tištěného textu, ručně psaného písma nebo dat z jakéhokoliv dokumentu. Na rozdíl od běžného OCR společnost Amazon implementovala vlastní algoritmy, které mají za cíl identifikovat, pochopit a následně extrahovat textová data. Nejedná se tedy o pouhou identifikaci a použití algoritmu na rozpoznání znaku. Amazon Textract spoléhá na strojové učení, které eliminuje manuální a drahé procesy ručního konfigurování OCR. Mezi hlavní typy průmyslů, ve kterých tento produkt prezentují, jsou finanční služby, zdravotnictví a veřejný sektor. [9]

Společnost AWS uvádí například případovou studii pro společnost Anthem, která se zabývá zdravotním pojištěním. V této studii díky produktu Amazon Textract společnost Anthem zmiňuje, že v roce 2021 automatizovali 80 % jejich pracovních postupů při zpracování žádostí. Společnost uvedla, že očekává nárůst této hodnoty až na 90 %. [10]

Microsoft Azure

Microsoft Azure je cloudová platforma od společnosti Microsoft. OCR služba nabízená touto platformou se nazývá Read. Podobně, jako tomu bylo u AWS, Read je založen na speciálně vyvinutých algoritmech a strojovém učení. Dle vstupu do algoritmu se dále dělí na Computer Vision pro zpracování obrázků a Form Recognizer, který je specializován na detekci v digitálních i skenovaných dokumentech včetně obrázků. [2]

Jednou ze společností, která tuto platformu využívá, je Fujitsu Limited, která je největší společností zabývající se skenováním dokumentů na globálním trhu, kde má více než 50 % celosvětového podílu. Fujitsu díky produktu Form Recognizer zvýšilo míru úspěšného rozpoznání znaků až na 99,9 %. Před použitím produktu Form Recognizer společnost Fujitsu měla problém dosáhnout 95 %. Přesná data o úspěšnosti rozpoznávání znaků ze zřejmých důvodů společnost neuvádí. Dříve zmíněného navýšení míry úspěšného rozpozávání bylo dosaženo za tři měsíce a pomocí třech vývojářů. [11; 12] Tato fakta zdůrazňují dostupnost a jednoduchost implementace OCR do produktů.

Google Cloud Platform

Třetím nejvýznamnější platformou v oblasti cloud computingu a OCR je Google Cloud Platform (zkráceně GCP), která je vytvořena společností Google LLC. Ve svých nabízených službách má GCP v kategorii Google Cloud AI dvě verze OCR. Podobně jako tomu bylo u Microsoft Azure, GCP nabízí zpracování dokumentů a zpracování obrázku a videí. [13] Tyto verze se liší v předzpracování, ale následně využívají společné umělé inteligence zvané Document AI. Document AI je sadou nástrojů pro extrakci, analyzování, hledání a ukládání dat, vyvinutých společností Google. [14]

Společnost Google se spojila s newyorským deníkem The New York Times, aby jim pomohla digitalizovat jejich rozsáhlou sbírku fotografií. Kompletní sbírka čítá mezi pěti až sedmi miliony fotografií. Tyto fotografie jsou archivovány ve stovkách kartoték tři patra pod úrovní ulice. Přestože je obsah archivu zaznamenán v katalogu, není praktické v tomto archivu vyhledávat. Společnost Google vytvořila ve spolupráci s The Times systém, jenž umožňuje zpracovávat a rozpoznávat text, rukopis a další detaily, které lze ve fotografiích nalézt. Pro spojení dat a fotografií bylo implementováno strojového učení, které tyto mezery doplňuje a po zažádání o fotografii systém vrací také kontext a obsah, který byl nalezen na jiných místech v systému. [15]

1.3.6 Shrnutí využití OCR

Technologie OCR nabízí široké možnosti využití v různých oblastech. Jednotlivé případy užití představené v předešlých podkapitolách ilustrují rozmanité aplikace těchto technologií v praxi.

Důležité je však také zdůraznit, že běžně dostupné aplikace pro digitalizaci textu z fotografií a dokumentů většinou nedokážou automaticky vytvářet asociace mezi digitalizovanými a očekávanými daty na základě uživatelských požadavků. Výjimkou jsou aplikace velice specificky zaměřené na určité typy dat, jako jsou například informace z vizitek nebo konkrétních typů dokumentů. **Právě jednomu přizpůsobitelnému přístupu k automatizaci sběru dat pomocí rozpoznání textu se věnuje prototypová aplikace, která je popsána v praktické části této práce.**

2 MOBILNÍ PLATFORMY SPOLEČNOSTI APPLE

Společnost Apple nabízí v současnosti tři mobilní platformy. Mezi tyto platformy patří iOS, iPadOS a watchOS. S výjimkou macOS jsou všechny operační systémy společnosti Apple uzavřené a je na ně možné instalovat aplikace pouze za použití předinstalované aplikace App Store. Mobilní platformy Apple a aplikace App Store jsou popsány podrobněji v následujících kapitolách.

2.1 iOS

Mezi mobilními operačními systémy společnosti Apple je nejrozšířenější iOS, jelikož je instalován na její nejpopulárnější produkt, kterým je iPhone. Apple prodal již přes dvě miliardy zařízení iPhone [16] a v roce 2021 jich prodal 228 milionů [17]. První verze byla představena v roce 2007 pod názvem iPhone OS a od té doby vychází každý rok verze nová [18]. Tento systém vycházel z operačního systému pro počítače zvaného Mac OS X.

2.2 iPadOS

Operační systém iPadOS je speciálně navržený pro zařízení z kategorie tabletů – iPad. Prvně zařízení iPad sdílela stejný operační systém jako iPhone, ale v roce 2019 došlo k oddělení a nyní je na těchto zařízeních instalován operační systém iPadOS. [19] Právě z důvodu nedávného oddělení je běžné se v praxi setkat s tím, že jsou systémy iPadOS a iOS považovány za varianty stejné platformy. Tento systém nabízí několik unikátních funkcí, jako například podporu pro více otevřených oken najednou, podporu pro stylus Apple Pencil nebo podporu pro bezdrátové využití iPadu jako externí displej (pouze pro zařízení Mac).

2.3 watchOS

Zařízení Apple Watch mají instalovaný operační systém watchOS, který byl vydán v roce 2015. Tento operační systém je také založen na iOS. Podobnost mezi iOS a watchOS není jen pro ušetření práce na vývoji, ale také umožňuje hluboké propojení jednotlivých zařízení. Toto propojení je obzvláště důležité mezi zařízením iPhone a Apple Watch, jelikož tyto hodinky není bez zařízení iPhone možné používat. [20] Pro digitalizaci textu tento operační systém nemá příliš využití, a proto je tato práce zaměřena hlavně na iOS (případně iPadOS).

2.4 App Store

App Store je digitální distribuční platforma pro operační systémy společnosti Apple. Prvně byl představen na iOS v roce 2008 a od počátku představuje jediný zdroj aplikací pro zařízení s těmito operačními systémy. [21] Později byl představen také na ostatních operačních systémech této společnosti. Všechny aplikace jsou před vydáním pečlivě kontrolovány společností Apple, aby byly v souladu s jejich pravidly a zásadami. Každá aplikace, jež žádá o zveřejnění v App Store, je prověřena, zda neobsahuje malware (pro zařízení nebo uživatele škodlivý kód). Ročně je zamítnuto přes 215 tisíc žádostí o zveřejnění v App Store. Kromě toho musí každá stažená aplikace žádat o přístup k datům nebo funkcím zařízení a uživatel je na tyto požadavky upozorněn ještě před stažením aplikace. Díky tomu je App Store považován za bezpečný zdroj aplikací pro uživatele. V současné době je v App Store k dispozici 1,8 milionu aplikací a přes 2,3 milionu jich bylo odstraněno, protože nebyly udržovány nebo nefungují na posledních verzích operačních systémů. [22] V posledních letech byl Apple výrazně kritizován a žalován za vysoké poplatky, které standardně činní 30 % z účtované částky. To může být pro vývojáře obtížné, pokud se snaží udržet cenu svých aplikací konkurenceschopnou. Apple na tyto události reagoval snížením sazby na 15 % nebo úplným zrušením poplatků pro specifické typy aplikací. V některých zemích byl soudem nucen zahrnout alternativní způsoby plateb nebo snížit poplatky. [23]

2.5 Shrnutí mobilních platforem od společnosti Apple

Mobilní operační systémy společnosti Apple a distribuční platforma App Store slouží jako nezbytné prvky pro vývojáře při vytváření a distribuci aplikací. Díky nim mají vývojáři přístup k velkému počtu uživatelů a mohou si tak snadněji najít své zákazníky. To může vést k vyššímu zisku a většímu úspěchu aplikace. Kromě toho vývojáři získávají také přístup k nástrojům pro testování a zpětnou vazbu od uživatelů.

Operační systémy, jako je například iOS (ale také Android), poskytují vývojářům soubor nástrojů a tzv. API (rozhraní pro programování aplikací) [24], které umožňují vytváření sofistikovaných aplikací s vysokou úrovní funkčnosti. To vývojářům umožňuje vytvářet kvalitní aplikace, které mohou být snadno integrovány s dalšími službami a systémy.

Další výhodou těchto operačních systémů a App Store je, že poskytují vývojářům přístup k širokému spektru uživatelských dat, což jim umožňuje zlepšovat a přizpůsobovat své aplikace na základě chování uživatelů. To může vést ke zlepšení uživatelských interakcí v aplikacích nebo celkové zvýšení spokojenosti uživatelů při používání aplikace.

Vývojáři však musí také dodržovat pravidla a podmínky stanovené těmito mobilními operačními systémy a platformou App Store, což může omezovat jejich svobodu a ovlivňovat způsob, jakým mohou vytvářet a distribuovat své aplikace.

3 POPIS NÁSTROJŮ A KNIHOVEN

V této kapitole jsou popsány nástroje a knihovny používané při vývoji aplikací pro iOS. Detailně je představen programovací jazyk Swift, framework SwiftUI pro tvorbu uživatelských rozhraní, Vision pro detekci objektů v obrázcích a Core Data pro správu perzistentních dat. Dále je popsáno vývojové prostředí Xcode, sada symbolů SF Symbols a nástroje pro design – Excalidraw a Figma.

3.1 Swift

Swift je moderní a výkonný programovací jazyk, který byl představen v roce 2014 na každoroční vývojářské konferenci s názvem „*World Wide Developer Conference*“ (zkráceně WWDC), jakožto nový preferovaný jazyk pro vývoj pro platformy od společnosti Apple (vedle jazyků Objective-C a C) [25]. O rok později byl zveřejněn jeho zdrojový kód a programovací jazyk se stal oficiálně *open-source* [26]. Swift umožňuje psát software pro široké spektrum zařízení, včetně iPhone (iOS), Mac (macOS), Apple Watch (watchOS) a dalších. Tento jazyk kombinuje znalosti inženýrů společnosti Apple spolu s *open-source* komunitou. Kompilátor tohoto jazyka je optimalizován pro výkon, zatímco jazyk je koncipován tak, aby co nejvíce usnadnil a zefektivnil práci vývojářů. Těchto dvou cílů bylo dosaženo způsobem, aniž by v první nebo druhé oblasti došlo ke kompromisům. [27]

Swift usiluje o minimalizaci běžných programátorských chyb tím, že využívá následující moderní programátorské vzory a postupy:

- proměnné jsou před použitím vždy inicializovány,
- indexy pole jsou kontrolovány proti překročení jeho hranic,
- celá čísla jsou kontrolována na přetečení (anglicky *overflow*),
- volitelné hodnoty (anglicky *optionals*) zajišťují explicitní ošetření případů, kdy daný objekt nemá přiřazenou hodnotu (neboli má hodnotu *nil*, což je reprezentace absence platné instance pro daný datový typ – u některých jiných programovacích jazyků je ekvivalentem *null*),
- paměť je spravována automaticky,
- zacházení s chybami umožňuje řízené zotavení z neočekávaných selhání. [27]

3.1.1 Hello World

Při představování jazyka je zvykem jako první program napsat tzv. *Hello World*. [28] Jak tento program vypadá v jazyce Swift je zobrazeno níže na obrázku číslo 4.

```
1 print("Hello, world!")
2 // Prints "Hello, world!"
```

Obrázek 4 Programu Hello World
v jazyce Swift [28]

Tato syntaxe může být povědomá těm, kteří psali kód v jazycích C nebo Objective-C. Swift se totiž vyvinul na základě některých principů dříve zmíněných jazyků. Pro tento program není nutné importovat knihovnu pro funkce sloužící pro vstup/výstup nebo manipulaci s řetězci. Kód psaný v globálním rozsahu slouží jako vstupní bod programu, takže není třeba psát ani hlavní funkci (často označovanou jako *main*). Navíc není nutné psát středníky na konci každého příkazu, jak je tomu v některých jiných jazycích. Tento styl zápisu kódu je elegantní, přehledný a šetří čas programátorovi. [28]

3.1.2 Proměnné

V jazyce Swift rozlišujeme dva typy proměnných. Standardní proměnnou je možno vytvořit pomocí klíčového slova *var*, konstantu pak pomocí klíčového slova *let*. Hodnota konstanty nemusí být známa při překladu programu, ale musí jí být celkově přiřazena hodnota právě jednou. Konstanta nebo proměnná musí mít stejný typ jako hodnota, která jí má být přiřazena. Zadání hodnoty při vytváření konstanty nebo proměnné umožňuje překladači odvodit její typ. Obrázek číslo 5 ilustruje situaci, kde překladač odvodí datový typ proměnné *myVariable* na základě její počáteční hodnoty, která je celým číslem. [28]

```
1 var myVariable = 42
2 myVariable = 50
3 let myConstant = 42
```

Obrázek 5 Proměnné a konstanty
v jazyce Swift [28]

Typy proměnných nejsou nikdy implicitně měněny. V případě, že počáteční hodnota neposkytuje dostatek informací pro určení typu (nebo pokud počáteční hodnota neexistuje), požadovaný typ je možno napsat za proměnnou a oddělit jej dvojtečkou. [28]

3.1.3 Řízení toku

Podmíněný průchod programem je realizován pomocí klíčového slova *if* nebo *switch*, za které následuje logický výraz. Pokud je tento výraz pravdivý, provede se část kódu určená touto podmínkou. Jestliže výraz není pravdivý, je příslušný blok kódu přeskočen. Podmínku *if* je možno rozšířit o další logické výrazy pomocí *else if*. Pro definování kódu, který má být vykonán v případě, že předešlé logické výrazy nejsou pravdivé, slouží klíčová slova *else* u *if* podmínek nebo *default* u *switch* podmínek. [28]

Dalším způsobem řízení toku programu jsou smyčky. Pro tvorbu smyček slouží cykly *for-in*, *while* a *repat-while*. Tělo podmínek i smyček se píše dovnitř složených závorek (viz obrázek číslo 6). [28]

```
let individualScores = [75, 43, 103, 87, 12]
var teamScore = 0
for score in individualScores {
    if score > 50 {
        teamScore += 3
    } else {
        teamScore += 1
    }
}
print(teamScore)
// Prints "11"
```

Obrázek 6 Příklad použití cyklu for a podmínky if [28]

V jazyce Swift je možné deklarovat proměnnou, která může, ale nemusí obsahovat hodnotu. Tyto typy proměnných se nazývají *optionals* (česky nepovinné). Nepovinný typ proměnné buď obsahuje konkrétní hodnotu, nebo je nastavena na *nil*, což značí její absenci. [28]

Je běžné kombinovat klíčová slova *if* a *let* společně pro práci s hodnotami, které mohou chybět. Tento způsob použití je možno vidět na obrázku číslo 7.

```
let nickname: String? = nil
if let nickname {
    print("Hey, \(nickname)")
}
// Doesn't print anything, because nickname is nil.
```

Obrázek 7 Vytvoření nepovinné proměnné a její použití v kódu [28]

Dalším způsobem práce s nepovinnými proměnnými je využití výchozích hodnot. Při tomto způsobu je použito dvou znaků otazníku za proměnnou, za které se uvede výchozí hodnota (viz obrázek číslo 8). [28]

```
1 let nickname: String? = nil
2 let fullName: String = "John Appleseed"
3 let informalGreeting = "Hi \(nickname ?? fullName)"
```

Obrázek 8 Bezpečné rozbalení nepovinné proměnné pomocí výchozí hodnoty [28]

3.1.4 Funkce

Pro deklaraci funkce se používá klíčové slovo *func*. Funkce je následně volána tak, že za jejím názvem bude následovat seznam argumentů v závorkách. Názvy a typy parametrů jsou od návratového typu funkce odděleny pomocí znaků „->“. [28] Deklaraci funkce je možno vidět na obrázku číslo 9.

```
1 func greet(person: String, day: String) -> String {
2     return "Hello \(person), today is \(day)."
3 }
4 greet(person: "Bob", day: "Tuesday")
```

Obrázek 9 Deklarace funkce a její volání [28]

Funkce v jazyce Swift podporují zanoření, což umožňuje vytvořit funkci uvnitř jiné funkce. Jelikož jsou funkce vnímány jako tzv. *first-class type*, mohou vracet jinou funkci jako jejich návratovou hodnotu nebo je přijímat jako vstupní argument. [28]

3.1.5 Struktury a třídy

Struktury a třídy jsou univerzální, flexibilní konstrukce, které tvoří základní prvky programu. Proměnné a metody patřící strukturám a třídám jsou definovány pomocí stejné syntaxe, jako běžné konstanty, proměnné a funkce. [28] Na rozdíl od některých jiných programovacích jazyků Swift nevyžaduje, aby pro vlastní struktury a třídy byly vytvářeny samostatné soubory rozhraní a implementací. V jazyce Swift je struktura nebo třída definována v jediném souboru a externí rozhraní této třídy nebo struktury je automaticky zpřístupněno pro použití dalšímu kódu. [29]

Struktury a třídy mají mnoho společného. Obě mohou:

- definovat proměnné pro ukládání hodnot,
- definovat metody pro poskytování funkcionality,
- definovat tzv. *subscripts*, což jsou zkratky pro přístup k členským prvkům kolekce, seznamu nebo sekvence,
- definovat tzv. *initalizers* (metody, které připraví instanci třídy nebo struktury pro použití),
- být rozšiřovány nad rámec jejich základní implementace,
- splňovat požadavky protokolů (implementovat nezbytné funkce, metody, nebo vlastnosti definované v daném protokolu). [29]

Třídy mají oproti strukturám navíc následující vlastnosti:

- umožňují dědičnost,
- disponují kontrolou a interpretací typu za běhu,
- mají možnosti používat vlastní tzv. *deinitializers* (metody, které předcházejí uvolnění instance třídy z paměti),
- mohou mít více referencí na jednu třídu. [29]

Přidané schopnosti, které třídy podporují, jsou za cenu zvýšené složitosti. Obecně platí, že struktury jsou preferovány a doporučeny vývojáři jazyka Swift. Třídy by měly být použity, když je jejich přidaná funkcionalita vhodná nebo nutná pro daný případ užití. V praxi to znamená, že většina vlastních typů, které budou v rámci zdrojového kódu definovány, budou struktury a výčty. [29]

Třídy se deklarují použitím klíčového slova *class*, za kterým následuje název třídy a blok kódu (viz obrázek číslo 10). Obdobným způsobem se deklarují struktury, které namísto klíčového slova *class* používají slovo *struct*. [28]

```
1 class Shape {
2     var numberOfSides = 0
3     func simpleDescription() -> String {
4         return "A shape with \(numberOfSides) sides."
5     }
6 }
```

Obrázek 10 Deklarace třídy v jazyce Swift [28]

3.1.6 Enumerations

Enumeration je označení pro definici společného typu pro skupinu souvisejících hodnot. Tato definice umožňuje pracovat s hodnotami v rámci programu typově bezpečným způsobem. [30]

V podobných jazycích, jako je například programovací jazyk C, je každé položce ve skupině přiřazena odpovídající celočíselná hodnota. V jazyce Swift jsou *enumerations* více flexibilní a nemusí obsahovat odpovídající hodnotu pro každý prvek ve skupině. Deklarace *enumeration* je zobrazena na obrázku číslo 11. Pokud je pro každý případ *enumeration* poskytnuta hodnota (známá anglicky jako *raw* hodnota), může být tato hodnota typu řetězce textu, znaku nebo hodnotou jakéhokoli celočíselného nebo desetinného typu. [30]

```
1 enum CompassPoint {
2     case north
3     case south
4     case east
5     case west
6 }
```

Obrázek 11 Ukázka deklarace enumeration [30]

3.1.7 Asynchronní a paralelní kód

Swift má zabudovanou podporu pro psaní asynchronního a paralelního kódu. Pro označení funkce jako asynchronní slouží klíčové slovo *async* (viz obrázek číslo 12). Pro spuštění asynchronního kódu paralelně je potřeba napsat při definici konstanty klíčové slovo *async* před slovem *let*. [28]

```
1 func connectUser(to server: String) async {
2     async let userID = fetchUserID(from: server)
3     async let username = fetchUsername(from: server)
4     let greeting = await "Hello \(username), user ID \(userID)"
5     print(greeting)
6 }
```

Obrázek 12 Ukázka asynchronního kódu v jazyce Swift [28]

Pro práci s asynchronním kódem má Swift zabudované typy *Task* a *TaskGroup*. *Task* (česky úloha) je jednotkou práce, která může být spuštěna jako část programu. Z jednotlivých úloh

je možné vytvořit *TaskGroup* (česky skupina úloh), který vývojáři umožňuje mít více kontroly nad prioritou, rušením úloh a také umožňuje vytvářet dynamický počet úloh. [31]

Programovací jazyky jako C++ nebo Java pro paralelní kód využívají konceptu vláken. Swift má nad vlákny abstraktní vrstvu zvanou *concurrency model* a vývojář v ní nepracuje přímo s jednotlivými vlákny. To má za následek to, že vývojář definuje, který kód chce spouštět asynchronně a který paralelně a Swift spravuje jednotlivá fyzická vlákna za něj. [31]

3.1.8 Protokoly a rozšíření

Protokol je definicí požadavků na konkrétní metody, proměnné nebo jiné aspekty, které musí splnit daný objekt či třída pro realizaci určitého úkolu nebo funkcionality. Kromě zadání požadavků, které musí vyhovující typy implementovat, je možno protokol rozšířit o implementaci některých z těchto požadavků nebo o implementaci dalších funkcí, které vyhovující typy mohou využít. Protokoly tedy kombinují koncepty dědičnosti a rozhraní z jiných jazyků. [32] Pro vytvoření protokolu je rezervováno klíčové slovo *protocol*. Rozšíření funkcionality se deklaruje pomocí klíčového slova *extension*. [28]

3.1.9 Vypořádání se s chybami

Error handling (česky vypořádání se s chybami) je proces reagování a zotavení se z chybových stavů v programu. Swift nabízí několik nástrojů, jako jsou například *throwing*, *catching* nebo *propagating*, které usnadňují práci s chybami při využívání programu. [33] Tyto nástroje umožňují tvorbu vlastních chybových hlášek společně s předem definovanými reakcemi pro obnovení chodu programu z takto definovaných chyb.

3.1.10 Generika

Generický kód umožňuje vytvářet flexibilní, znovupoužitelné funkce a typy, které mohou interagovat s libovolným typem (v závislosti na požadavcích, které vývojář definuje). Tento způsob psaní kódu redukuje duplicitu a vyjadřuje svůj záměr jasným a abstraktním způsobem. [34]

Generický kód je běžně využíván a značná část vestavěných knihoven jazyka Swift je napsána tímto způsobem. Mezi názorné příklady patří například proměnná typu pole nebo slovníku (pole hodnot, tvořených vazbou mezi klíčem a hodnotou). Díky generice je možné vytvářet pole a slovníky pro různé typy, i když definice pole i slovníku je pouze jedna. [34]

3.2 SwiftUI

SwiftUI je deklarativní framework určený pro tvorbu aplikací pro všechny platformy od společnosti Apple, který byl představen v roce 2019 na události WWDC [35]. SwiftUI poskytuje nástroje, které umožňují definovat uživatelské rozhraní aplikace a chování specifické pro určité platformy.

Zkratka UI v názvu SwiftUI odkazuje na anglický termín *User Interface*, což v češtině odpovídá pojmu uživatelské rozhraní. Všechny tyto termíny – UI, *User Interface* a uživatelské rozhraní – mají stejný význam a používají se pro označení části softwaru, se kterou uživatel přímo interaguje.

Deklarativní způsob definování UI umožňuje vývojáři popsat, jak mají části aplikace vypadat a jak se mají chovat, zatímco SwiftUI pro něj řeší samotnou implementaci, aby bylo těchto volně definovaných cílů dosaženo. Deklarativní přístup, v kontrastu s imperativním, šetří čas vynaložený na psaní kódu. Je také jednodušší a jasnější ke čtení a pochopení pro vývojáře. [40]

3.2.1 Struktura view souboru

Základním prvkem při tvorbě UI je tzv. *view*. Ve SwiftUI je *view* definováno prostřednictvím protokolu, který reprezentuje jednotlivé části UI aplikace. Tento protokol nabízí modifikátory, pomocí nichž lze konfigurovat vlastnosti a chování zobrazovaných prvků. *Views* se píšou a ukládají do souborů s koncovkou *swift* a tyto soubory jsou nazývány *view* soubory.

Ve výchozím stavu obsahují soubory ve SwiftUI dvě struktury (viz obrázek číslo 13). První struktura náleží *View* protokolu a popisuje obsah a rozvržení UI. Druhá struktura slouží pro vytvoření náhledu pro *view* popsané v první struktuře. [36]

```swift
import SwiftUI

struct ContentView: View {
    var body: some View {
        Text("Hello, World!")
            .padding()
    }
}

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

Obrázek 13 Ukázka dvou základních struktur tvořících *view* soubor ve SwiftUI

3.2.2 Práce s views

Apple nabízí řadu předdefinovaných základních *views*, z nichž lze skládat *views* složitější. Při tvorbě UI tedy dochází k zanoření několika *views* ve více úrovních. V současnosti je předdefinováno celkem 68 základních *views*, mezi něž patří například tlačítko, textový blok nebo tabulka. Vývojáři mohou vytvářet UI z těchto předdefinovaných *views*, nebo si mohou vytvářet *views* vlastní. Jako *views* jsou definovány také prvky zajišťující rozložení, kterým se říká *stacks*. *Stacks* zajišťují rozložení prvků, a to vertikálně, horizontálně nebo zezadu dopředu (určující který prvek je „navrchu“). [36]

3.2.3 Modifikátory

SwiftUI umožňuje přizpůsobit vzhled a chování jednotlivých *views* pomocí modifikátorů. Mnoho modifikátorů je možno aplikovat na specifické druhy *views* či na specifické chování. Toto specifické chování jednotlivých prvků umožňuje například podmíněně a dynamicky skrýt *view* nastavením jeho průhlednosti. [37] Podobně jako u *views* má vývojář k dispozici předdefinované modifikátory, nebo má možnost vytvářet své vlastní. Mezi předdefinované modifikátory patří například *padding*, který přidává odsazení kolem *view*, *background*, který umožňuje nastavit barvu pozadí pro *view*, a *shadow*, který aplikuje stín kolem *view*.

3.2.4 Navigace mezi views

Pro navigaci mezi jednotlivými *views* slouží struktura nazývaná *NavigationStack*. *NavigationStack* je opět *view*, které zobrazuje počáteční *view* a umožňuje navigaci k dalším views.

Mimo jiné lze sledovat, jak klíčovou roli zastává *view* ve frameworku SwiftUI, jelikož vše, co se zobrazuje uživateli, je složeno z mnohonásobně zanořených *views*.

NavigationStack je založen na principu „poslední dovnitř, první ven“, anglicky *Last In First Out* (zkráceně LIFO) [38]. Tato datová struktura je běžně využívána pro udržování historie pohybu v aplikacích nebo na webových stránkách, jelikož zajišťuje, že prvky přidané poslední jsou odstraňovány první. Tím je zajištěn pohyb „zpět“ na nadřazené *views* ve SwiftUI. Uživateli je vždy prezentován poslední prvek nacházející se v datové struktuře. Tento přístup také umožňuje ovládání *views* programově. Uživateli se tak může zobrazit *view* na základě nějaké události nebo funkce v aplikaci, jako je například jeho výskyt na určité lokaci, dokončení nějakého procesu nebo změně dat ze serveru.

Deklarativní charakter SwiftUI je patrný i při použití *NavigationStack*. Pro navigaci mezi různými *views* aplikace stačí, když vývojář vytvoří odkaz na cílové *view*. Po aktivaci tohoto odkazu pak samotné přepnutí mezi *views* a další detaily, jako jsou zobrazení tlačítka pro návrat, gesta pro navigaci v aplikaci či animace, zajišťuje framework SwiftUI.

3.2.5 Předávání dat mezi views

Předávání dat mezi *views* je základním prvkem vývoje aplikací v rámci frameworku SwiftUI. Pro tento proces existuje hned několik způsobů, které jsou popsány v následujících podkapitolách. Předávání dat je nezbytné při vytváření dynamických uživatelských rozhraních a umožňuje synchronizaci dat mezi různými částmi aplikace.

Předávání dat pomocí properties

Nejzákladnější a nejjednodušší způsob předávání dat je přes tzv. *properties*. *Property* je proměnná, která je součástí struktury, třídy nebo výčtového typu. Tato metoda spočívá v definici proměnné u rodičovského *view*, která je poté předána do *view* potomka prostřednictvím konstruktoru (viz obrázek číslo 14).

```swift
struct ParentView: View {
    let message = "Hello, World!"

    var body: some View {
        ChildView(message: message)
    }
}

struct ChildView: View {
    let message: String

    var body: some View {
        Text(message)
    }
}
```

Obrázek 14 Předávání dat pomocí properties ve SwiftUI

Výhodou tohoto přístupu je jednoduchá implementace a pro malé aplikace s jednoduchými datovými závislostmi také přehlednost. Nevýhodou je, že tento přístup nepodporuje sledování změn dat v případě, že se data v rodičovské struktuře změní. Pro složité aplikace s mnoha *views* nebo komplikovanými datovými vazbami může být tento přístup méně srozumitelný.

Předávání dat pomocí @State a @Binding

Pro vytvoření synchronizovaného (sdíleného) stavu mezi dvěma *views* slouží *property wrappers* *@State* a *@Binding* (viz obrázek číslo 15). *Property wrapper* je aplikován na standardní proměnné, které jsou obaleny (anglicky *wrapped*) do jiného typu, a doplňuje jim dodatečné chování a vlastnosti pro získávání a nastavování hodnot. K deklaraci *property wrappers* slouží speciální anotace začínající symbolem @ (například *@State*, *@Binding* nebo *@ObservedObject*). [39]

Property wrapper *@State* umožňuje udržování a automatické aktualizace stavu v rámci jednoho *view*. Jakmile se hodnota proměnné označené anotací *@State* změní, SwiftUI tuto změnu detekuje a provede potřebné aktualizace. [39]

Pokud je potřeba sdílet stav mezi více *views*, používá se k tomu *property wrapper* *@Binding*. Tato anotace vytváří obousměrnou vazbu mezi *views*. To má za následek, že pokud dojde ke změně proměnné označené jako *@Binding* ve *view* potomka, změna bude zaznamenána také v rodičovském *view*. Tato změna je totiž reálně prováděna právě v proměnné označené

jako *@State*, která je hlavním zdrojem pravdy v programu (anglicky *source of truth*) a do *view* potomka je předávána reference na proměnnou označenou anotací *@State*.

```
1  struct ContentView: View {
2      @State private var count = 0
3
4      var body: some View {
5          VStack {
6              Text("Count: \(count)")
7              ChildView(count: $count)
8          }
9      }
10 }
11
12 struct ChildView: View {
13     @Binding var count: Int
14
15     var body: some View {
16         Button(action: {
17             count += 1
18         }) {
19             Text("Increment")
20         }
21     }
22 }
```

Obrázek 15 Použití property wrappers @State a @Binding

Tento přístup nabízí dvě hlavní výhody. První spočívá v automatické aktualizaci *views* v případě jakékoliv změny stavu. Druhá výhoda je obousměrná synchronizace dat mezi různými *views*, což zajišťuje jejich aktuálnost a konzistenci. Mezi nevýhody použití *@State* a *@Binding* se řadí složitější implementace ve srovnání s předáváním dat za použití *property* nebo důraz na správnou správu stavových proměnných a závislostí.

Předávání dat pomocí @ObservedObject a @StateObject

Použití *property wrappers @ObservedObject* a *@StateObject* umožňuje synchronizaci stavu mezi *views*. I když plní obdobný účel jako *property wrapper @State*, používají se pro složitější datové typy. Typickým použitím je sledování vlastní vytvořené třídy.

Rozdíl mezi *@ObservedObject* a *@StateObject* je ve vlastnictví. *@StateObject* je specializovaná verze *@ObservedObject* a obojí podléhá protokolu *ObservableObject*. Použití těchto *property wrappers* je řízeno pravidlem: *View*, které jako první vytváří sledovaný objekt, musí použít *@StateObject*. Tímto frameworku SwiftUI udává, že je dané *view* vlastníkem dat a je odpovědné za jejich udržování. Všechny ostatní *views* musí používat

@ObservedObject, aby SwiftUI mohlo sledovat změny objektu i když objekt nevlastní přímo. [39] Použití *@ObservedObject* a *@StateObject* je možno vidět na obrázku číslo 16.

```
class Model: ObservableObject {
    @Published var count = 0
}

struct ContentView: View {
    @StateObject var model = Model()

    var body: some View {
        VStack {
            Text("Count: \(model.count)")
            ChildView(model: model)
        }
    }
}

struct ChildView: View {
    @ObservedObject var model: Model

    var body: some View {
        Button(action: {
            model.count += 1
        }) {
            Text("Increment")
        }
    }
}
```

Obrázek 16 Použití property wrappers @StateObject a @ObservedObject

Zřejmou výhodou tohoto způsobu předávání dat je aktualizace *views* při změně stavu sledovaných dat. Nevýhodou použití *@StateObject* a *@ObservedObject* je složitější implementace a správa stavových proměnných, než tomu bylo u předešlých přístupů.

Předávání dat pomocí @EnvironmentObject

V případě, že je potřeba sdílet data na mnoha místech aplikace, je vhodné použít *property wrapper @EnvironmentObject*. Tento způsob umožňuje definovat data na globální úrovni z libovolného místa v aplikaci a na jiném libovolném místě tato data sledovat a automaticky aktualizovat po jejich změně (viz obrázek 17). [39]

```swift
class UserData: ObservableObject {
    @Published var username: String = ""
}

struct ContentView: View {
    @EnvironmentObject var userData: UserData

    var body: some View {
        VStack {
            Text("Welcome, \(userData.username)!")
            TextField("Enter username", text: $userData.username)
        }
    }
}
```

Obrázek 17 Použití property wrapper @EnvironmentObject

Tento přístup nabízí výhodu v podobě eliminace potřeby předávání dat do různých *views* prostřednictvím konstruktoru. Na druhé straně může použití *@EnvironmentObject* vést k nepřehlednosti kódu způsobené nepřímým přístupem k datům. Navíc vytváří implicitní závislosti, které mohou být těžko sledovatelné a mohou tak zvyšovat pravděpodobnost chyb.

3.2.6 Srovnání a integrace s UIKit

SwiftUI je relativně novým frameworkem pro vývoj uživatelských rozhraní pro aplikace na platformách společnosti Apple. Ve srovnání s tradičním frameworkem UIKit, který byl představen v roce 2007 [40] a podporuje zařízení již od verze iOS 2 [41], přináší SwiftUI řadu výhod i nevýhod.

UIKit je hojně používaný a zavedený framework pro vývoj aplikací na platformách iOS, iPadOS a tvOS. Vývojářům nabízí vysokou úroveň přizpůsobení a flexibility při návrhu uživatelského rozhraní. Na rozdíl od SwiftUI, kde Apple nedoporučuje specifickou architekturu, pro UIKit Apple doporučuje strukturovat kód dle architektury MVC (Model-View-Controller) [42].

Výhody použití SwiftUI

SwiftUI má mnoho výhod, jednou z nich je rychlejší a jednodušší vývoj uživatelských rozhraní. Díky dříve zmíněné deklarativní syntaxi mohou vývojáři ve SwiftUI stanovit vzhled a funkcionalitu uživatelského rozhraní, aniž by museli detailně popisovat postupy pro dosažení těchto cílů. [40]

Další výhodou je kvalitní integrace SwiftUI se Swift frameworkem. Swift framework je souhrnné označení pro soubor nástrojů a knihoven, které rozšiřují základní funkcionalitu

jazyka Swift a zajišťují jeho bezproblémovou integraci do produktů společnosti Apple. Pro vývojáře, kteří jsou již s těmito nástroji a knihovnami dobře obeznámeni, je přechod k práci s frameworkem SwiftUI snazší. [40]

Klíčovou výhodou je také kompatibilita s více platformami. SwiftUI umožňuje vytvářet uživatelská rozhraní pro všechny hlavní Apple zařízení, což vede k úspoře času i finančních nákladů při vývoji. SwiftUI také obsahuje mnoho předdefinovaných přechodů a animací ve srovnání s UIKit, kde by tyto prvky musely být složitěji definovány v kódu. [40]

Nevýhody použití SwiftUI

Jednou z hlavních nevýhod SwiftUI je omezená podpora pro starší verze iOS. Vývojáři, kteří chtějí podporovat zařízení s operačním systémem starší než iOS 13, musí použít jiný framework. SwiftUI je dostupné až od verze iOS 13 a novější. [40]

Dalším možným omezením ve srovnání s frameworkem UIKit je složitější přizpůsobení. Deklarativní přístup SwiftUI poskytuje vývojářům menší kontrolu nad tím, jak UI vypadá a jak se chová. To může být omezující faktor pro aplikace, které potřebují vysokou míru personalizace uživatelského rozhraní. Ze stejného důvodu mohou také vznikat výkonnostní problémy. [40]

Výhody použití UIKitu

Jednou z hlavních výhod frameworku UIKit je jeho dlouhá historie a s tím spojená vyspělost. Od začátku vývoje iOS byl UIKit používán k vytvoření řady známých aplikací. Díky tomu existuje rozsáhlá a aktivní komunita vývojářů, kteří jsou s tímto frameworkem obeznámeni. [40]

Další výhodou je větší výběr možností přizpůsobení, které UIKit nabízí. Díky imperativní syntaxi UIKitu mají vývojáři lepší kontrolu nad tím, jak UI vypadá a jak se chová. To může být klíčové pro aplikace, které vyžadují velmi přizpůsobené UI. [40]

UIKit je také známý pro svou vysokou výkonnost. Vývojáři mohou využít optimalizované grafické schopnosti UIKitu, což je důležité pro aplikace, které využívají animace, video nebo jiné grafické efekty. [40]

Nevýhody použití UIKitu

Jednou z hlavních nevýhod UIKitu je jeho obtížnost při učení. Vývojáři musí definovat sekvenci procedur pro vytvoření uživatelského rozhraní, protože UIKit používá imperativní syntaxi. [40]

Další nevýhodou UIKitu je, že vývoj uživatelského rozhraní může být časově náročnější. Při navrhování UI musí totiž vývojář napsat více kódu k dosažení v některých případech identických výsledků. To může zároveň vést k většímu počtu chyb. [40]

Důležité je také zmínit, že UIKit je určen pouze pro vývoj iOS, iPadOS nebo tvOS aplikací. SwiftUI framework byl navržen pro tvorbu aplikací pro všechny platformy od společnosti Apple.

Integrace SwiftUI a UIKitu

Frameworky SwiftUI a UIKit jsou kompatibilní a umožňují integraci a vzájemné propojení. Pro integraci SwiftUI do již existující UIKit aplikace slouží tzv. *hosting controllers*. *Hosting controller* obaluje sadu SwiftUI pohledů do formy, kterou je možné použít v aplikaci využívající UIKit. [43]

Naopak je také možné přidat UIKit *views* a *view controllery* do SwiftUI aplikace. Využívá se k tomu tzv. reprezentovatelný objekt, který obaluje určené *view* nebo *view controller* a usnadňuje komunikaci mezi obaleným objektem a dalšími SwiftUI *views*. [43]

Shrnutí

Při rozhodování mezi SwiftUI a UIKitem neexistuje univerzální řešení, které by vyhovovalo všem typům aplikací. Nejlepší volba závisí vždy na konkrétních potřebách projektu, protože každý framework má své výhody a nevýhody. Pokud aplikace vyžaduje vysoce přizpůsobitelné UI a rychlou grafiku nebo animace, může být UIKit lepší volbou. Na druhou stranu, pro aplikace, které potřebují fungovat na více Apple platformách, může být lepší volbou SwiftUI. [40]

3.3 Vision framework

Vision je framework vyvinutý společností Apple, který je podporován na všech operačních systémech této společnosti. Pro platformy iOS a iPadOS je konkrétně dostupný od verze 11. Tento framework poskytuje vývojářům soubor nástrojů pro aplikaci algoritmů počítačového vidění, které mohou být použity pro různé úkoly založené na vizuálních datech. Díky Vision frameworku je možné provádět pokročilé úlohy, jako jsou detekce textu, rozpoznávání obličejů a jejich rysů, čárových kódů, registrace obrazu a obecné sledování vlastností objektů. S využitím Vision lze také integrovat vlastní modely z Core ML pro klasifikaci a detekci objektů v různých scénářích použití. [44]

Ačkoli se práce s Vision frameworkem odlišuje pro každý nástroj obsažený v tomto frameworku, sdílí některé společné doporučení:

- Z důvodu snížení zbytečného výpočetního zatížení je doporučeno vytvořit všechny požadavky před dotazem na Vision a odeslat je v jediném volání. [45]
- Pro detekci více obrázků je doporučeno vytvořit požadavek pro každý obrázek zvlášť a spouštět je paralelně na více vláknech. Každý požadavek na zpracování obrázku přidává čas a zvyšuje vytížení paměti, a proto se nedoporučuje je spouštět v hlavním vlákně. [45]
- Základní mechanismus pro zpracování obrázků je schopen detekovat objekty napříč libovolným počtem obrázků, ale není navržen pro sledování objektů v průběhu času. Pro sledování konkrétních objektů (tj. pro sledování jejich pohybu nebo změn mezi snímky) je potřeba použít specifický požadavek zvaný *VNSequenceRequestHandler*. Tento požadavek je přizpůsoben pro sledování objektů v sekvenci snímků. [45]

3.3.1 Rozpoznání textu pomocí Vision frameworku

Jednou z mnoha funkcí Vision frameworku je detekce a rozpoznávání textu v obrázcích v několika jazycích. Tuto funkcionalitu je možno použít v reálném čase i bez přístupu k internetu. Ve všech případech se veškeré zpracování Vision frameworku děje na zařízení uživatele pro zlepšení výkonu a soukromí uživatele. [46]

Pro detekci textu ve Vision je potřeba si zvolit jeden z následujících způsobů:

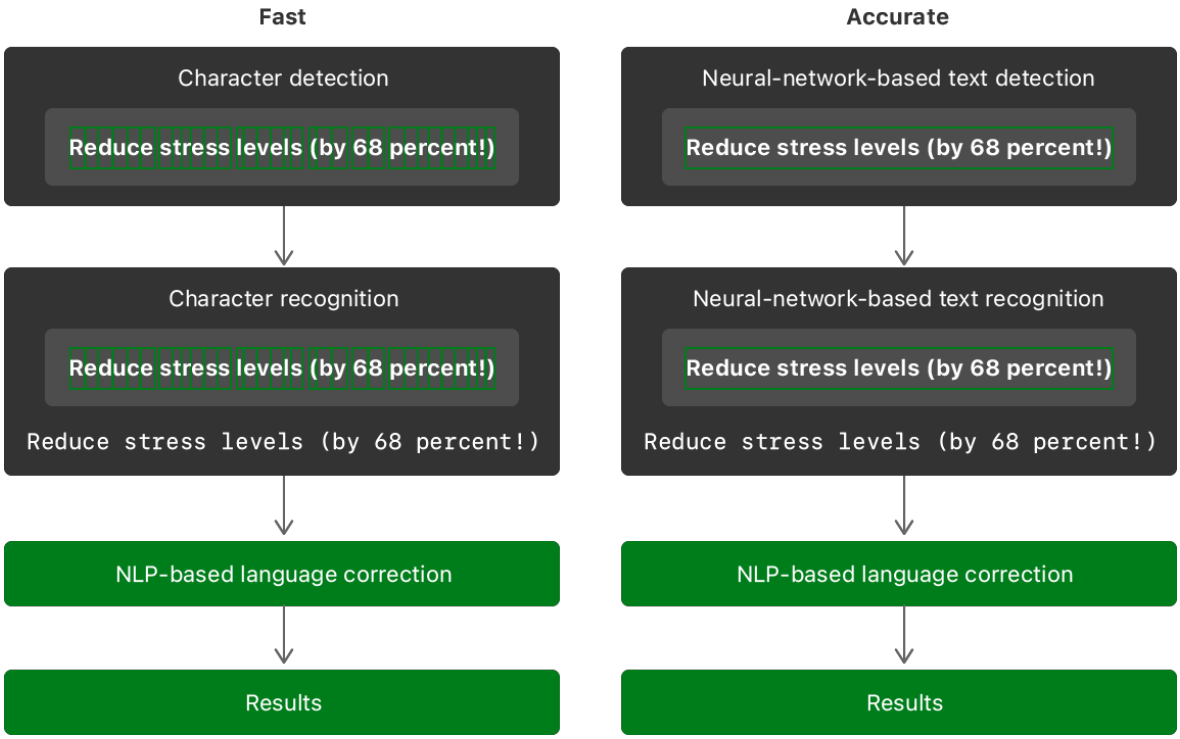
- **Fast**

Fast je způsob, který upřednostňuje rychlost detekce (*fast* – česky rychlý). Je při něm využito schopnosti detekce znaků k nalezení jednotlivých znaků a poté je použit malý model strojového učení k rozpoznání těchto znaků a slov. Tento přístup je svým principem podobný tradičním OCR řešením. [46]

- **Accurate**

Accurate je způsob, který se zvýšenou časovou náročností upřednostňuje přesnost detekce (*accurate* – česky přesný). Tento přístup používá neuronovou síť k nalezení textu ve formě řetězců a řádků a poté provádí další analýzu k nalezení jednotlivých slov a vět. Principem fungování je tak více v souladu s tím, jak text čtou lidé. [46]

Při zvolení *fast* i *accurate* způsobu detekce je následně možné přidat fázi korektury jazyka. Ta je založena na metodách z oblasti zpracování přirozeného jazyka pro minimalizaci nesprávného čtení. [46] Diagram popisující fungování obou přístupů je možno vidět na obrázku číslo 18 níže.



Obrázek 18 Diagram popisující fungování dvou přístupů rozpoznání textu ve Vision frameworku [46]

Vision poskytuje možnosti rozpoznávání textu pomocí třídy *VNRecognizeTextRequest*, která je specifickým typem požadavku určeným pro vyhledávání a extrakci textu z obrázku. Běžným postupem je vytvoření požadavku pro zpracování obrázku pomocí třídy *VNImageRequestHandler*, která zajistí vykonání požadavku *VNRecognizeTextRequest*. Tento požadavek rozpozná text v obrázku, který byl vstupním parametrem do této funkce v podobě třídy *CGImage*. [46] *CGImage* je třída reprezentující bitmapový obrázek nebo masku obrázku v kódu.

Na obrázku číslo 19 je zobrazen proces zpracování textu pomocí Vision frameworku. *VNRecognizeTextRequest* používá ve výchozím stavu způsob zpracování *accurate*. Zvolení metody *fast* je možno provést nastavením proměnné *recognitionLevel*, která patří třídě *VNRecognizeTextRequest*, na hodnotu *VNRequestTextRecognitionLevel.fast*.

```swift
// Get the CGImage on which to perform requests.
guard let cgImage = UIImage(named: "snapshot")?.cgImage else { return }

// Create a new image-request handler.
let requestHandler = VNImageRequestHandler(cgImage: cgImage)

// Create a new request to recognize text.
let request = VNRecognizeTextRequest(completionHandler: recognizeTextHandler)

do {
    // Perform the text-recognition request.
    try requestHandler.perform([request])
} catch {
    print("Unable to perform the requests: \(error).")
}
```

Obrázek 19 Ukázka zpracování textu ve Vision framework [46]

Poté, co je požadavek na zpracování dokončen, je volán uzavírací blok (anglicky *completion closure*) požadavku, který přijímá požadavek a případné chyby, které nastaly. Po provedení uzavíracího bloku je možno provést dotaz na tzv. *observations*. Pro anglický termín *observation* je v této práci používán český překlad postřeh. Postřehy z textového požadavku mají podobu třídy *VNRecognizedTextObservation*. V této podobě je každý detekovaný text umístěn do pole prvků. Postřeh zahrnuje informace o nalezeném textu a míru jistoty v přesnost rozpoznání. Dále je možné z postřehu získat také souřadnice nalezeného textu, které je možné použít například pro vykreslení obdélníků, které ohraničují detekovaný text. [46] Ukázku zpracování výsledků z rozpoznání a extrakce textu z postřehů je možno vidět na obrázku číslo 20.

```swift
func recognizeTextHandler(request: VNRequest, error: Error?) {
    guard let observations =
            request.results as? [VNRecognizedTextObservation] else {
        return
    }
    let recognizedStrings = observations.compactMap { observation in
        // Return the string of the top VNRecognizedText instance.
        return observation.topCandidates(1).first?.string
    }

    // Process the recognized strings.
    processResults(recognizedStrings)
}
```

Obrázek 20 Ukázka zpracování nalezeného textu ve Vision framework [46]

Vybraná metoda zpracování (rychlá nebo přesná) spolu s použitou revizí API určuje, které jazyky jsou algoritmy pro rozpoznání jazyka podporovány. Pokud není specifikováno jinak, tak je jako preferovaný jazyk automaticky použita angličtina. V případě, že je specifikováno více jazyků, tak jejich pořadí určuje jejich relativní důležitost. [46]

Pro korekci chyb v rozpoznávání je možné povolit korekci jazyka a nastavit vlastní slova, která mají přednost při korekci. Tato vlastnost umožňuje přizpůsobit rozpoznávání textu odborným termínům a slovníkům pro specifické oblasti, jako jsou například medicína nebo technika. [46]

3.3.2 Předinstalované iOS aplikace využívající Vision framework

Tato podkapitola se zaměřuje na předinstalované aplikace v operačním systému iOS, které využívají technologii Vision framework pro detekci a digitalizaci textu. Konkrétně jsou popsány aplikace Fotoaparát, Fotky a Poznámky, jež tuto technologii využívají k usnadnění práce s textem na fotografiích, videích nebo skenovaných dokumentech.

V rámci aplikace Fotoaparát umožňuje framework Vision detekovat text v reálném čase při snímání obrazu. V okamžiku, kdy je text identifikován v záběru, jej aplikace automaticky rozpozná a poskytne uživateli možnosti pro další interakci s detekovaným textem. Například kopírování, výběr specifických částí, vyhledání v internetovém prohlížeči, překlad do jiného jazyka nebo sdílení.

Aplikace Fotky, na druhou stranu, implementuje framework Vision k detekci a rozpoznání textu v již uložených obrázcích. Aplikace nabízí srovnatelné možnosti interakce s detekovaným textem, jako aplikace Fotoaparát. Uživatelé tak mají schopnost efektivně vyhledávat fotografie obsahující specifický text, jako jsou například názvy míst. V případě detekce textu ve formátu datumu nebo času, aplikace nabízí další akce, jako jsou vytvoření události, nastavení připomínky nebo zobrazení daného termínu v kalendáři.

Další oblastí, kde technologie Vision nachází uplatnění, je skenování a digitalizace dokumentů. Konkrétně předinstalovaná aplikace Poznámky poskytuje uživatelům možnost rychlého a efektivního převodu fyzických dokumentů do digitální podoby. Celý proces digitalizace je automatizovaný: aplikace detekuje okraje dokumentu a následně ořízne a optimalizuje snímek tak, aby dosáhl kvality tradičně skenovaných dokumentů. Dále aplikace disponuje funkcí rozpoznání textu na skenovaných dokumentech, čímž uživatelům umožňuje další možnosti práce s digitalizovaným textem.

Jak je patrné z předinstalovaných aplikací využívajících Vision, tyto technologie přináší významné výhody v oblasti digitalizace. Ať už se jedná o detekci textu v reálném čase při focení, rozpoznání textu na již pořízených fotografiích, nebo skenování a digitalizaci fyzických dokumentů, Vision framework tyto úkony usnadňuje a zvyšuje efektivitu práce s daty.

Je však důležité poznamenat, že funkcionalita v zabudovaných aplikacích je primárně určena pro základní operace, jako je kopírování nebo sdílení textu, a proto není určena pro hromadné nebo automatizované sbírání dat.

3.4 Core Data

Core Data je framework vyvíjen společností Apple pro správu modelové vrstvy aplikací pro Apple platformy. Umožňuje snadné ukládání, načítání a spravování dat v aplikaci. Data uložená pomocí Core Data je také možné synchronizovat mezi více zařízeními prostřednictvím produktu CloudKit. [47]

Jednou z hlavních vlastností frameworku Core Data je abstrakce mapování objektů do úložiště, což zjednodušuje ukládání a získávání dat bez nutnosti přímého zásahu do databáze. Další výhodou je přítomnost správce vrácení změn (anglicky *undo manager*), který sleduje provedené změny a umožňuje jejich individuální, skupinové nebo celkové vrácení. Tímto způsobem se snadno implementuje podpora pro vrácení a opětovné provedení akcí (anglicky *undo* a *redo*) ve vyvíjené aplikaci. Framework také nabízí podporu pro asynchronní zpracování dat na pozadí, což umožňuje provádění potenciálně blokujících operací, jako je například vytvoření objektů z dat ve formátu JSON bez negativního vlivu na uživatelské rozhraní. Výsledky těchto operací lze následně ukládat do mezipaměti nebo paměti, což minimalizuje nutnost časté komunikace se serverem. Navíc Core Data poskytuje mechanismy pro verzování datového modelu a migraci uživatelských dat, což umožňuje správu a aktualizaci dat v různých verzích aplikace. Tímto je zajištěna dlouhodobá udržitelnost a flexibilita datového modelu. [47]

3.5 Xcode

Xcode IDE (anglická zkratka pro *Integrated Development Environment*) je integrované vývojové prostředí vyvíjené společností Apple jako hlavní nástroj pro tvorbu software pro jejich platformy. Xcode je zdarma ke stažení prostřednictvím obchodu App Store na operačním systému macOS. Mezi podporované programovací jazyky v Xcode patří

kromě Swift také například C, C++ nebo Objective-C. Vývojové prostředí Xcode bylo oficiálně vydáno roku 2003, ale jako samotný operační systém macOS má i Xcode původ již v technologiích od společnosti NeXT (z roku 1992). Před vydáním Mac OS X se sada vývojových nástrojů nazývala Developer Tools. [48]

Xcode má zabudovanou podporu pro správu zdrojového kódu, a to konkrétně distribuovaný systém pro verzování a správu kódu Git. Vytvořené Git repozitáře mohou být následně nahrány na platformy pro správu zdrojového kódu, jako jsou například GitHub, Bitbucket nebo GitLab. Uživatel má tak možnost provádět operace jako *commit*, *push* a *pull* přímo z vývojového prostředí.

Kromě manuálního psaní zdrojového kódu Xcode nabízí několik nástrojů, které využívají grafického uživatelského rozhraní (anglicky GUI – *Graphical User Interface*) pro tvorbu a úpravy částí aplikace. Tímto způsobem je vývojářům umožněno snadno a efektivně přidávat a upravovat prvky uživatelského rozhraní bez potřeby psaní zdrojového kódu. Tyto nástroje umožňují vývojářům pracovat efektivněji a rychleji, a to díky možnosti provádět úpravy přímo v grafickém uživatelském rozhraní, aniž by museli psát zdrojový kód.

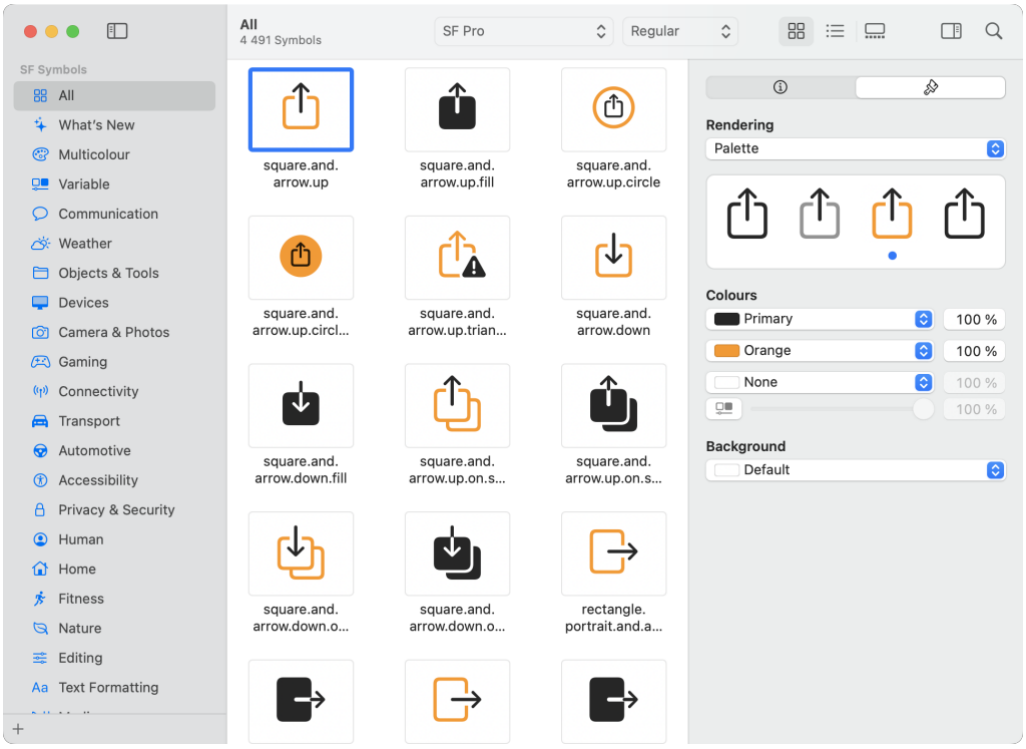
3.6 SF Symbols

SF Symbols je knihovna ikonografie navržena tak, aby se dokonale integrovala s písmem San Francisco, systémovým písmem pro platformy Apple. Obsahuje přes 4400 symbolů a každý z nich je obsažen v devíti váhách a třech měřítkách. Tyto symboly je možno exportovat a dále upravovat. [49] Kromě základních vah a měřítek je také možné měnit vykreslovací režimy (monochromatický, hierarchický, paletový a vícebarevný), variabilní barvy (v závislosti na určených prahových hodnotách) nebo varianty jednotlivých symbolů dle výplně. Knihovna SF Symbols prošla několika aktualizacemi a v současnosti (květen 2023) je nejnovější verze SF Symbols verze 4.

Použití sjednocených symbolů má za následek snadnost a intuitivnost při interakci mezi zařízeními, službami a aplikacemi. Symboly z SF Symbols jsou často využívány například v navigačních lištách, záložkách nebo tlačítkách.

SF Symbols je možno stáhnout jako samostatnou aplikaci pro operační systém macOS. Aplikace poskytuje nástroje pro vyhledávání a procházení symbolů, zobrazování informací o symbolu a jeho vlastnostech a exportování symbolů pro použití v jiných aplikacích. Dále

aplikace umožňuje vývojářům upravovat vlastnosti symbolu, jako jsou váha a měřítko, a zobrazovat náhledy symbolu v různých režimech vykreslování.



Obrázek 21 Rozhraní aplikace SF Symbols

Druhým způsobem přístupu k SF Symbols je použití nástroje Library v Xcode. Library poskytuje sbírku předdefinovaných prvků uživatelského rozhraní a modifikátorů, včetně SF Symbols, které je touto formou velice snadné přidat do aplikace. Použití Library v Xcode umožňuje vývojářům snadno přidávat symboly z knihovny SF Symbols, bez nutnosti psát kód nebo opouštět prostředí Xcode.

3.7 Excalidraw

Excalidraw je bezplatný program s otevřeným zdrojovým kódem, dostupný prostřednictvím webového prohlížeče. Jeho hlavní funkcí je neomezená virtuální tabule, která umožňuje psát a kreslit pomocí nástrojů, které napodobují ručně kreslené čáry a písmo. Tento nástroj se vyznačuje svou jednoduchostí a zachováním přirozeného vzhledu kreseb, což ho činí ideální volbou pro tvorbu nákresů a konceptuálního návrhu uživatelského rozhraní. Excalidraw rovněž podporuje spolupráci v reálném čase a zajišťuje bezpečnost dat díky *end-to-end* šifrování. [50] Jeho schopnost napodobit kresbu tužkou a podporovat tvůrčí proces přispívají k efektivnímu vytváření návrhů.

3.8 Figma

Figma je grafický nástroj specializovaný na vektorovou grafiku. Tento nástroj je široce používán při navrhování uživatelských rozhraní díky své flexibilitě a intuitivnímu uživatelskému rozhraní.

Jednou z unikátních schopností Figmy je, že tento nástroj lze používat přímo ve webovém prohlížeči, což zvyšuje dostupnost a umožňuje snadnou spolupráci mezi uživateli. Kromě toho jsou pro platformy macOS a Windows k dispozici také dedikované aplikace. Pro zobrazení náhledu na mobilních zařízeních nabízí Figma aplikace pro operační systémy iOS a Android. [51]

Figma je nabízena ve třech verzích – bezplatné, profesionální placené a placené pro organizace. Pro účely této práce, konkrétně pro tvorbu drátěných modelů, byla využita bezplatná verze. Tato verze poskytuje dostatečný soubor nástrojů a funkcí, jejíž omezení nijak neomezovala proces vytváření drátěných modelů pro tento projekt. [51]

II. PRAKTICKÁ ČÁST

4 POPIS A POŽADAVKY PROTOTYPOVÉ APLIKACE

V moderní společnosti je nárůst množství digitálních dat a jejich využití obrovský, což vede k vysoké poptávce po efektivních, přesných a cenově dostupných metodách digitalizace. Tento požadavek je ještě výraznější v prostředích, kde přetrvává použití tradičních metod sběru dat, jako je ruční přepis a zadávání dat do informačních systémů. Tyto techniky jsou nejen časově náročné, ale také náchylné k chybám, což je činí neefektivními a nevyhovujícími pro současné rychle se měnící technologické trendy.

V rámci praktické části práce je navržen, implementován a testován prototyp aplikace, který aktivně využívá nejmodernější technologie a potenciál mobilních platforem od společnosti Apple za účelem řešení konkrétní problematiky digitalizace dat ze zařízení, kde použití tradičních metod není vhodné. V této první části práce je aplikace definována formou popisu, cílů a požadavků. Toto vše s cílem zjistit, zda a do jaké míry lze tyto technologie efektivně využít k řešení významného problému v oblasti rozpoznání a digitalizace textu.

4.1 Popis aplikace

Aplikace představuje pokročilou technologii rozpoznávání a extrakce textu, a poskytuje tak uživatelům robustní nástroj pro efektivní a rychlou digitalizaci textových dat z fotografií. Klíčovou vlastností aplikace je schopnost uživatelů popsat strukturu dat k digitalizaci. Po nahrání fotografie aplikace tato data zpracuje dle uživatelem definované struktury a automaticky přiřadí správné informace z fotografie k očekávaným hodnotám.

V aplikaci má uživatel možnost pořizovat fotografie nebo nahrávat již existující obrázky, které obsahují text určený k digitalizaci. Aplikace využívá kombinaci Vision frameworku a vlastních funkcí k automatickému detekování a rozpoznání textu na obrázcích.

Po úspěšném zpracování a digitalizaci textu je možné tyto digitalizované informace ukládat jako data. Aplikace také poskytuje možnost exportování digitalizovaných textových dat do souboru ve formátu CSV (anglická zkratka pro *comma-separated values*). Tímto uživatelům přináší flexibilitu při sdílení a dalším využití těchto informací.

CSV soubory s digitalizovanými daty je možné snadno sdílet s jinými aplikacemi nebo lidmi, což poskytuje uživatelům možnost efektivně využívat digitalizovaná data z fotografií. Tímto způsobem je uživatelům poskytnuta flexibilita a možnost efektivně využívat digitalizovaná data z fotografií.

4.2 Cíle aplikace

- Poskytnout uživatelům jednoduchý a intuitivní způsob digitalizace textových dat z fotografií s možností samostatně definovat strukturu digitalizovaných dat.
- Zrychlit a zjednodušit proces převodu textu z obrazové podoby do digitálního formátu.
- Umožnit uživatelům efektivně upravovat a spravovat digitalizovaný text.
- Poskytnout možnost exportovat digitalizovaná data do formátu CSV pro další zpracování a sdílení.
- Zlepšit produktivitu uživatelů a usnadnit proces digitalizace dat v průmyslovém nebo domácím prostředí.

4.3 Funkcionální požadavky

V tabulce číslo 1 jsou popsány funkcionální požadavky na aplikaci.

Tabulka 1 Funkcionální požadavky na aplikaci

ID požadavku	Popis požadavku
F001	Uživatel bude mít možnost v aplikaci fotografovat nebo nahrávat obrázky obsahující text pro jejich zpracování.
F002	Aplikace bude detekovat a extrahovat text z nahraných fotografií pomocí Vision frameworku.
F003	Uživatel bude mít možnost upravit detekovaný text před jeho uložením.
F004	Aplikace bude detekovaný text ukládat do databáze pomocí Core Data frameworku.
F005	Aplikace bude umožňovat uživateli exportovat detekovaný text do souboru ve formátu CSV.
F006	Exportovaný CSV soubor bude možné uložit na zařízení uživatele nebo sdílet přes systémové okno pro sdílení.

4.4 Nefunkcionální požadavky

V tabulce číslo 2 jsou popsány nefunkcionální požadavky na aplikaci.

Tabulka 2 Nefunkcionální požadavky na aplikaci

ID požadavku	**Popis požadavku**
N001	Aplikace bude reagovat na uživatelské interakce rychle, s průměrnou dobou odezvy kratší než 500 milisekund.
N002	Aplikace bude schopna na testovaném zařízení zpracovat a extrahovat text ze snímku do 5 sekund od nahrání fotografie.
N003	Velikost aplikace (nepočítaje databázi) nebude více než 100 MB, aby minimalizovala využití paměti na zařízení.
N004	Aplikace bude navržena tak, aby uživatelé mohli efektivně využívat její funkce bez potřeby formálního školení. K efektivnímu používání budou stačit nápovědy v aplikaci.
N005	Pravděpodobnost chyby při detekci a extrakci textu bude u předem definovaných testovacích fotografií menší než 3 % z důvodu minimalizace chyb ve výsledcích.

5 TVORBA PROTOTYPOVÉ APLIKACE

Tato kapitola je zaměřena na tvorbu prototypové aplikace, která slouží jako demonstrace pokročilé technologie rozpoznávání textu a jeho extrakce. Cílem této kapitoly je podrobně popsat proces návrhu a implementace aplikace.

Nejprve je věnována pozornost návrhu uživatelského rozhraní, kde jsou definovány a vysvětleny základní pojmy spojené s interakcí uživatele s aplikací. Jsou zkoumány koncepty, které jsou nezbytné pro správné pochopení a ovládání aplikace. Dále je představen nákres aplikace, který vizualizuje strukturu a uspořádání uživatelského rozhraní.

Následuje prezentace vytvořeného drátěného modelu aplikace, který poskytuje podrobnější pohled na uživatelské rozhraní. Tento model představuje vizuální reprezentaci jednotlivých obrazovek aplikace a jejich vzájemného propojení. Během tvorby drátěných modů byly zkoumány různé scénáře a funkčnosti aplikace, které jsou v nich zachyceny.

Druhá část kapitoly je zaměřena na samotnou implementaci aplikace. Nejprve je provedena inicializace projektu, kde je nastaveno prostředí a připraveny potřebné zdroje. Následně je provedena konfigurace grafických zdrojů, která zahrnuje správu ikon, barev a dalších vizuálních prvků v aplikaci.

Poté je přistoupeno k vytvoření datového modelu, který slouží pro uchovávání a manipulaci s daty v aplikaci. Je definována struktura a vztahy mezi jednotlivými entitami, které jsou v aplikaci využívány.

Důležitou částí implementace je také vytvoření samotného uživatelského rozhraní aplikace. V této části je věnována pozornost tvorbě obrazovek, rozvržení prvků a implementaci interakcí mezi uživatelem a aplikací.

Na závěr této kapitoly je přistoupeno k důležitým částem kódu, které představují nejdůležitější prvky implementace. Je zde popsáno, jak jsou řešeny konkrétní funkcionality aplikace, a jaké techniky a postupy byly využity.

5.1 Návrh uživatelského rozhraní

Tato podkapitola je zaměřena na návrh uživatelského rozhraní prototypové aplikace. Cílem je vytvořit intuitivní a uživatelsky přívětivé rozhraní, které umožní uživatelům snadno interagovat s aplikací a efektivně využívat její funkcionality.

Při tvorbě uživatelského rozhraní je důležité začít tvorbou představy o tom, jak by aplikace mohla a měla fungovat. V této fázi projektu bývají v praxi změny nejlevnější a je snadné rychle reagovat na připomínky a požadavky uživatelů. Z toho důvodu se běžně začíná vytvářením nákresů a konceptuálního návrhu. Pro tvorbu nákresů byl použit software Excalidraw, který byl podrobněji popsán v teoretické části této práce.

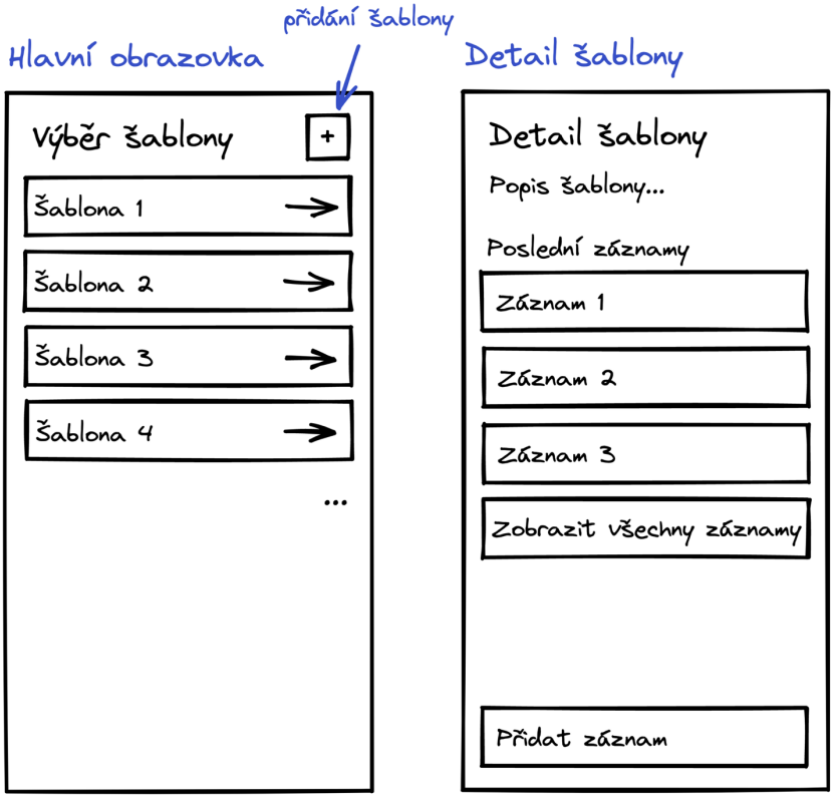
5.1.1 Definice základních pojmů v uživatelském rozhraní aplikace

Po spuštění aplikace je nezbytné umožnit uživatelům specifikovat data, která si přejí digitalizovat. Tyto definice jsou v kontextu aplikace označovány jako **šablony**. Šablona obsahuje informace o tom, jaké hodnoty jsou očekávány v konkrétních situacích. Jedno odečtení očekávaných hodnot v časovém okamžiku tvoří jeden **záznam**. Všechny záznamy poté tvoří sbíraná data, jenž má aplikace sbírat v co nejjednodušší a nejvíce automatizované podobě.

5.1.2 Nákres aplikace

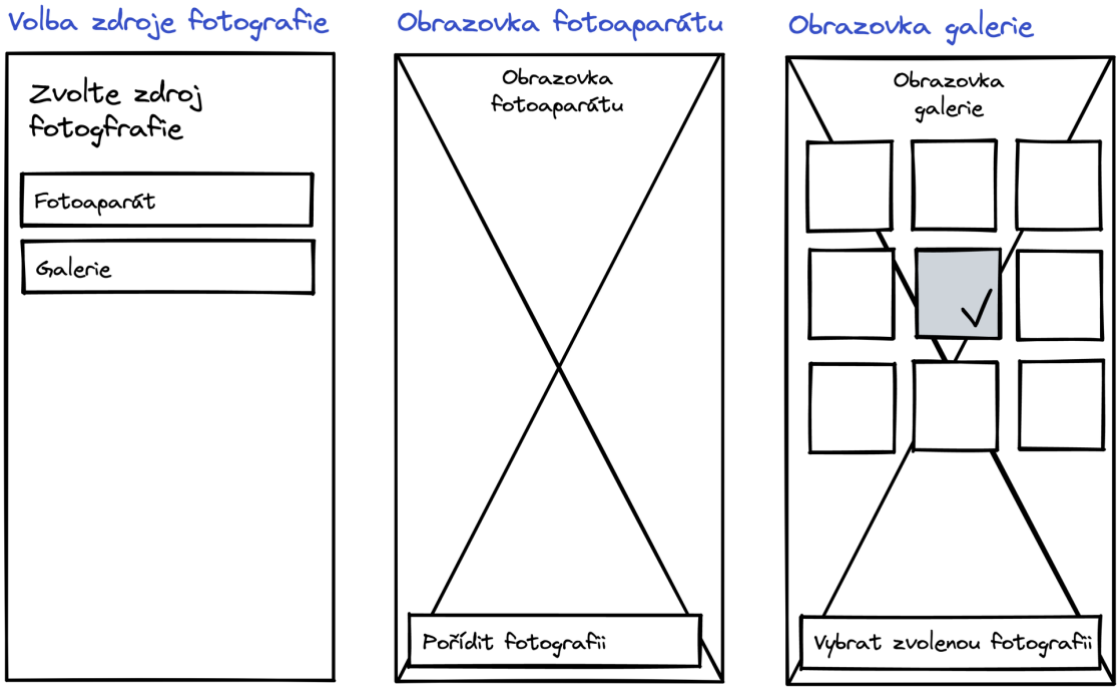
Nákresy slouží během vývoje aplikace k vizualizaci hlavních částí uživatelského rozhraní, návrhu uspořádání obrazovek a upřesnění základních prvků uživatelského rozhraní. Tímto způsobem lze získat lepší přehled o struktuře a přehlednosti definovaného rozhraní. V této fázi je důležité vzít v úvahu požadavky na aplikaci a zajistit, aby byly všechny zohledněny.

V levé části obrázku číslo 22 je viditelný návrh hlavní obrazovky aplikace, která umožňuje uživateli procházet všechny dostupné šablony, zobrazit si jejich detail nebo vytvořit zcela novou šablonu. Pravá polovina obrázku číslo 22 zobrazuje detail vybrané šablony. Tato část obrazovky uživateli poskytne podrobné informace a možnosti týkající se dané šablony. Uživatel má možnost nahlédnout na specifické detaily šablony, mezi které patří název, popis a další relevantní informace. Kromě toho je zde také možnost prohlížet již vytvořené záznamy uložené v dané šabloně. Dále je na této obrazovce umožněno přidávat nové záznamy.



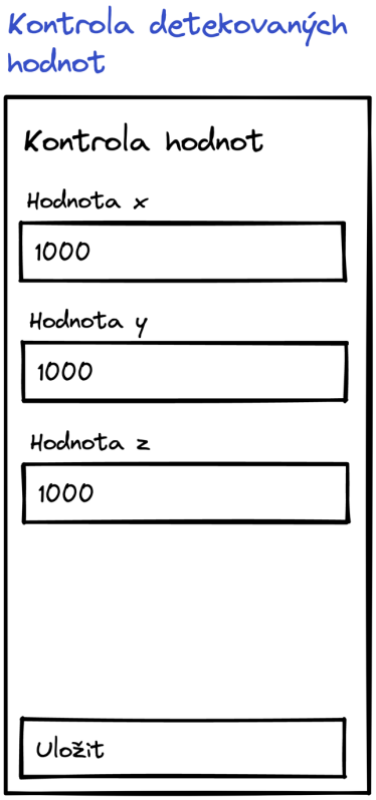
Obrázek 22 Nákres hlavní obrazovky aplikace a obrazovky pro zobrazení detailu šablony

Obrázek číslo 23 představuje tři různé obrazovky související s funkcionalitou nahrávání fotografií v aplikaci. Na první obrazovce v levé části je uživateli poskytnuta možnost vybrat jakým způsobem chce nahrát fotografii. Jsou zde zobrazeny dvě možnosti – vyfocení fotografie pomocí fotoaparátu nebo nahrání fotografie z galerie. Uživatel si může vybrat preferovaný způsob nahrávání dle svých potřeb a preferencí. Prostřední obrazovka tohoto obrázku symbolicky znázorňuje zabudované rozhraní fotoaparátu, čímž signalizuje, že aplikace podporuje přímé použití fotoaparátu pro vytváření fotografií. Tato funkce umožňuje uživatelům okamžitě zachytit fotografii bez nutnosti přepínání na zabudovanou aplikaci fotoaparátu. Poslední obrazovka na tomto obrázku (v pravé části) zobrazuje nákres přidání fotografie z galerie telefonu. Uživatel má možnost přistoupit ke své galerii fotografií a vybrat zde fotografii pro nahrání do aplikace. Obrazovka pro výběr fotografie z galerie využívá zabudované rozhraní, které je součástí operačního systému iOS, a odpovídá osvědčeným postupům pro intuitivní ovládání aplikace.



Obrázek 23 Nákres UI pro přidání fotografie do aplikace

Na obrázku číslo 24 je k nahlédnutí obrazovka, která umožňuje kontrolu a uložení detekovaných hodnot z fotografie. Uživatel má na této obrazovce možnost zkontrolovat aplikací předvyplněné digitalizované hodnoty a v případě potřeby je upravit. Důvodem pro úpravu hodnot může být chybná detekce textu na fotografiích z různých důvodů, jako jsou stíny na fotografii, prach na snímaném displeji nebo jiné faktory, které mohou ovlivnit správnost detekce. Aplikace tak automaticky předvyplňuje hodnoty pro usnadnění procesu digitalizace, avšak stále poskytuje možnost provést manuální úpravy, aby byla zajištěna správnost digitalizovaných dat. Po zkontrolování a případných úpravách má uživatel možnost tyto hodnoty uložit. Uživatel má díky těmto funkcím plnou kontrolu nad detekovanými hodnotami.



Obrázek 24 Nákres UI pro kontrolu a uložení detekovaných hodnot

Při tvorbě nákresu byl kladen důraz na nalezení optimálního postupu pro proces digitalizace dat a zajištění maximální jednoduchosti používání pro uživatele. Nákresy slouží jako důležitý podklad pro vytvoření detailnějšího drátěného modelu, ve kterém jsou podrobněji popsány jednotlivé obrazovky aplikace a jejich vzájemné interakce. Tím je zajištěno, že výsledný produkt bude co nejpřesněji odpovídat požadavkům a očekáváním, a připravuje se tak kvalitní základ pro další fáze vývoje.

5.1.3 Drátěný model aplikace

V této podkapitole jsou prezentovány drátěné modely jednotlivých obrazovek aplikace a jejich vzájemné interakce, které byly vytvořeny v aplikaci Figma.

Drátěný model aplikace představuje klíčový prvek v procesu návrhu uživatelského rozhraní. Jedná se o vizuální reprezentaci struktury a interakcí v aplikaci pomocí jednoduchých (většinou nebarevných) geometrických prvků a textu, které zachycují její jednotlivé obrazovky a navigační toky. Tomuto typu drátěných modelů se také anglicky říká *Low fidelity wireframe* (zkráceně *Lo-Fi wireframe*), což je anglický termín pro drátěné modely s nízkou grafickou věrností. [52]

Cílem drátěného modelu je poskytnout přehled o uspořádání aplikace a navigačních prvcích. Pomáhá lépe porozumět toku informací, umístění prvků na obrazovkách a způsobu jejich interakce. Je možno konstatovat, že drátěný model slouží jako prostředek pro propojení abstraktního konceptu aplikace s jejím konkrétním vizuálním provedením. Díky tomu je možné efektivněji sdílet a komunikovat návrhy, upřesnit požadavky a získat zpětnou vazbu.

Drátěné modely aplikace jsou přizpůsobeny frameworku SwiftUI, což zajišťuje konzistenci a snadnou implementaci uživatelského rozhraní. Tyto modely jsou navrhovány v souladu s Apple Human Interface Guidelines, které stanovují směrnice pro vytváření uživatelsky přívětivých aplikací na platformě iOS. [53]

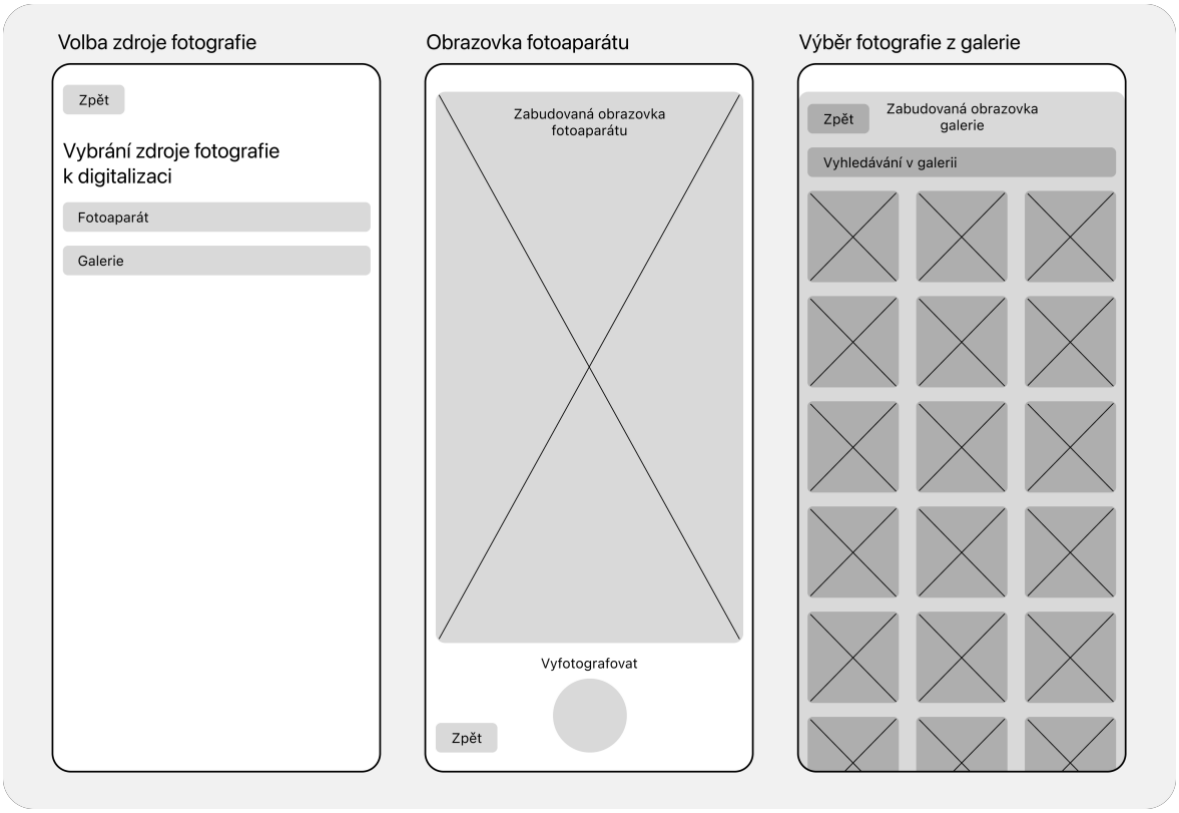
Po spuštění aplikace je uživateli prezentována hlavní obrazovka, na které se nachází jeho již vytvořené šablony. Tato obrazovka je zobrazena na levé straně obrázku číslo 25. Dále je na této obrazovce možnost dělat dvě základní operace. Jako první může uživatel vytvořit šablonu pomocí tlačítka *Přidat šablonu* nebo mazat šablony pomocí tlačítka *Upravit šablony*. Stisk tohoto tlačítka spustí tzv. *edit mode* z frameworku SwiftUI pro zobrazený seznam šablon. Tento režim je očekávaným chováním pro aplikace na platformě iOS pro položky zobrazené v seznamu a umožňuje smazání jednotlivých položek (včetně smazání pomocí gesta tažením prstu na položce z pravé strany do levé).

Po stisknutí konkrétní šablony ze seznamu je uživatel přesměrován na obrazovku detailu šablony, kterou je možno vidět v pravé části obrázku číslo 25. Z této obrazovky je možno provést šest akcí:

- vrátit se zpět na domovskou obrazovku,
- exportovat data šablony (tlačítkem *Sdílet*),
- provést úpravu šablony,
- zobrazit jeden z pěti posledních záznamů přidaných do šablony,
- zobrazit všechny záznamy pro danou šablonu,
- přidat nový záznam.

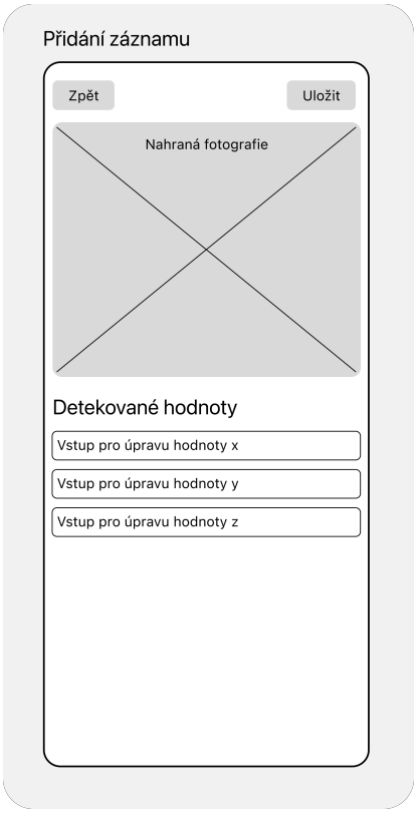
Obrázek 25 Drátěný model hlavní obrazovky a detailu šablony

Proces nahrání fotografie je zobrazen na obrázku číslo 26. Na tomto obrázku je v levé části obrazovka, jež dává uživateli na výběr, jaký je zdroj fotografie k následné digitalizaci. Fotografie může být přidána do aplikace buď pořízením fotografie z fotoaparátu nebo vybráním fotografie z galerie. Obrazovka fotoaparátu je zobrazena v centrální části, zatímco obrazovka pro výběr fotografie z galerie je zobrazena v pravé části obrázku číslo 26. Tyto obrazovky využívají standardní zabudované *views* pro příslušné operace.



Obrázek 26 Drátěné modely pro nahrání fotografie do aplikace

Poslední obrazovkou navrženou na základě předem vytvořených nákresů je obrazovka pro přidání záznamu. Tuto obrazovku je možné vidět na obrázku číslo 27. Při přidávání záznamu byl přidán náhled na nahranou fotografii, který zlepšuje UX (zkratka pro *user experience*, česky uživatelský prožitek). V případě, že by uživatel nahrál špatnou fotografii a dostal tak nevalidní detekované hodnoty, byl by na takovou situaci upozorněn prostřednictvím náhledu. Na této obrazovce je také možné hodnoty upravit v případě, že došlo k chybě při detekci. Po stisknutí tlačítka *Uložit* je záznam uložen a uživatel přesměrován na domovskou obrazovku.

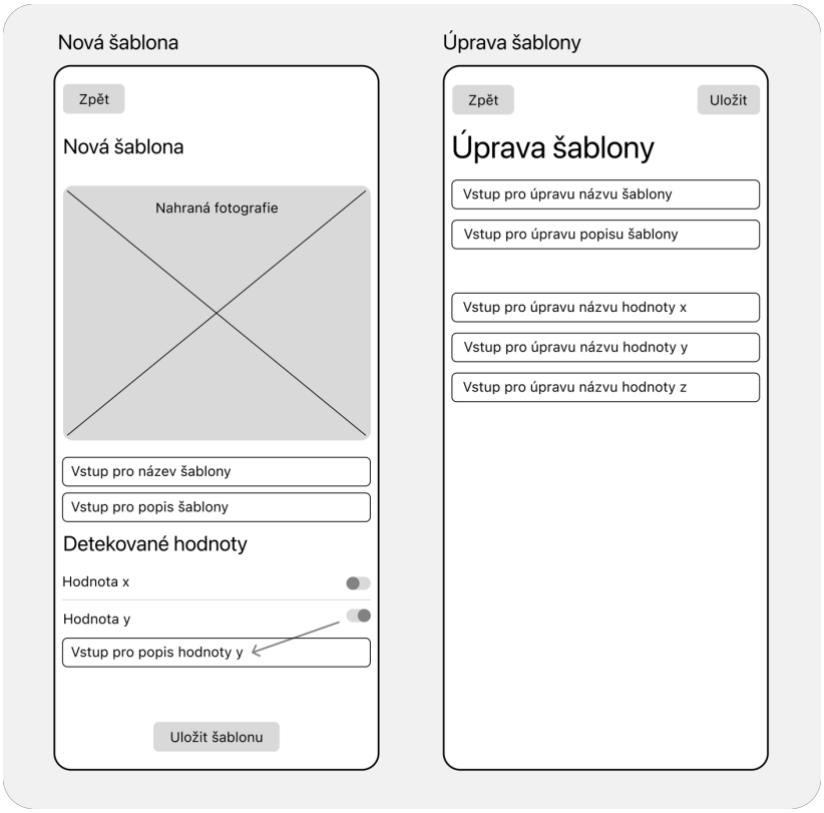


Obrázek 27 Drátěný model obrazovky pro přidání záznamu

Jak bylo již uvedeno v kapitole „Nákres aplikace“, drátěný model poskytuje podrobnější pohled na strukturu aplikace. Tento detailnější pohled vedl k vytvoření několika nových obrazovek, které rozšiřují základní funkčnost aplikace.

Jednou z nově přidaných obrazovek je obrazovka pro vytvoření nové šablony, kterou je možno vidět v levé části obrázku číslo 28. Po stisknutí tlačítka pro vytvoření nové šablony na hlavní obrazovce aplikace je uživatel znovu proveden pomocí již existujících obrazovek procesem nahrání fotografie. Z toho vyplývá, že tyto obrazovky jsou v aplikaci použitelné opakovaně. Po nahrání fotografie z ní aplikace detekuje a extrahuje text a zobrazí uživateli formulář pro vytvoření nové šablony. Formulář obsahuje pole pro vyplnění základních informací o šabloně, jako jsou název a popis a také rozhraní pro výběr textových hodnot, které uživatel chce digitalizovat. Tato část rozhraní je v aplikaci naprosto stěžejní, jelikož umožňuje uživateli definovat, která hodnota je pro něj důležitá pro digitalizaci pomocí základního přepínače. V případě, že je přepínač přepnut do zapnuté polohy a jeho pozadí je zbarveno zeleně, zobrazí se vstupní pole, kde uživatel musí zadat pod jakým názvem si přeje tuto hodnotu ukládat. Tento název je důležitý, aby bylo možné později rozpoznat, co odečtená hodnota znamená.

Další přidanou obrazovku je možné vidět v pravé části obrázku číslo 28, která zprostředkovává možnost upravovat parametry šablony.



Obrázek 28 Drátěný model obrazovek pro vytvoření nové šablony a úpravy šablony

Poslední přidanou obrazovkou je ta pro exportování dat z aplikace (viz obrázek číslo 29). K tomuto účelu slouží výchozí systémová obrazovka pro sdílení, která je používána v aplikacích zabudovaných v systému iOS.

Obrázek 29 Drátěný model obrazovky pro exportování dat z aplikace

Návrh drátěných modelů poskytuje důležitý přehled o struktuře aplikace a vzájemných interakcích, což je podstatným krokem před samotnou implementací. Tyto modely slouží jako cenný referenční bod pro další vývoj a umožňují přesnější definici datového modelu v Core Data a návrh uživatelského rozhraní ve SwiftUI. Díky takto realizovanému návrhu je možné postupovat efektivněji a s větší přesností při implementaci aplikace.

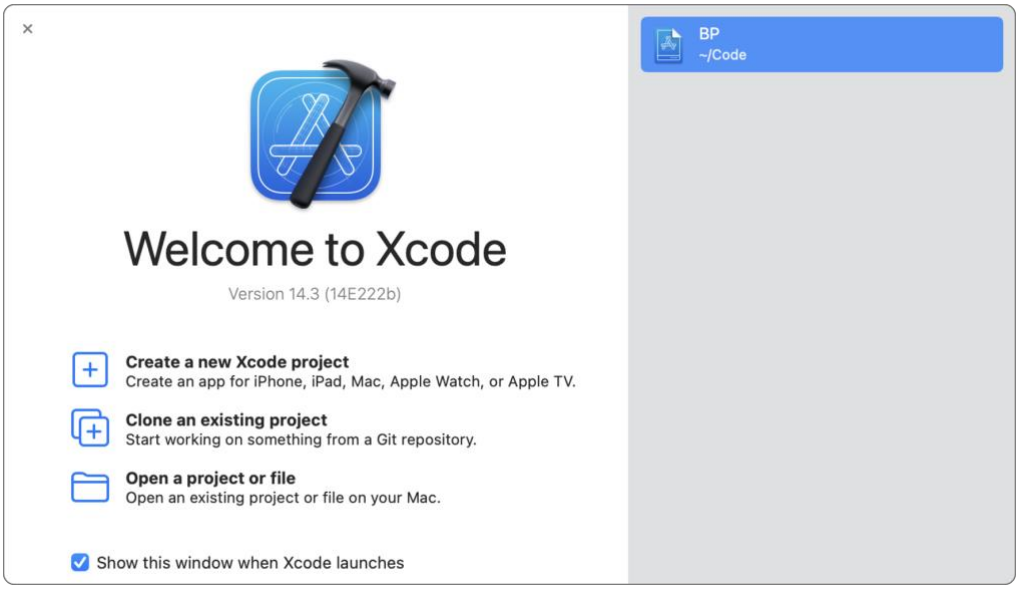
5.2 Implementace aplikace

Podkapitola implementace aplikace začíná inicializací projektu v IDE Xcode a jeho konfigurací. Následně je proveden návrh datového modelu ve frameworku Core Data, který slouží jako základní struktura pro uchovávání a správu dat. Poté je vytvářeno uživatelské rozhraní ve frameworku SwiftUI. V závěru této podkapitoly jsou popsány nejdůležitější části kódu v projektu zajišťující hlavní funkcionalitu aplikace.

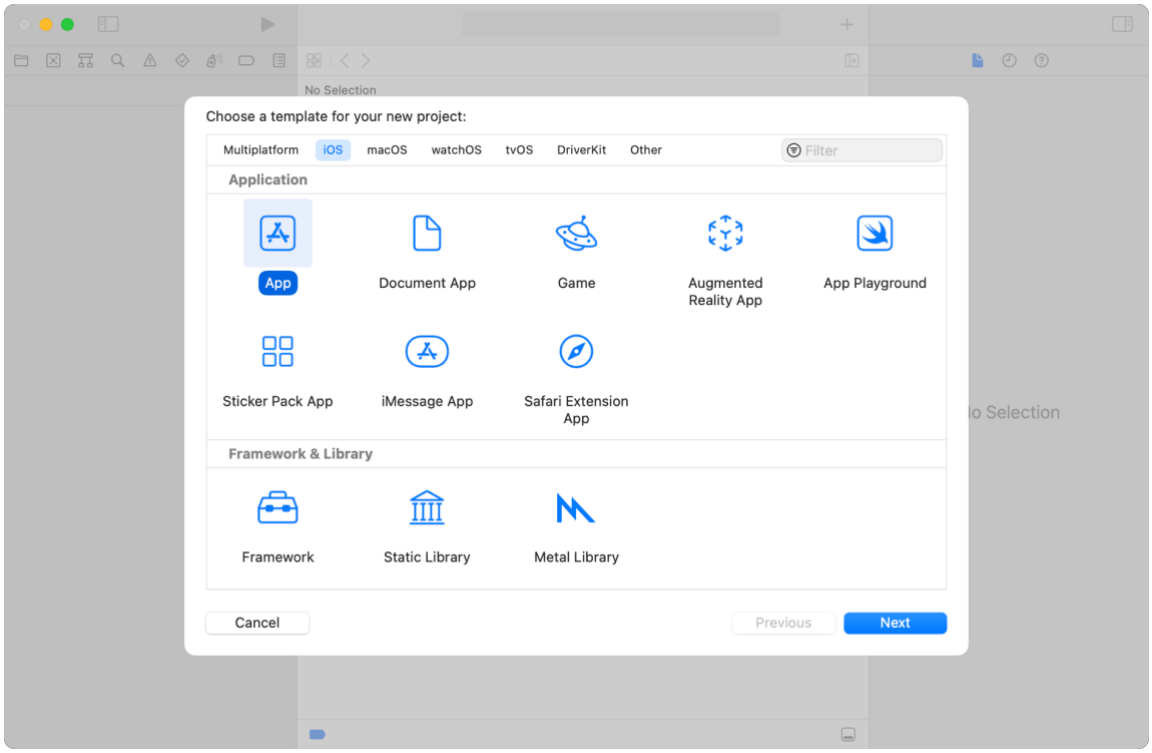
5.2.1 Inicializace projektu

Inicializace projektu iOS aplikace v Xcode zahrnuje vytvoření nového projektu a základní konfiguraci pro vývoj. To je docíleno spuštěním Xcode a vybráním možnosti *Create a new*

Xcode project (viz obrázek číslo 30), Následně je potřeba zvolit šablonu aplikace pro platformu iOS (viz obrázek číslo 31).



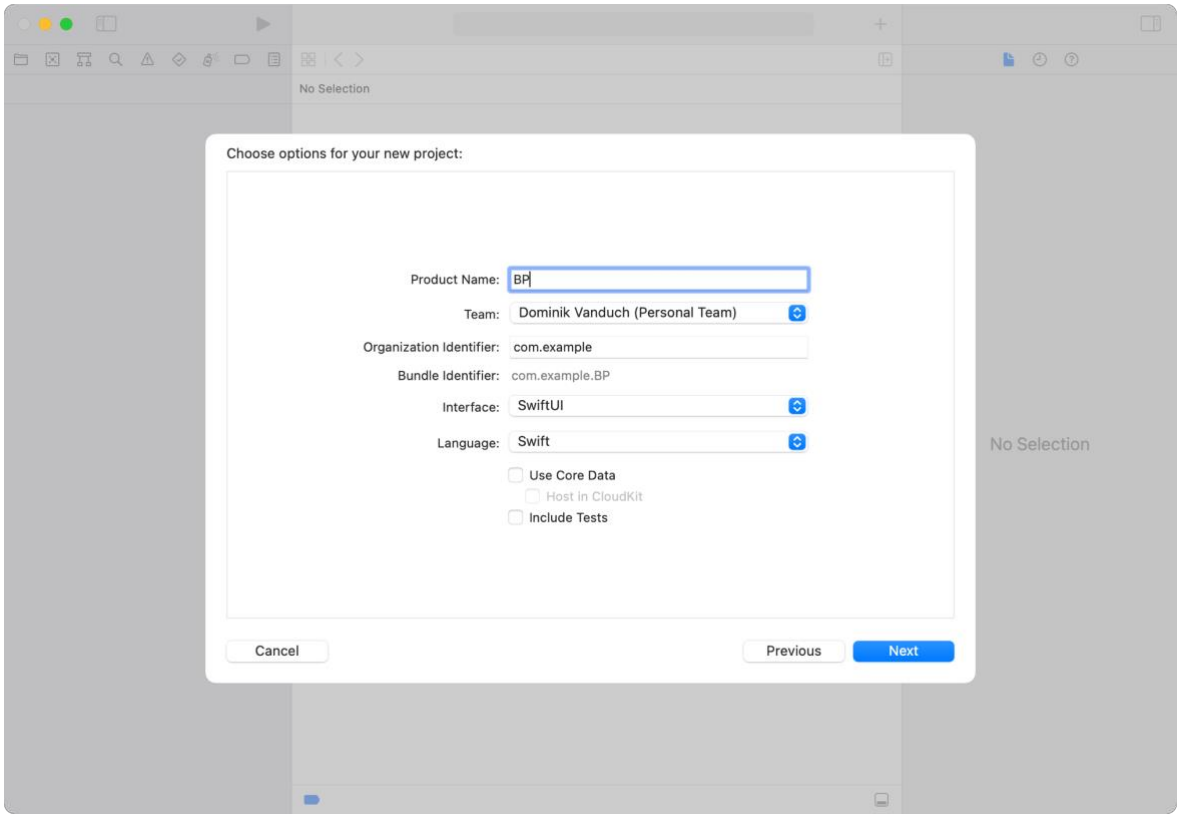
Obrázek 30 Rozhraní Xcode IDE pro vytvoření a otevření projektu



Obrázek 31 Rozhraní Xcode IDE pro zvolení typu projektu při vytváření

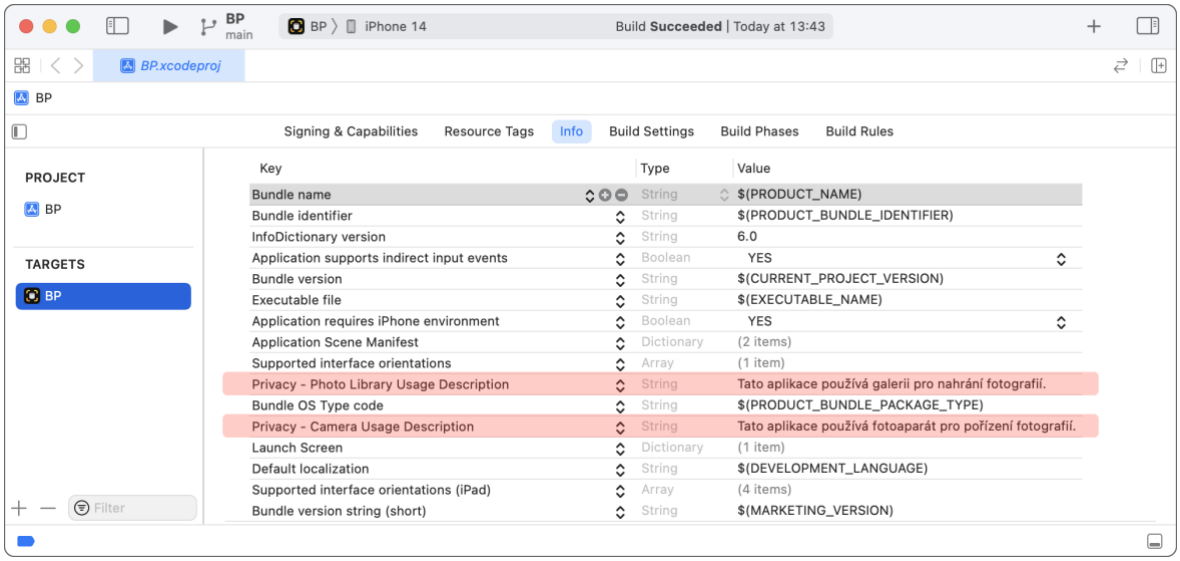
V další části je uživatel vyzván vyplnit potřebné informace o projektu, jako je například jeho jméno, preferovaný framework nebo programovací jazyk. Tuto obrazovku je možno vidět na obrázku číslo 32. Pro prototypovou aplikaci byl zvolen framework SwiftUI

a programovací jazyk Swift. V tento moment je také možné do projektu přidat framework Core Data, ale tato možnost přidá do projektu několik souborů a konfigurací, které by bylo později třeba odstranit a měnit. Z toho důvodu při vytváření prototypové aplikace bylo toto zaškrtávací pole ponecháno prázdné.



Obrázek 32 Rozhraní Xcode IDE pro konfiguraci nového projektu

Po vytvoření projektu je možné provést další konfigurace, jako například nastavení oprávnění pro přístup k funkcím zařízení. Vzhledem k tomu, že prototypová aplikace vyžaduje přístup k fotoaparátu a galerii uživatele, bylo nezbytné zajistit, že tyto funkce jsou transparentní a jasně komunikovány uživatelům. Specificky, dle požadavků operačního systému iOS, bylo potřeba do konfigurace projektu začlenit klíče *Privacy – Camera Usage Description* a *Privacy – Photo Library Usage Description*. Konfigurace těchto klíčů se provádí v souboru s příponou *xcodeproj*, konkrétněji v záložce *Info*. Tyto klíče mají za cíl definovat zprávy, které se zobrazí uživatelům při požadování přístupu k jejich osobním datům – v tomto případě k fotoaparátu a galerii. Proces konfigurace je možno vidět na obrázku číslo 33, kde jsou klíče zvýrazněny červenými obdélníky. Soubor s příponou *xcodeproj* je možné zobrazit vybráním položky projektu v navigátoru (levý panel IDE zobrazující soubory v projektu).



Obrázek 33 Přidání klíčů pro upozornění uživatele na použití fotoaparátu a galerie

Tím je projekt nakonfigurován a připraven pro vývoj iOS aplikace.

5.2.2 Konfigurace grafických zdrojů

Konfigurace grafických zdrojů je nezbytnou součástí vývoje aplikací pro iOS a pro jejich správu slouží soubor *Assets.xcassets*. Tento soubor funguje jako centrální úložiště pro všechny grafické prvky, jako jsou ikony, obrázky a barvy. Díky tomuto souboru je možné efektivně organizovat a spravovat tyto zdroje, které jsou poté integrovány do uživatelského rozhraní aplikace.

Barva aplikace

V rámci konfigurace souboru *Assets.xcassets* byla jako doplňková barva (nazývaná *accentColor*) zvolena oranžová. Výběr doplňkové barvy má značný vliv na celkový vizuální dojem uživatelského rozhraní a slouží k zdůraznění různých prvků, včetně ikon, textu a ovládacích prvků.

Ikona aplikace

Ikona aplikace byla navržena s využitím nástroje Figma. Při návrhu ikony byl kladen důraz na symboliku a význam, který by měl být pro uživatele snadno pochopitelný a zapamatovatelný. Zhotovenou ikonu je možno vidět na obrázku číslo 34.

Centrálním prvkem ikony je symbol úložiště, který je zobrazen v šedých odstínech na černém pozadí. Symbol úložiště byl zvolen, aby vyvolával asociace s ukládáním a organizací dat, což je jednou z hlavních funkcí prototypové aplikace.

Symbol úložiště je navíc obklopen rámečkem, který představuje „zaostřovací box", charakteristický pro vizuální reprezentaci skenování nebo zaostřování v oblasti fotografie. Tento rámeček vytváří dojem skenování nebo detekce, což je druhým důležitým prvkem prototypové aplikace. Rámeček je vyhotoven v barevném přechodu mezi žlutou a oranžovou barvou, což volně odráží barvu rozhraní.

Černé pozadí ikony bylo zvoleno, aby vytvořilo kontrast, který by umožnil symbolu úložiště a rámečku vyniknout. Toto rozhodnutí bylo učiněno s cílem zvýšit vizuální atraktivitu ikony a usnadnit její rozpoznání uživatelům.

Obrázek 34 Ikona aplikace

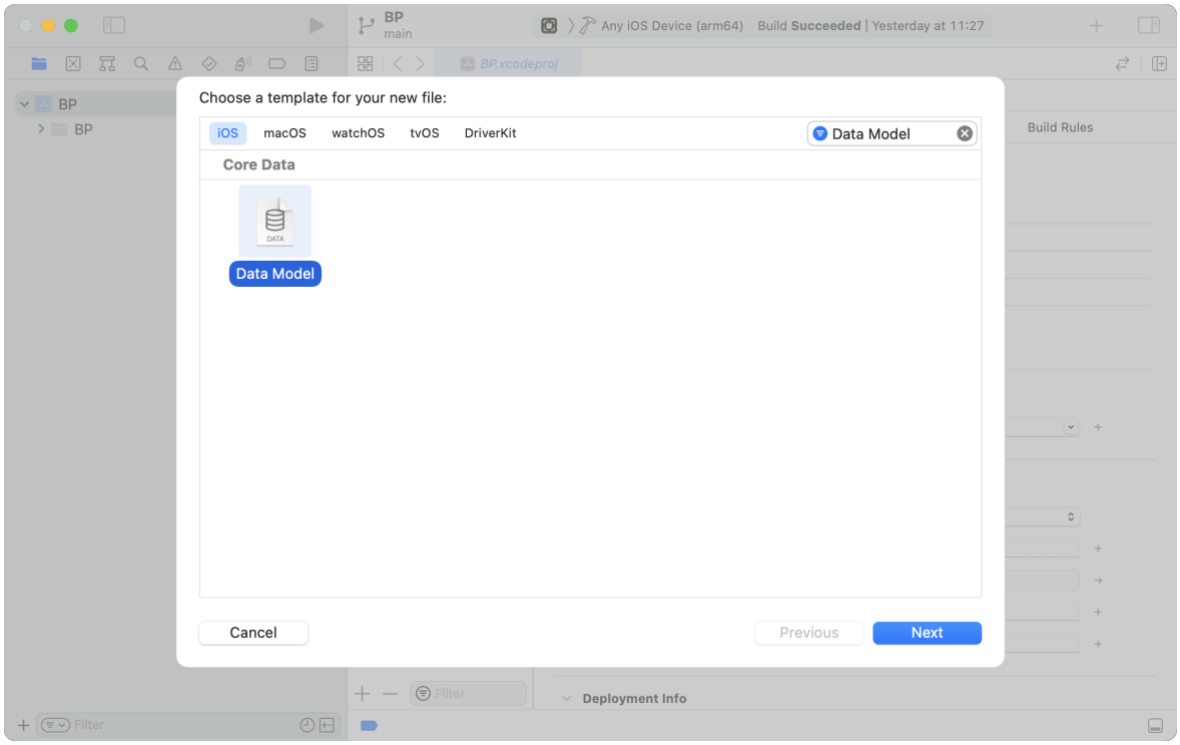
5.2.3 Vytvoření datového modelu

Při vytváření projektu nebyla záměrně zvolena možnost pro automatické přednastavení Core Data. Toto rozhodnutí bylo učiněno z toho důvodu, že po zvolení této možnosti je do projektu přidáno určité množství kódu, který je potřeba změnit nebo úplně odstranit. Z toho důvodu bylo pro úsporu času a kompletní přizpůsobení požadavkům prototypové aplikace zvoleno manuální přidání frameworku Core Data, namísto automatického přednastavení. Do nově vytvořeného Core Data modelu poté byly vytvořeny jednotlivé entity a nastaveny jejich atributy a vztahy.

Pro přidání Core Data do projektu je potřeba provést tři úkony:

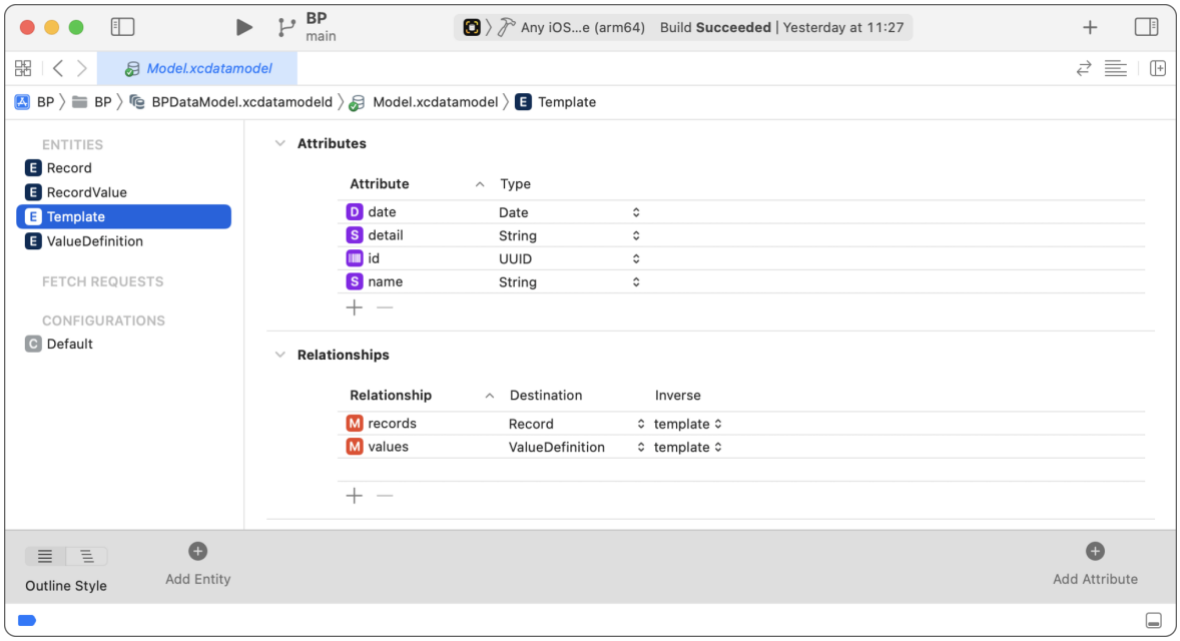
- vytvořit soubor pro datový model,
- vytvořit *NSPersistentContainer*,
- použít Core Data kontejner v aplikaci.

Přidání datového modelu se provádí vytvořením souboru v projektu, který se nazývá *Data Model* (viz obrázek číslo 35).



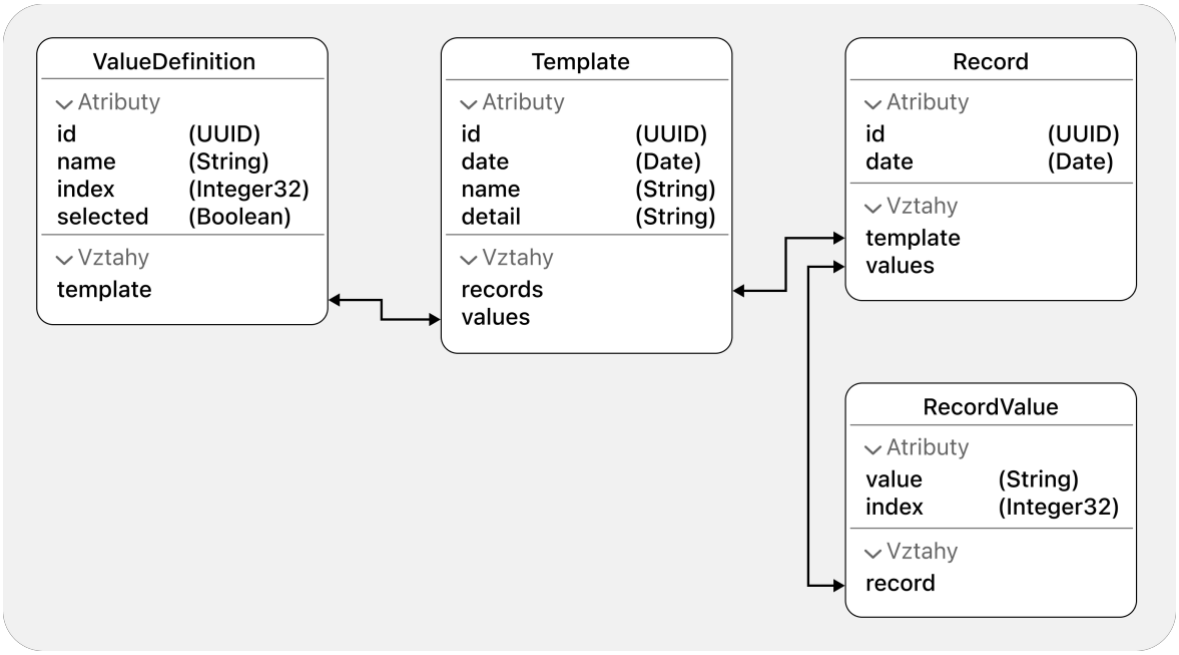
Obrázek 35 Vytvoření souboru datového modelu

Tento soubor má příponu *xcdatamodeld* a Xcode při jeho otevření zobrazuje speciální uživatelské rozhraní pro definování a práci s entitami. Toto uživatelské rozhraní se nazývá Core Data model editor a je zobrazeno na obrázku číslo 36. Core Data model editor umožňuje definovat a pracovat s entitami, atributy a vztahy pomocí grafického rozhraní.



Obrázek 36 Uživatelské rozhraní Core Data Model editoru

Na základě předchozí analýzy požadavků a drátěných modelů aplikace jsou definovány entity, atributy a jejich vzájemné vztahy. První entitou vyplývající z požadavků je entita šablony, pojmenována *Template*. Šablona uchovává informace o tom, jaké hodnoty v ní uživatel digitalizuje a také již digitalizované hodnoty v podobě záznamů. K definování digitalizovaných hodnot slouží entita *ValueDefinition*, která slouží k uchování jednotlivých hodnot pro digitalizaci dat v dané šabloně. Další entitou je *Record*, která uchovává záznamy. Ke každému záznamu se pojí konkrétní odečtené hodnoty. Ty jsou ukládány prostřednictvím poslední entity zvané *RecordValue*. Tyto entity je možno graficky vidět na obrázku číslo 37.



Obrázek 37 Diagram Core Data modelu aplikace

Další částí implementace Core Data do projektu je vytvoření *NSPersistentContainer*. *NSPersistentContainer* je třída ve frameworku Core Data, která spravuje úložiště a komunikaci s datovým modelem aplikace. Poskytuje jednoduché rozhraní pro manipulaci s daty, včetně jejich načítání, ukládání a vyhledávání.

Inicializace kontejneru v podobě třídy *NSPersistentContainer* je zobrazena na obrázku číslo 38. Jak je na základě číslování řádků možno pozorovat, tento soubor obsahuje více kódu, než je zobrazeno. Součástí souboru je také inicializace kontejneru s vygenerovanými daty za účelem poskytnutí dat do náhledů obrazovek při vývoji v Xcode. Tato data jsou umístěna do databáze v paměti (anglicky nazývané *in-memory database*) a slouží jen pro zobrazování uměle vytvořených dat do náhledů.

```
68      let container: NSPersistentContainer
69
70      init(inMemory: Bool = false) {
71          container = NSPersistentContainer(name: "BPDataModel")
72          if inMemory {
73              container.persistentStoreDescriptions.first!.url = URL(fileURLWithPath: "/dev/null")
74          }
75          container.loadPersistentStores(completionHandler: { (storeDescription, error) in
76              if let error = error as NSError? {
77                  fatalError("Unresolved error \(error), \(error.userInfo)")
78              }
79          })
80          container.viewContext.automaticallyMergesChangesFromParent = true
81      }
```

Obrázek 38 Inicializace kontejneru NSPersistentContainer

Poslední částí implementace frameworku Core Data je využití vytvořeného kontejneru v rámci aplikace. V ukázce kódu na obrázku číslo 39 je vidět příklad, kde je instance kontejneru uložena do proměnné *persistenceController*. Následně je pomocí metody *environment* předán kontext pro správu objektů do aplikace. Tato metoda slouží k nastavení globálního prostředí pro aplikaci, přičemž hodnota *managedObjectContext* definuje kontext pro správu objektů. Tímto způsobem je zajištěno, že aplikace má přístup k spravovanému kontextu a umožňuje práci s daty v rámci frameworku Core Data.

```
1  import SwiftUI
2
3  @main
4  struct BPApp: App {
5      let persistenceController = PersistenceController.shared
6
7      var body: some Scene {
8          WindowGroup {
9              ContentView()
10                 .environment(\.managedObjectContext, persistenceController.container.viewContext)
11         }
12     }
13 }
```

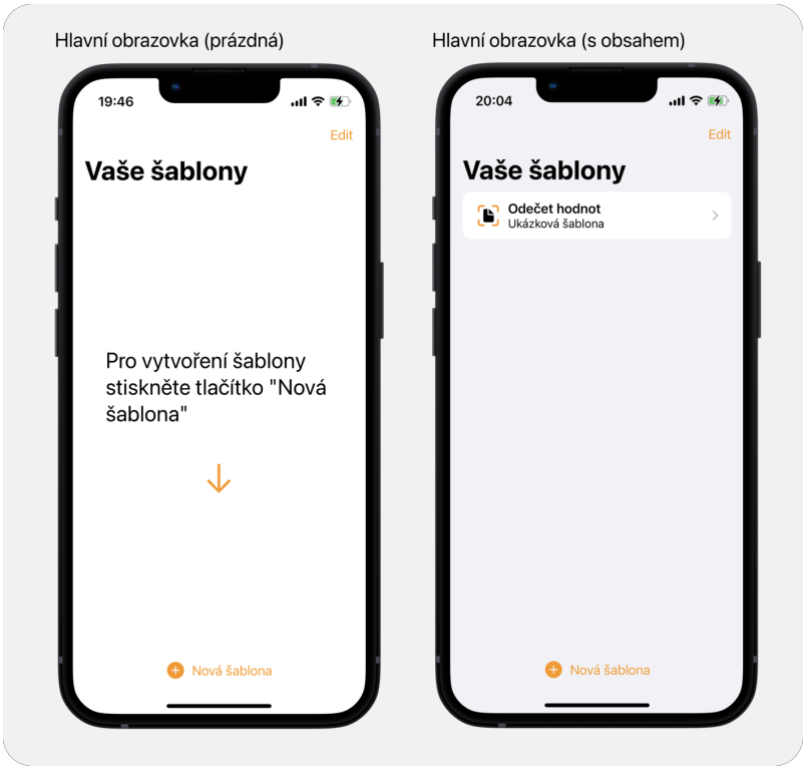
Obrázek 39 Použití Core Data kontejneru v aplikaci

5.2.4 Vytvoření uživatelského rozhraní

Tato podkapitola se zabývá implementací uživatelského rozhraní aplikace na základě předchozího drátěného modelu a stanovených požadavků. Hlavním cílem této kapitoly je představit uživatelské rozhraní vytvořené pomocí frameworku SwiftUI, který byl popsán a podrobněji vysvětlen v teoretické části této práce. V této podkapitole budou prezentovány klíčové aspekty implementovaného uživatelského rozhraní, včetně struktury, rozložení, stylování a interakce mezi jednotlivými *views*. Při vývoji byl kladen důraz na efektivitu a dodržování principů SwiftUI, které podporují modularitu a znovupoužitelnost jednotlivých

částí rozhraní. Tímto přístupem je zajištěna jednoduchá správa a rozšiřitelnost aplikace pro vývojáře, zatímco uživatelé mohou snadno ovládat aplikaci díky intuitivnímu využití osvědčených prvků uživatelského rozhraní iOS.

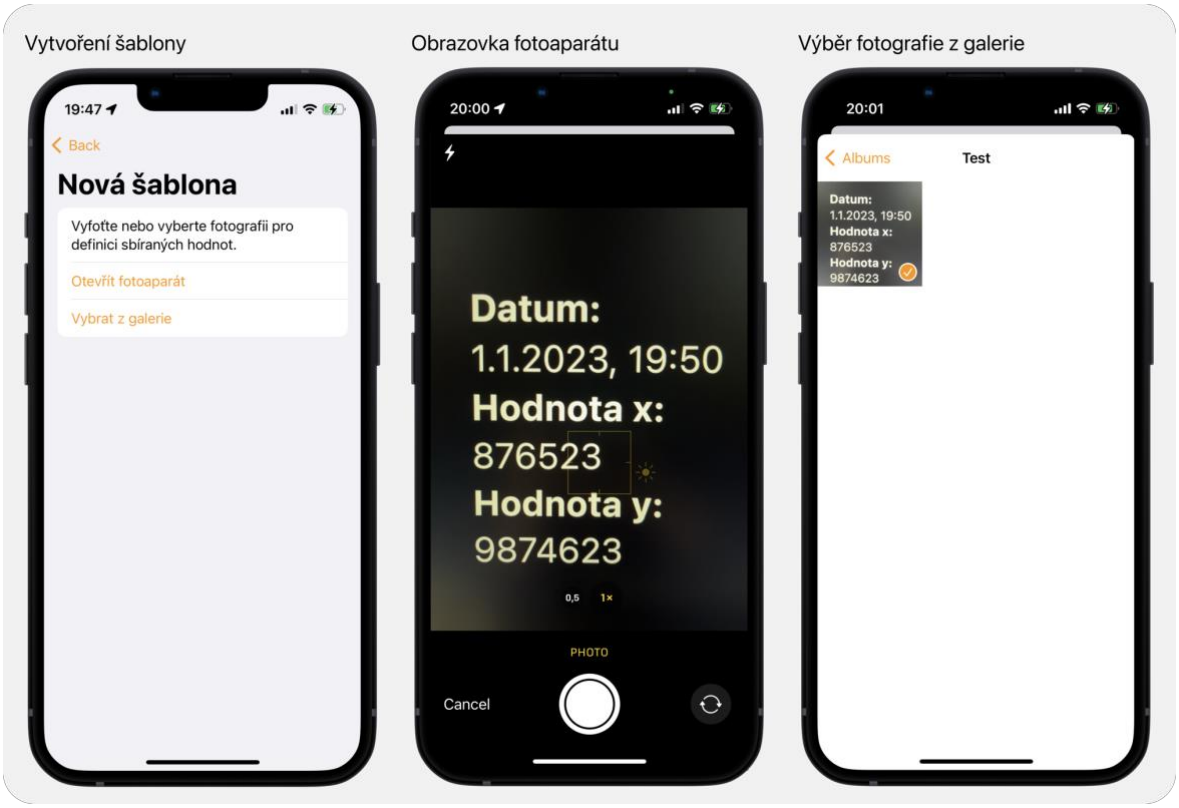
Hlavní obrazovka aplikace, která je základním prvkem uživatelského rozhraní, při prvním otevření prezentuje uživatelům dostupné funkce a navádí je k prvním krokům. Na obrázku číslo 40 je v levé části zobrazena hlavní obrazovka ihned po nainstalování aplikace. Pro usnadnění orientace uživatele v aplikaci jsou na této obrazovce zobrazeny instrukce, které vyzývají uživatele k vytvoření nové šablony stisknutím tlačítka *Nová šablona*. Instruktivní text je doplněn šipkou, která ukazuje směrem k tlačítku *Nová šablona* v dolní části aplikace. V pravé části obrázku číslo 40 je zobrazeno, jak hlavní obrazovka vypadá poté, co uživatel vytvoří první šablonu. Ihned po vytvoření šablony se zobrazované instrukce automaticky skryjí a místo nich jsou uživateli prezentovány vytvořené šablony, které jsou zobrazeny pod sebou. Tímto způsobem reaguje aplikace na aktuální stav a poskytuje uživatelům intuitivní a plynulý proces používání aplikace.



Obrázek 40 Hlavní obrazovka aplikace

Proces vytváření šablon v aplikaci je zobrazen v levé části obrázku číslo 41, kde je zachyceno, jak aplikace uživatele vyzývá k výběru zdroje fotografie pro vytvoření šablony. Tato interaktivní obrazovka umožňuje uživatelům vybrat, zda chtějí použít fotoaparát

zařízení nebo vybrat fotografii z galerie. Pokud uživatel vybere volbu fotoaparátu, aplikace zobrazí prostřední obrazovku, která představuje zabudovanou obrazovku fotoaparátu z iOS. Tato integrace umožňuje uživatelům okamžitě pořídit fotografii a použít ji jako základ pro vytvoření šablony. Na poslední obrazovce v pravé části obrázku číslo 41 je zobrazena možnost výběru fotografie z galerie, který je realizován také pomocí zabudovaného rozhraní. Tímto způsobem aplikace umožňuje uživatelům snadno nahrát již existující fotografie a použít je pro vytvoření šablon.

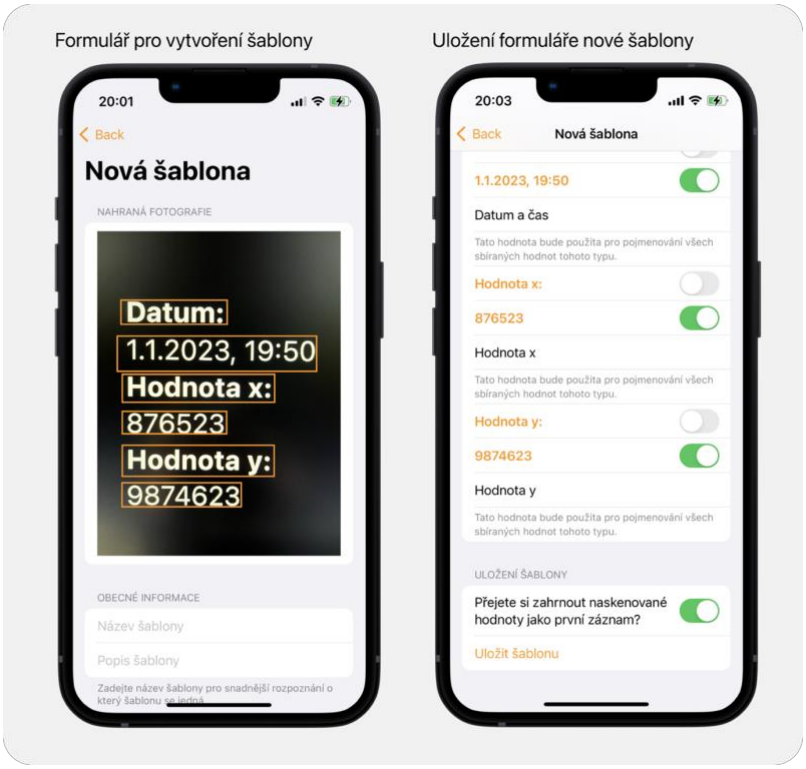


Obrázek 41 Volba zdroje fotografie pro vytvoření šablony

Na obrázku číslo 42 jsou zobrazeny dva snímky obrazovky, které zachycují formulář pro tvorbu šablony. V levé části obrázku je první polovina formuláře, která zobrazuje nahranou fotografii doplněnou o vykreslení obdélníků kolem detekovaného textu. Tímto uživatel může vizuálně ověřit správnost nahrání fotografie a současně sledovat, zda proběhla detekce textu správně. Pod touto fotografií má uživatel možnost vyplnit obecné informace, jako jsou název a popis šablony. V pravé části obrázku je zobrazena druhá polovina formuláře, kde má uživatel možnost si vybrat, které z detekovaných hodnot chce digitalizovat. Všechny detekované hodnoty jsou zobrazeny pod sebou a vedle každé z hodnot je přepínač, který určuje, zda je hodnota označena jako sbíraná nebo zda bude při digitalizaci ignorována. Pokud uživatel vybere některou z hodnot, zobrazí se mu vstupní pole, ve kterém musí uvést

název, pod kterým bude tato hodnota ukládána. Tímto způsobem se zajišťuje jednoznačnost a rozlišení mezi jednotlivými hodnotami při používání aplikace.

Jelikož se i v nahrané fotografii vyskytují data, aplikace se zeptá uživatele, zda si přeje uložit tato data jako první záznam hodnot pro tuto šablonu. Uživatel musí označit přepínač vedle této otázky, pokud chce tuto možnost využít. Po vyplnění všech požadovaných hodnot může uživatel uložit šablonu a poté je přesměrován na hlavní obrazovku aplikace.



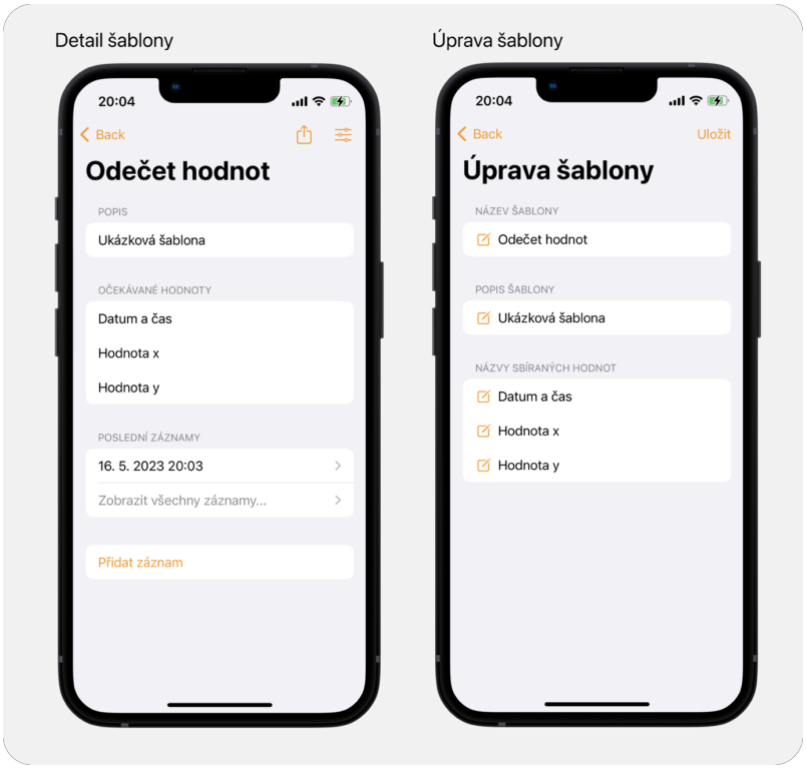
Obrázek 42 Formulář pro vytvoření šablony

Obrázek číslo 43 představuje dva snímky obrazovky, které ilustrují dvě důležité obrazovky aplikace: Detail šablony a Úprava šablony.

V levé části obrázku číslo 43 je zobrazena obrazovka s detailem šablony. Tato obrazovka je přístupná po kliknutí na konkrétní šablonu na hlavní obrazovce aplikace a poskytuje uživateli podrobný přehled o dané šabloně. Obsahuje informace o šabloně, jako je její popis a očekávané hodnoty pro digitalizaci. Dále zde uživatel nalezne posledních pět záznamů, které byly pro danou šablonu vytvořeny. Pro zobrazení všech záznamů je k dispozici tlačítko umožňující zobrazení kompletní historie záznamů. V horní části obrazovky, v sekci nazývané *toolbar* (česky lišta nástrojů), je umístěna možnost exportu dat pro danou šablonu. Tato možnost je reprezentována SF symbolem používaným pro vyjádření činnosti sdílení (čtverec se šipkou směrem nahoru), kterou uživatelé iOS znají a intuitivně ji vnímají jako

možnost sdílení nebo exportu dat. Vedle symbolu sdílení je symbol konfigurace, který uživatelům umožňuje přejít na obrazovku pro úpravu šablony. Poslední neméně významnou možností, kterou tato obrazovka nabízí, je přidání nových záznamů pro danou šablonu.

V pravé části obrázku číslo 43 je zobrazena již dříve zmíněná obrazovka pro úpravu šablony. Tato obrazovka umožňuje uživatelům provést změny v hodnotách šablony, jako je její název, popis nebo názvy jednotlivých sbíraných hodnot. Vedle vstupních polí je symbol tužky a papíru naznačující možnost tento text upravovat.



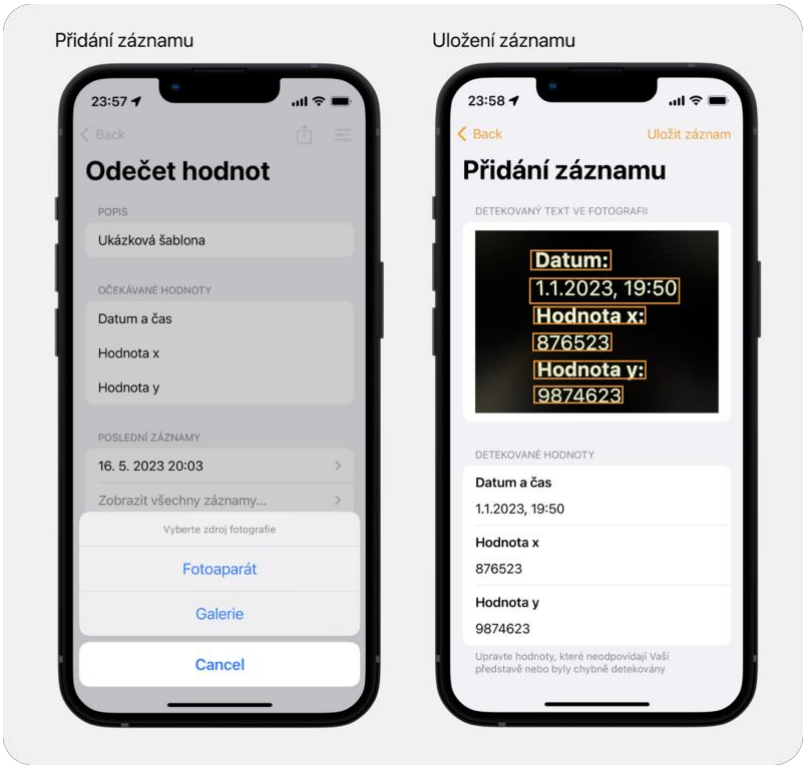
Obrázek 43 Detail a úprava šablony v aplikaci

Z obrazovky detailu šablony, jak bylo demonstrováno v předchozích částech, může uživatel přidat nový záznam. Po stisku tlačítka pro přidání záznamu je uživateli prezentováno rozhraní pro výběr zdroje fotografie. Uživatel zde má možnost zvolit buď fotoaparát zařízení, galerii, nebo zrušit akci. Tento moment je zobrazen v levé části obrázku číslo 44.

Pokud uživatel zvolí možnost fotoaparátu nebo galerie, aplikace pokračuje k digitalizaci dat ze zvoleného obrazového zdroje. Výsledek této digitalizace je zobrazen na pravé polovině obrázku číslo 44. Zde je uživateli prezentována nahraná fotografie s obdélníky kolem detekovaného textu, přičemž se jedná o stejný vizuální postup, jaký byl použit při vytváření šablony. Pod tímto náhledem je sekce „detekované hodnoty“, která zobrazuje hodnoty identifikované během digitalizace.

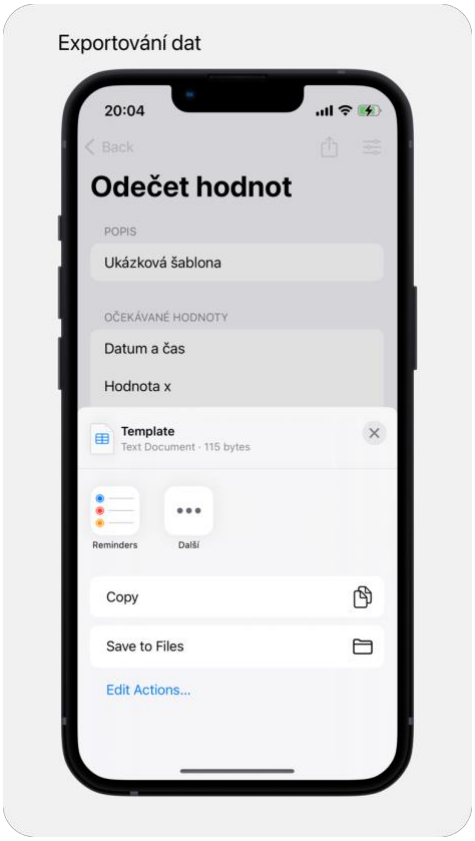
Důležité je podotknout, že aplikace je navržena tak, aby správně identifikovala, která data ve fotografii mají být přiřazeny ke kterým sbíraným hodnotám. Tento proces je závislý na předchozím nastavení šablony.

Uživatel má možnost tyto detekované hodnoty upravit, pokud detekce textu neproběhla správně, což může být způsobeno faktory jako jsou odlesky, stíny nebo prach na fotografii. Pokud je uživatel s detekovanými hodnotami spokojen, může záznam uložit a pokračovat v další interakci s aplikací.



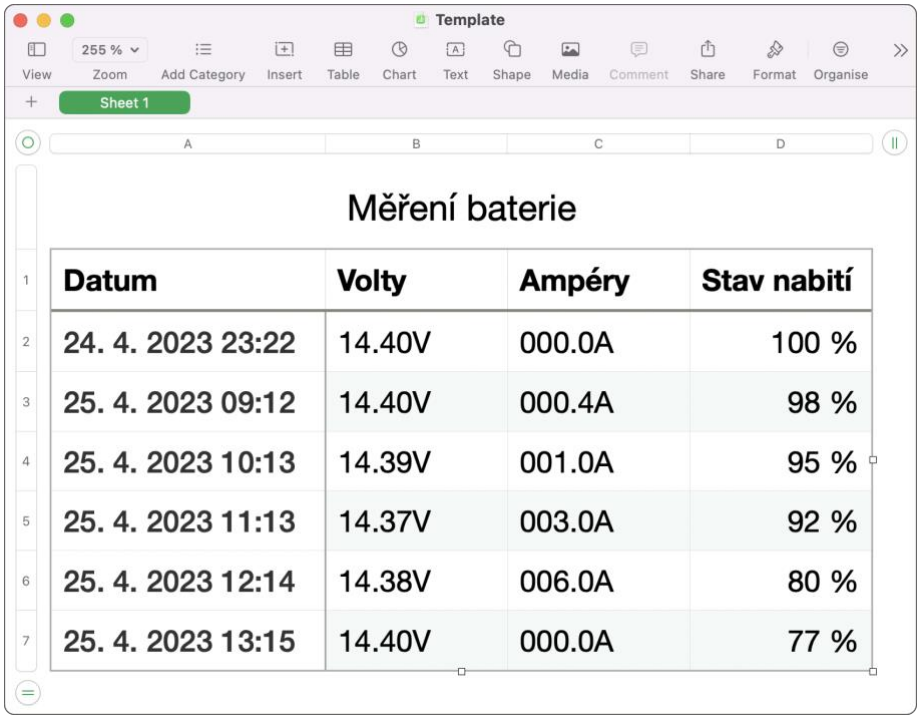
Obrázek 44 Proces přidávání nového záznamu

Z obrazovky detailu šablony má uživatel možnost také exportovat data pomocí tlačítka označeného symbolem sdílení/exportování. Po stisku tohoto tlačítka se zobrazí zabudovaná obrazovka sdílení, která je standardní součástí prostředí systému iOS. Tato obrazovka poskytuje uživateli řadu možností pro manipulaci dat, včetně jejich uložení do souborů, sdílení s jinými aplikacemi nebo lidmi. Tento proces je zobrazen na obrázku číslo 45.



Obrázek 45 Exportování dat z aplikace

V návaznosti na možnost exportu dat, jak je uvedeno výše, je v obrázku číslo 46 prezentováno, jak exportovaná data vypadají po otevření v externí aplikaci. Pro ilustrativní účely byla zvolena aplikace Numbers na operačním systému macOS. Tato aplikace představuje jeden z mnoha možných způsobů, jakými mohou uživatelé interagovat s exportovanými daty.



Obrázek 46 Exportovaná data z aplikace otevřená v programu Numbers

Kapitola „Vytvoření uživatelského rozhraní“ podrobně ilustruje proces převodu teoretických konceptů a nákresů do praktického vývoje aplikace. Díky nákresům, drátěnému modelu a modelu dat bylo dosaženo hlubokého chápání struktury a funkcí aplikace, což usnadnilo implementaci uživatelského rozhraní. Modularita a znovupoužitelnost prvků uživatelského rozhraní byly plně využity k vytvoření flexibilního a intuitivního rozhraní, které je snadno rozšiřitelné pro budoucí potřeby. Implementace běžně se vyskytujících částí uživatelského rozhraní v systému iOS byla provedena s cílem umožnit uživatelům intuitivní interakce s aplikací.

5.2.5 Důležité části kódu

Tato podkapitola se soustředí na klíčové části kódu, které tvoří jádro funkcionality prototypové aplikace. Po detailním prozkoumání designu uživatelského rozhraní a datového modelu následuje detailní pohled na kód, který tuto aplikaci implementuje a poskytuje uživatelům rychlý a intuitivní proces digitalizace textu.

Podkapitola se zaměřuje na čtyři zásadní části kódu:

- digitalizaci textu, která využívá Vision framework spolu s vlastní logikou,
- vykreslování ohraničujících obdélníků kolem detekovaného textu ve fotografiích,

- spouštění vytvořených funkcí v rámci aplikace,
- exportování dat do formátu CSV.

Tyto funkcionality byly navrženy a implementovány na základě požadavků a cílů aplikace, které byly zformulovány v návrhové fázi a vizualizovány v nákresech a drátěných modelech.

Kód aplikace je napsán v programovacím jazyce Swift, který v kombinaci s frameworkem SwiftUI umožňuje efektivně implementovat cíle a požadavky stanovené v průběhu vývoje. Každá část kódu byla navržena s ohledem na modularitu a snadnou údržbu, přičemž stále plní efektivně cíle aplikace. Byl kladen důraz na to, aby aplikace poskytovala pro uživatele jednoduché a intuitivní prostředí pro digitalizaci textu, přičemž složitou interní logiku udržovala skrytou.

Digitalizace textu

Jeden z hlavních kroků v procesu digitalizace provádí funkce *getObservations*. Tato funkce přijímá jako vstupní parametr obrázek typu *UIImage?* a vrací pole postřehů, které jsou typu *VNRecognizedTextObservation*.

V kontextu Vision frameworku jsou postřehy výsledkem procesu rozpoznávání a analýzy obrazu. V případě rozpoznávání textu jsou tyto postřehy abstraktní reprezentací textu rozpoznaného na obrázku.

Funkce *getObservations* provádí klíčovou úlohu v procesu digitalizace textu, jelikož slouží jako propojení mezi obrázkem a jeho analýzou pomocí frameworku Vision. Na začátku této funkce je deklarována proměnná *observations*, která slouží pro uchování detekovaných postřehů. Dále je provedena kontrola, jestli byl do funkce vložen validní obrázek. V případě že nebyl, program vygeneruje chybovou hlášku. Poté je implementován algoritmus pro detekci textu, který se skládá z následujících částí:

- vytvoření instance *VNImageRequestHandler*, která slouží pro zpracování obrazu,
- vytvoření instance *VNRecognizeTextRequest*, což je požadavek pro detekci textu z obrazu,
- spuštění požadavku pro detekci textu.

Po spuštění požadavku jsou výsledky detekce uloženy do proměnné *observations* a vráceny jako výstup funkce. Kompletní implementaci funkce *getObservations* lze vidět na obrázku číslo 47.

```swift
func getObservations(image: UIImage?) -> [VNRecognizedTextObservation] {
    var observations = [VNRecognizedTextObservation]()

    guard let cgImage = image?.cgImage else {
        fatalError("could not get cgImage")
    }

    // Handler
    let handler = VNImageRequestHandler(cgImage: cgImage, options: [:])

    // Request
    let request = VNRecognizeTextRequest { request, error in
        guard let results = request.results as? [VNRecognizedTextObservation],
              error == nil else {
            return
        }

        observations = results
    }

    // Process request
    do {
        try handler.perform([request])
    }
    catch {
        print(error)
    }

    return observations
}
```

Obrázek 47 Kompletní implementace funkce getObservations

Funkce *getTextFromObservation* představuje další část procesu digitalizace textu. Jejím úkolem je zpracování vstupního pole obsahujícího postřehy detekovaného textu a vrácení výsledného pole textových řetězců s rozpoznaným textem.

Na počátku funkce je inicializováno prázdné pole *result*, které slouží pro uchování výsledného rozpoznaného textu. Poté funkce prochází všechny postřehy vstupního pole. Pro každý postřeh je získán nejlepší kandidát (nejpravděpodobnější rozpoznaný text) metodou *topCandidates*. Tato metoda vrací pole s nejlepšími kandidáty na základě pravděpodobnosti správného rozpoznání. Funkce pracuje pouze s nejlepším kandidátem, kterého získává pomocí metody *first*. Pokud není k dispozici žádný kandidát, cyklus pokračuje dalším postřehem. Pokud je kandidát dostupný, jeho text je přidán do výsledného pole. Poté, co jsou zpracovány všechny postřehy, je výsledné pole vráceno jako výstup funkce.

Kompletní implementaci funkce *getTextFromObservation* je možno vidět na obrázku číslo 48.

```swift
func getTextFromObservation(from observations: [VNRecognizedTextObservation]) -> [String] {
    var result = [String]()

    for observation in observations {
        guard let topCandidate = observation.topCandidates(1).first else { continue }
        result.append(topCandidate.string)
    }

    return result
}
```

Obrázek 48 Implementace funkce getTextFromObservation

Vykreslování ohraničujících obdélníků

V rámci procesu digitalizace textu aplikace zahrnuje funkci, která vykresluje ohraničující obdélníky kolem detekovaného textu na nahraných fotografiích. Tato funkce, ač není nezbytně nutná pro samotnou digitalizaci textu, značně zlepšuje komfort a kvalitu uživatelské interakce při používání aplikace. Poskytuje totiž uživatelům okamžitou vizuální zpětnou vazbu, díky které mohou snadno pochopit a ověřit, zda proběhla detekce textu úspěšně.

Následující úryvek kódu na obrázku číslo 49 ukazuje, jak je tento proces implementován v rámci formuláře pro vytváření šablony. Zobrazení nahrané fotografie je zajištěno prvkem *Image* z frameworku SwiftUI. Tento prvek je nastaven tak, aby byl schopen měnit velikost a zachovat při tom původní poměr stran obrázku. Dále je na něj aplikována funkce *overlay*, která umožňuje přidání grafických prvků přímo na obrázek.

Pomocí funkce *GeometryReader* je získána aktuální velikost obrázku, což umožňuje přesné vykreslení ohraničujících obdélníků. Pro každou detekovanou oblast textu, která je uložená v proměnné *observations*, je pak vykreslen ohraničující obdélník.

Je důležité poznamenat, že souřadnice detekovaných oblastí textu, které jsou získány z frameworku Vision, musí být převedeny na souřadnicový systém používaný ve SwiftUI. Zatímco Vision definuje počátek souřadnicového systému v levém dolním rohu, SwiftUI (i UIKit) mají počátek definován v levém horním rohu.

```
 1 import CoreData
 2 import SwiftUI
 3 import Vision
 4
 5 struct TemplateFormView: View {
32     // ...
33
34     var body: some View {
35         List {
36             Section {
37                 Image(uiImage: image)
38                     .resizable()
39                     .aspectRatio(contentMode: .fit)
40                     .overlay(
41                         GeometryReader { geometry in
42                             ForEach(observations, id: \.uuid) { observation in
43                                 let boundingBox = observation.boundingBox
44                                 let x = boundingBox.origin.x * geometry.size.width
45                                 let y =
46                                     (1 - observation.boundingBox.origin.y - observation.boundingBox.height)
47                                         * geometry.size.height
48                                 let width = boundingBox.width * geometry.size.width
49                                 let height = boundingBox.height * geometry.size.height
50
51                                 Rectangle()
52                                     .stroke(Color.accentColor, lineWidth: 2)
53                                     .frame(width: width, height: height)
54                                     .position(x: x + width / 2, y: y + height / 2)
55                             }
56                         }
57                     )
58             } header: {
59                 Text("Nahraná fotografie")
60             }
61
62             // ...
116         }
117     }
166     // ...
167 }
```

Obrázek 49 Vykreslování ohraničujících obdélníku ve formuláři nové šablony

Použití funkcí digitalizace textu v aplikaci

V této podkapitole je představen způsob začlenění procesu digitalizace do aplikace. Konkrétně je zobrazena obrazovka pro vytváření nové šablony, která je navržena tak, aby dynamicky reagovala na nahrání nové fotografie.

V přiloženém úryvku kódu na obrázku číslo 50 je implementována obrazovka pro volbu zdroje fotografie při vytváření nové šablony, pojmenována jako *NewTemplateView*. Tato struktura obsahuje deklaraci několika proměnných, které udržují její stav. Mezi tyto proměnné patří informace o aktuálně vybraném obrázku, potenciálních chybových hlášeních a také o výsledcích procesu detekce a rozpoznání textu.

Při výběru nového obrázku je proces digitalizace spuštěn automaticky díky modifikátoru *onChange*, který je aplikován na proměnnou *image*. Tento modifikátor zajistí, že jakmile je vybrán nový obrázek, dojde k vyhodnocení a uložení detekovaných postřehů a rozpoznaného textu. Pokud se nepodaří detekovat žádné postřehy, uživatel je o tom informován prostřednictvím chybové hlášky.

```swift
import SwiftUI
import Vision

struct NewTemplateView: View {
    @State private var image: UIImage?
    @State private var showCameraSheet = false
    @State private var showPhotosSheet = false
    @State private var errorMessage = ""

    @State private var observations: [VNRecognizedTextObservation] = []
    @State private var recognizedText: [String] = []

    var body: some View {
        NavigationStack {
            ZStack {
                // ...
            }
            .navigationTitle("Nová šablona")
            .onChange(of: image) { newImage in
                observations = getObservations(image: newImage)
                if observations.isEmpty {
                    errorMessage = "Nepodařilo se detekovat text. Nahrajte jinou fotografii."
                } else {
                    recognizedText = getTextFromObservation(from: observations)
                }
            }
        }
    }
}
```

Obrázek 50 Automatické spouštění digitalizace v aplikaci

Tento přístup k implementaci digitalizace textu umožňuje poskytnutí co nejpříjemnější a nejefektivnější interakce mezi uživatelem a aplikací. Uživatelské rozhraní je navrženo tak, aby minimalizovalo potřebu aktivního zapojení uživatele a maximalizovalo automatizaci. Jakmile je vybrán nový obrázek, proces detekce a rozpoznání textu se spustí automaticky. Nemusí dojít k žádné další interakci ze strany uživatele. Pokud v obrázku nelze detekovat žádný text, aplikace o tomto výsledku digitalizace informuje uživatele a navrhne nahrání jiné fotografie. Tímto způsobem je předcházeno potenciálním nedorozuměním nebo nejasnostem, které by mohly vyplývat z neúspěšného pokusu o detekci textu.

Exportování dat do formátu CSV

V této části je popsán proces generování a sdílení souboru ve formátu CSV v aplikaci. CSV, celým názvem *comma-separated values*, je formát, který byl zvolen pro svou jednoduchost a univerzálnost. Jde o textový soubor, ve kterém jsou jednotlivé hodnoty odděleny čárkami. Tento formát je široce podporován většinou tabulkových procesorů a databázových systémů, což z něj činí ideální volbu pro export dat z aplikace.

V prototypové aplikaci je generování souboru CSV aktivováno po stisknutí symbolu sdílení na obrazovce detailu šablony. Funkcionalita generování CSV souboru je implementována v rozšíření třídy *Template*, ve funkci *generateCSV*, kterou je možno vidět na obrázku číslo 51.

Funkce *generateCSV* začíná inicializací prázdného řetězce *csvString*, který postupně naplní daty pro export. Prvním krokem je přidání názvu prvního sloupce, který je nazýván *Datum*. Následuje získání definic vybraných hodnot (neboli hodnot, které uživatel digitalizuje) z dané šablony a jejich seřazení podle indexu. Názvy těchto definovaných hodnot jsou následně přidány do řetězce *csvString* jako názvy dalších sloupců v CSV souboru. Tímto způsobem je vytvořen první řádek CSV souboru, který obsahuje názvy sloupců.

```
1  import CoreData
2  import Foundation
3
4  extension Template {
5      func generateCSV() -> String {
6          var csvString = ""
7
8          // Add the "Datum" column name to the CSV string
9          csvString += "Datum,"
10
11         // Get the value definitions with selected == true and sort them by index
12         let valueDefinitions = values?.allObjects as? [ValueDefinition] ?? []
13         let selectedValueDefinitions = valueDefinitions.filter { $0.selected }
14         let sortedValueDefinitions = selectedValueDefinitions.sorted(by: { $0.index < $1.index })
15
16         // Add the column names to the CSV string
17         let columnNames = sortedValueDefinitions.map { $0.name ?? "" }
18         csvString += columnNames.joined(separator: ",") + "\n"
19
20         // ...
46         }
47
48         return csvString
49     }
50 }
```

Obrázek 51 Vytvoření prvního řádku CSV souboru

Po vytvoření prvního řádku CSV souboru s názvy sloupců, následuje proces přidávání jednotlivých záznamů, který je zobrazen na obrázku číslo 52. Každý záznam odpovídá jednomu řádku v CSV souboru a obsahuje příslušné hodnoty pro všechny sloupce.

Prvním krokem je získání a seřazení všech záznamů ze šablony podle data pořízení. Pro každý záznam je pak proveden následující postup:

- do řetězce *csvString* je přidán datum pořízení záznamu ve formátu definovaném v aplikaci a v případě absence datumu je přidána prázdná hodnota,
- hodnoty záznamu jsou seřazeny podle indexu a přidány do CSV řetězce jako další hodnoty aktuálního řádku.

Tento postup je opakován pro všechny záznamy, dokud nejsou do CSV souboru přidány všechny záznamy. Na konci procesu je řetězec *csvString*, který nyní obsahuje kompletní CSV soubor, vrácen jako výsledek funkce *generateCSV*.

```
1  import CoreData
2  import Foundation
3
4  extension Template {
5      func generateCSV() -> String {
6          var csvString = ""
20         // ...
21
22         // Get the records and sort them by date
23         let records = self.records?.allObjects as? [Record] ?? []
24         let sortedRecords = records.sorted(by: { $0.date ?? Date() < $1.date ?? Date() })
25
26         // Add the record values to the CSV string
27         for record in sortedRecords {
28             // Add the record date to the CSV string
29             if let date = record.date {
30                 csvString += "\(date.formatted(.dateTime.day().month().year().hour().minute())),"
31             } else {
32                 csvString += ","
33             }
34
35             var rowValues: [String] = []
36
37             // Get the record values and sort them by index
38             let recordValues = record.values?.allObjects as? [RecordValue] ?? []
39             let sortedRecordValues = recordValues.sorted(by: { $0.index < $1.index })
40
41             for recordValue in sortedRecordValues {
42                 rowValues.append(recordValue.value ?? "")
43             }
44
45             csvString += rowValues.joined(separator: ",") + "\n"
46         }
47
48         return csvString
49     }
50 }
```

Obrázek 52 Přidání záznamů do CSV souboru

Poslední fází procesu exportování dat je funkce *shareTemplate*, kterou je možno vidět na obrázku číslo 53. Po stisknutí tlačítka pro sdílení na obrazovce detailu šablony je spuštěna právě tato funkce, která se stará o přípravu a sdílení CSV souboru.

Nejprve je zavolána dříve popsaná funkce *generateCSV*, která generuje řetězec CSV dat. Tento řetězec je následně převeden na objekt *Data*. Poté, pomocí *FileManager.default.temporaryDirectory*, je vytvořena dočasná cesta k souboru CSV a data jsou zapsána na toto místo. Nakonec je vytvořen *UIActivityViewController*, který je prezentován uživateli. Tato část uživatelského rozhraní nabízí různé možnosti pro sdílení vytvořeného CSV souboru, včetně uložení do souborů, sdílení s jinými aplikacemi nebo lidmi. *UIActivityViewController* je standardní součástí uživatelského rozhraní iOS a byl

podrobněji popsán v předešlé podkapitole týkající se implementace uživatelského rozhraní. Stojí za zmínku, že *UIActivityViewController* je součástí frameworku UIKit. I když je tato aplikace naprogramována v modernějším frameworku SwiftUI, *UIActivityViewController* je stále možné použít díky kompatibilitě těchto dvou frameworků.

```swift
private func shareTemplate() {
  let csvString = template.generateCSV()
  let data = Data(csvString.utf8)

  let tempURL = FileManager.default.temporaryDirectory.appendingPathComponent("Template.csv")
  do {
      try data.write(to: tempURL)
      let activityViewController = UIActivityViewController(
        activityItems: [tempURL], applicationActivities: nil
      )
      if let windowScene = UIApplication.shared.connectedScenes.first as? UIWindowScene {
          windowScene.windows.first?.rootViewController?.present(
            activityViewController, animated: true, completion: nil
          )
      }
  } catch {
      print(error.localizedDescription)
  }
}
```

Obrázek 53 Implementace sdílení obrázků do uživatelského rozhraní

V průběhu této kapitoly byly detailně popsány klíčové prvky kódu, které tvoří základní funkcionalitu prototypové aplikace. Každá z těchto ukázek kódu tvoří důležitou část v procesu digitalizace textu, což je primárním cílem aplikace. Tyto části kódu nejen, že představují technické základy aplikace, ale také reflektují teoretické koncepty a principy, které byly popsány v předchozích kapitolách.

6 VALIDAČNÍ TESTOVÁNÍ APLIKACE

Kapitola šestá se zaměřuje na proces ověření funkčnosti a použitelnosti prototypu aplikace a jeho výsledky. Jsou zde podrobně popsány testy, které byly provedeny s cílem ověřit, zda aplikace funguje podle očekávání a splňuje stanovené cíle. Dále je představena skupina testovacích uživatelů, jejichž zpětná vazba sehrála klíčovou roli v dalším vývoji a vylepšení aplikace. Kapitola také představuje metodiky použité k testování a hodnotí výsledky, které byly získány v průběhu tohoto procesu. Závěr kapitoly shrnuje klíčové poznatky získané během testování a jejich vliv na další vývoj aplikace.

6.1 Úvod k testování vytvořeného prototypu

Průběh vývoje aplikace je možno charakterizovat jako hledání jednoduchého a praktického způsobu, jak efektivně digitalizovat uživatelem definované datové struktury. V tomto procesu byl klíčový ideální teoretický přístup, který byl vytyčen na základě předchozího návrhu aplikace a postupně přizpůsoben možnostem a omezením, které nabízí zabudované nástroje pro vývoj mobilních aplikací na platformě iOS.

Tato dynamika a kreativní proces vedly k rozhodnutí implementovat testy vycházející z požadavků na aplikaci. Testování založené na požadavcích patří do kategorie validačního testování [54] a tyto testy byly zvoleny s cílem ověřit, že celá aplikace funguje podle očekávání a splňuje stanovené cíle.

6.2 Metodiky testování

Hodnocení splnění funkcionálních požadavků je přímočaré a v další části jsou zaznamenány jeho jasné výsledky – buď byly požadavky splněny, nebo ne.

Nefunkcionální požadavky se týkají klíčových aspektů aplikace, jako je rychlost odezvy či uživatelská přívětivost. Jejich měření a vyhodnocení je výrazně složitější než u funkcionálních požadavků, což si vyžádalo definování specifických parametrů testování.

Jedním z takových parametrů bylo zvolení zařízení použitého při testování. Pro ověření splnění požadavků N001, N002 a N003 bylo vybráno zařízení iPhone 13 mini. Bylo provedeno měření rychlosti odezvy aplikace na uživatelské vstupy pomocí nahrávání obrazovky, což umožnilo stanovit přesnou dobu trvání těchto interakcí.

Dále bylo nezbytné ověřit splnění požadavků N004 a N005. Pro tyto účely byla vybrána skupina tří uživatelů bez předešlých zkušeností s aplikacemi sloužícími k digitalizaci a bez

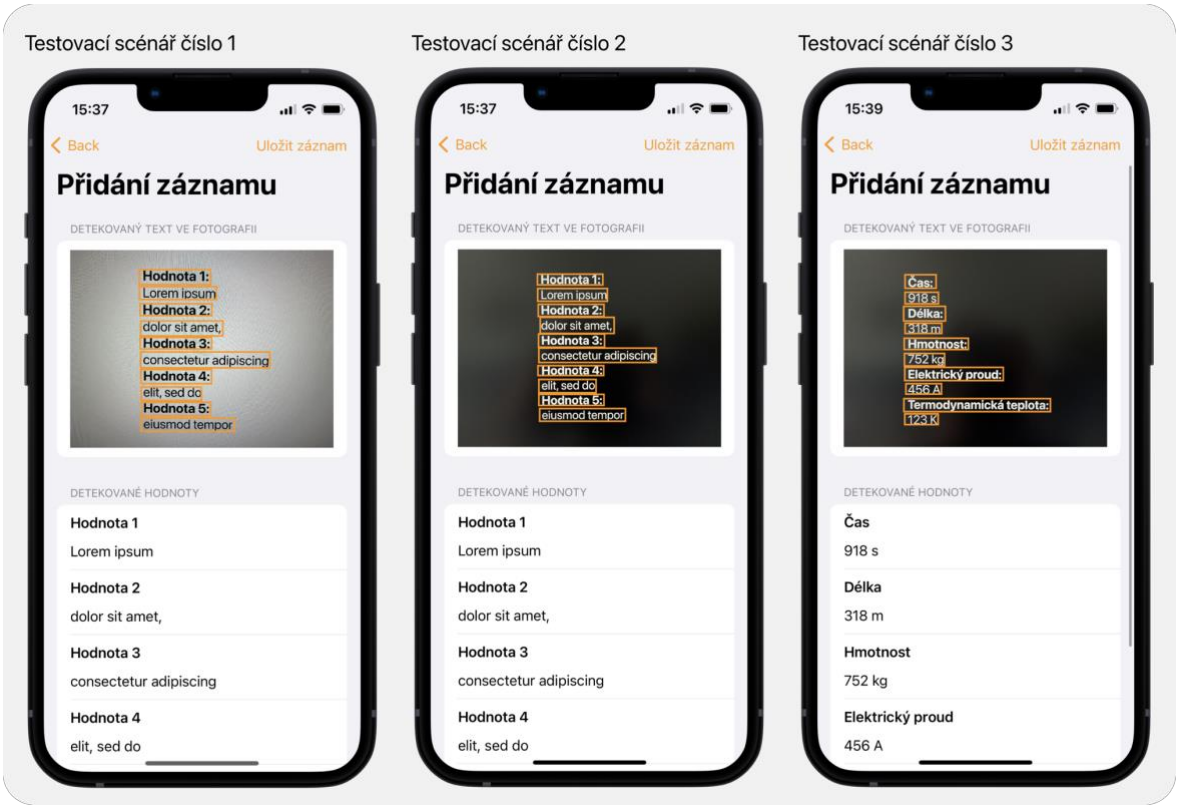
odborných znalostí v oblasti IT. Tito uživatelé byli považováni za reprezentativní vzorek cílové skupiny potenciálních uživatelů aplikace. Testování těchto dvou posledních požadavků bylo provedeno na třech různých zařízeních – iPhone 11 Pro, iPhone 13 mini a iPhone 14 Pro – aby byla zjištěna kompatibilita a přesnost testování.

Požadavek N004 hodnotil, zda uživatelé byli schopni ovládat aplikaci bez nutnosti školení. V případě, že by jeden z uživatelů nezvládl správně ovládat některou z funkcí aplikace, výsledek splnění požadavku by byl hodnocen jako negativní.

Ověření požadavku N005 proběhlo prostřednictvím tří scénářů, které jsou zobrazeny na obrázku číslo 54. V rámci každého scénáře uživatel vytvořil dvacet záznamů. Záznamy byly hodnoceny jako správné, pokud se očekávaná a digitalizovaná data shodovala. Účelem jednotlivých scénářů bylo testovat úspěšnost detekce specifických vlastností na vstupních fotografiích.

Důležité je zmínit, že aplikace umožňuje uživatelům detekovaný text upravit před tím, než jej uloží. Přestože tato funkce byla během testování dostupná, uživatelé byli instruování ji nevyužívat, aby byla zjištěna právě úspěšnost aplikace v detekci předem definovaného textu.

První testovaný scénář se zaměřuje na ověření detekce černého textu na bílém pozadí. Druhý scénář zkoumá detekci bílého textu na černém pozadí, což je inverze barevného schématu prvního scénáře. Třetí scénář byl navržen tak, aby otestoval aplikaci na datech, která nejvíce odpovídají původnímu účelu jejího vývoje. Tento scénář používá text, který nejlépe odráží skutečné využití aplikace, a tak poskytuje nejcennější informace o tom, jak efektivně aplikace plní svůj záměr v reálném provozu.



Obrázek 54 Ukázka testovacích scénářů při vytváření záznamů

6.3 Výsledky testování

V následující tabulce číslo 3 jsou zapsány výsledky uspokojení funkcionálních požadavků.

Tabulka 3 Výsledky testování funkcionálních požadavků

ID požadavku	Popis požadavku	Výsledek
F001	Uživatel bude mít možnost v aplikaci fotografovat nebo nahrávat obrázky obsahující text pro jejich zpracování.	**Splněn**
F002	Aplikace bude detekovat a extrahovat text z nahraných fotografií pomocí Vision frameworku.	**Splněn**
F003	Uživatel bude mít možnost upravit detekovaný text před jeho uložením.	**Splněn**
F004	Aplikace bude detekovaný text ukládat do databáze pomocí Core Data frameworku.	**Splněn**
F005	Aplikace bude umožňovat uživateli exportovat detekovaný text do souboru ve formátu CSV.	**Splněn**
F006	Exportovaný CSV soubor bude možné uložit na zařízení uživatele nebo sdílet přes systémové okno pro sdílení.	**Splněn**

V tabulce číslo 4 jsou zapsány výsledky testování nefunkcionálních požadavků, které budou dále popsány v textu níže.

Tabulka 4 Výsledky testování nefunkcionálních požadavků

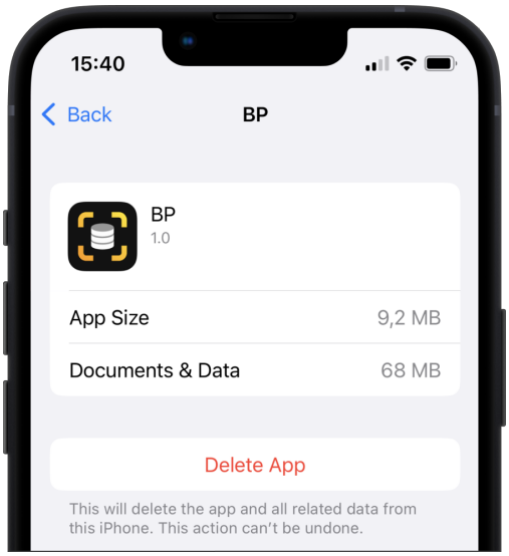
ID požadavku	Popis požadavku	Výsledek
N001	Aplikace bude reagovat na uživatelské interakce rychle, s průměrnou dobou odezvy kratší než 500 milisekund.	**Splněn**
N002	Aplikace bude schopna na testovaném zařízení zpracovat a extrahovat text ze snímku do 5 sekund od nahrání fotografie.	**Splněn**
N003	Velikost aplikace (nepočítaje databázi) nebude více než 100 MB, aby minimalizovala využití paměti na zařízení.	**Splněn**
N004	Aplikace bude navržena tak, aby uživatelé mohli efektivně využívat její funkce bez potřeby formálního školení. K efektivnímu používání budou stačit nápovědy v aplikaci.	**Splněn**
N005	Pravděpodobnost chyby při detekci a extrakci textu bude u předem definovaných testovacích fotografií menší než 3 % z důvodu minimalizace chyb ve výsledcích.	**Splněn**

6.4 Shrnutí testování aplikace

Testování prototypu aplikace prokázalo jeho plnou funkčnost a splnění jak funkcionálních, tak nefunkcionálních požadavků. Funkcionální požadavky byly jednoznačně splněny, což bylo ověřeno prostřednictvím základního ověření jednotlivých funkcí aplikace.

Testování nefunkcionálních požadavků bylo z hlediska realizace složitější, neboť tyto požadavky zahrnovaly více abstraktních aspektů, jako je rychlost odezvy a uživatelská přívětivost. První dva nefunkcionální požadavky (N001 a N002) byly otestovány měřením doby odezvy různých uživatelských interakcí. Ukázalo se, že na zařízení iPhone 13 mini je doba interakce omezena pouze dobou animací jednotlivých obrazovek SwiftUI. Nebylo při měření zaznamenáno žádné další zpoždění či načítání.

Velikost aplikace se po instalaci na zařízení iPhone 13 mini ukázala být pouhých 9,2 MB, což je výrazně pod stanoveným limitem 100 MB (požadavek N003). Tento limit aplikace nepřekročila ani po provedení všech testů a započítání dat uložených v aplikaci.



Obrázek 55 Úložiště zařízení s nainstalovanou prototypovou aplikací

U nefunkcionálního požadavku N004, který se týkal schopnosti uživatelů ovládat aplikaci bez formálního školení, bylo zjištěno, že uživatelé bez problému dokázali ovládat všechny funkce aplikace.

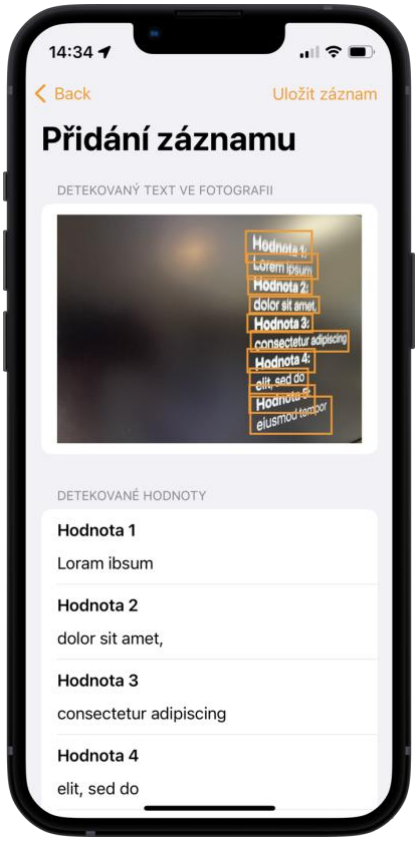
Nejnáročnější bylo ověření splnění požadavku N005, týkajícího se přesnosti detekce a extrakce textu. Za kontrolovaných světelných podmínek a při digitalizování předem definovaných scénářů byli uživatelé schopni správně detekovat 100 % vstupního textu.

V průběhu implementace aplikace bylo zaznamenáno, že úroveň správné detekce je vysoká a aplikace je tedy použitelná, avšak stoprocentní úspěšnost byla nad rámec očekávání. Dosažení tohoto výsledku může být vysvětleno několika faktory. Mezi tyto faktory se řadí použití moderních zařízení s kvalitními fotoaparáty, použití kvalitního osvětlení během testování a také snaha uživatelů pořídit co nejkvalitnější fotografie při používání aplikace.

Nejdůležitějším faktorem pro úspěšnou digitalizaci je kvalita vstupní fotografie. Při implementaci aplikace byla například pořízena fotografie, na které:

- byl úmyslně scénář vyfocen s fotoaparátem natočeným do strany, a ne přímo k displeji,
- pomocí světelných zdrojů byly vytvořeny odlesky na displeji,
- se zařízením bylo záměrně pohnuto při stisku spouště fotoaparátu pro docílení rozmazaného obrazu.

Z této fotografie, přestože byla záměrně pořízená velmi nekvalitně, aplikace úspěšně rozpoznala 96 % znaků ve fotografii a tento záznam je zobrazen na obrázku číslo 56.



Obrázek 56 Detekce textu z fotografie s nižší kvalitou

Tímto pozorováním bylo zjištěno, že existují hranice pro účinnost detekce v aplikaci. V extrémních situacích, kdy je kvalita fotografie některými faktory snížena, se mohou začít objevovat chyby v detekci. První chybou, která se začne běžně vyskytovat, je záměna podobně vypadajících písmen. Příklad této záměny byl vidět na výše zmíněném obrázku číslo 56, kde byl znak „e“ chybně identifikován jako „a“, a znak „p“, který byl chybně identifikován jako "b". Toto představuje pomyslnou hranici úspěšnosti digitalizace aplikace, která je důležitá pro pochopení efektivního využití aplikace v praxi.

7 ZHODNOCENÍ POUŽITÍ MOBILNÍCH PLATFOREM OD SPOLEČNOSTI APPLE PRO ROZPOZNÁNÍ A DIGITALIZACI TEXTU

Tato kapitola je zaměřena na zhodnocení využití mobilních platforem od společnosti Apple pro automatické rozpoznání a digitalizaci textu. Pro tento účel jsou brány v potaz především teoretické aspekty použití těchto platforem a nástrojů. Zhodnocení prototypové aplikace je předmětem další kapitoly.

7.1 Technologie a nástroje

Hlavním technologickým pilířem, na němž společnost Apple staví své nástroje pro rozpoznání a digitalizaci textu, je Vision framework. Tento univerzální framework je integrální součástí většiny operačních systémů od Apple, což mu umožňuje široké uplatnění a flexibilitu v různých aplikacích.

Vision framework je využíván také v zabudovaných aplikacích, jako jsou například aplikace Fotoaparát, Fotky nebo Poznámky, a to nejen pro identifikaci a kategorizaci objektů a lidí na fotografiích, ale také pro detekci a manipulaci s textem.

7.2 Výhody použití mobilních platforem Apple pro digitalizaci textu

Použití mobilních platforem od společnosti Apple pro digitalizaci textu má řadu výhod, které tuto technologii činí atraktivní volbou pro mnoho uživatelů a vývojářů. Tyto výhody jsou:

- **Snadná integrace do aplikací pro vývojáře**: Vision framework, který je součástí mobilních platforem od Apple, je navržen tak, aby byl snadno integrovatelný do aplikací. Díky přehledné dokumentaci a podpoře pro vývojáře je možné rychle a efektivně implementovat funkce pro rozpoznání a digitalizaci textu do aplikací.
- **Bezpečnost a soukromí**: Apple klade velký důraz na ochranu osobních údajů a soukromí uživatelů. Proto je Vision framework navržen tak, aby všechny údaje zůstaly na zařízení uživatele a nebyly sdíleny bez explicitního svolení.
- **Přístupnost na více platformách Apple**: Vision framework je dostupný napříč všemi Apple platformami, včetně iOS, iPadOS a macOS. To umožňuje vývojářům vytvářet aplikace, které mohou efektivně fungovat na různých zařízeních.

- **Optimalizace díky blízké integraci hardware a software**: Apple je známý svou těsnou integrací hardwaru a softwaru. Toto je zvlášť vidět u Vision frameworku, kde je optimalizace pro konkrétní hardware zařízení schopna významně zlepšit výkon a efektivitu procesu rozpoznání a digitalizace textu.
- **Možnost využití bez přístupu k internetu**: Jelikož je Vision framework integrován přímo do operačních systémů Apple, může díky tomu provádět rozpoznání a digitalizaci textu bez potřeby připojení k internetu. Toto je zvlášť užitečné v situacích, kdy je připojení k internetu omezené nebo nedostupné.
- **Bezplatné využití algoritmů pro rozpoznání a digitalizaci textu**: Významnou výhodou použití Vision frameworku od Apple je fakt, že využití jeho algoritmů pro rozpoznání a digitalizaci textu je zcela bezplatné. Na rozdíl od některých cloudových řešení, kde mohou být spojeny dodatečné náklady s použitím podobných funkcí, Apple tuto službu nabízí bez jakéhokoliv poplatku. To vede k výrazné úspoře nákladů při vývoji a provozu aplikací.

7.3 Nevýhody použití mobilních platforem Apple pro digitalizaci textu

Přestože použití technologií od společnosti Apple přináší řadu výhod, existují také některé nevýhody, které je třeba zvážit. Mezi ně patří:

- **Omezená integrace a interoperabilita s jinými značkami**: Operační systémy a technologie společnosti Apple jsou často velice uzavřené. Tato charakteristika může omezovat nebo znemožňovat integraci a interoperabilitu s produkty a technologiemi jiných společností.
- **Limitovaná podpora jazyků**: Vision framework podporuje v současnosti pouze 14 jazyků. Pro méně rozšířené jazyky, jako je čeština, je podpora omezená. Tato skutečnost se může projevovat například problémy s detekcí diakritických znamének a dalších specifických jazykových prvků.
- **Absence podpory segmentových displejů**: Při testování rozpoznávání segmentových displejů, provedeném v rámci hledání limitů aplikace, bylo zjištěno, že je pro tyto účely nezbytné natrénovat specifický model.

8 ZHODNOCENÍ VYTVOŘENÉ PROTOTYPOVÉ APLIKACE

Prototypová aplikace úspěšně splňuje všechny předem definované cíle a požadavky. Aplikace využívá moderní technologie jako je Vision framework a SwiftUI, které umožňují vytvoření unikátního a uživatelsky přívětivého prostředí pro snadnou a rychlou digitalizaci textových dat.

S využitím Vision frameworku, který poskytuje pokročilé funkce pro rozpoznávání a extrakci textu, aplikace dokáže automaticky digitalizovat text na fotografiích. Nejdůležitější funkcionalitou aplikace je rozpoznání struktury dat, která jsou digitalizována a jejich automatické zpracování do uživatelem vytvořené šablony. Díky využití Core Data frameworku jsou data efektivně ukládána a uživatelům je poskytnuta možnost digitalizovaný text spravovat nebo jej exportovat do formátu CSV pro další zpracování a sdílení.

V průběhu vývoje a testování aplikace byly zjištěny klíčové poznatky, které naznačují problémové oblasti a oblasti pro potenciální vylepšení. Jedním z těchto poznatků je silná závislost úspěšnosti detekce textu na kvalitě vstupní fotografie. Hlavními faktory ovlivňujícími kvalitu snímků jsou světelné podmínky a schopnosti uživatele pořídit kvalitní fotografii. Všechny získané poznatky a zpětné vazby daly najevo, že aplikace úspěšně digitalizuje text z fotografií, avšak existují oblasti, které by mohly být v budoucnu vylepšeny.

Společnost B2A Software Development, která byla zapojena do spolupráce při tvorbě zadání této bakalářské práce, vyjádřila na aplikaci pozitivní ohlasy. Ing. Roman Manďák, vedoucí vývoje ve společnosti B2A Software Development, se k práci vyjádřil následovně: *„Bakalářská práce se zabývá aktuálním tématem digitalizace dat. Výsledkem je aplikace, která velmi snadně a efektivně zaznamenává data v digitální podobě. Ta je následně možné v různých formátech distribuovat dále. Získané zkušenosti mají velký přínos pro naše zákazníky v oblasti digitalizace výroby a můžeme je aplikovat v reálné praxi například pro čtení dat z displejů strojů, které nemají rozhraní pro poskytování dat.“*

Význam a kvalita této práce byly dále potvrzeny v rámci soutěže Studentská tvůrčí a odborná činnost (STOČ), kde aplikace získala první místo v kategorii Informační systémy. Toto ocenění svědčí o schopnosti aplikace řešit reálné potřeby a zároveň o zájmu odborné veřejnosti o tuto praktickou metodu digitalizace. Dvě ze společností, jejichž zástupci byli součásti poroty, o aplikaci projevily zájem.

Celkově lze shrnout, že vytvořená aplikace je úspěšným příkladem propojení teoretických znalostí s praktickou implementací, což vytváří v praxi vysoce využitelný a efektivní způsob digitalizace textu. S ohledem na splněné cíle a požadavky, úspěšné výsledky testování a další způsoby validace, je jasné, že aplikace může nabídnout uživatelům přidanou hodnotu a usnadnit proces digitalizace dat z fotografií.

ZÁVĚR

Bakalářská práce se detailně zabývala problematikou vývoje mobilních aplikací pro rozpoznání a digitalizaci textu v rámci platformy iOS. Práce byla strukturována do dvou hlavních částí – teoretické a praktické.

Teoretická část se věnovala popisu tří hlavních oblastí. V první části bylo charakterizováno rozpoznávání textu, konkrétně jeho principy, dělení a případy užití. Součástí této kapitoly byl také popis existujících aplikací, které jsou typickými zástupci pro daný případ užití. Druhá oblast byla zaměřena na mobilní platformy od společnosti Apple. Ve třetí oblasti byly popsány nástroje a knihovny používané k vývoji aplikací pro rozpoznání a digitalizaci textu.

V rámci praktické části byl navržen a implementován prototyp aplikace, který zobrazuje možnosti využití technologií popsaných v teoretické části. Po implementaci návrhu byla prototypová aplikace otestována, kde byla ověřena její funkčnost a spolehlivost. Následně byla zhodnocena možnost využití mobilních platforem Apple pro digitalizaci textu, přičemž byly zváženy jak výhody, tak i potenciální omezení tohoto přístupu. V závěru praktické části byla zhodnocena celková funkčnost a účinnost vytvořeného prototypu aplikace.

Výsledkem práce je úspěšná integrace teoretických poznatků z oblasti vývoje iOS aplikací a technologie rozpoznávání textu do praktického procesu tvorby prototypové aplikace. Práce předává ucelený pohled na využití mobilních platforem společnosti Apple pro digitalizaci textu. Zároveň prezentuje unikátní přístup k digitalizaci dat, který byl představen v rámci vytvořeného prototypu aplikace. Tímto se bakalářská práce stává cenným přínosem pro pochopení možností a potenciálu, které nabízí mobilní platformy společnosti Apple v oblasti rozpoznávání a digitalizace textu.

SEZNAM POUŽITÉ LITERATURY

[1] What Is Optical Character Recognition (OCR)?. *IBM* [online]. 2022 [cit. 2023-05-09]. Dostupné z: https://www.ibm.com/cloud/blog/optical-character-recognition

[2] OCR - Optical Character Recognition. *Microsoft Learn* [online]. 2023 [cit. 2023-05-09]. Dostupné z: https://learn.microsoft.com/en-us/azure/cognitive-services/computer-vision/overview-ocr

[3] What is OCR and why is OCR software important?. *Adobe* [online]. 2023 [cit. 2023-05-09]. Dostupné z: https://www.adobe.com/acrobat/guides/what-is-ocr.html

[4] Text recognition v2. *Google for Developers | ML Kit* [online]. 2023 [cit. 2023-05-09]. Dostupné z: https://developers.google.com/ml-kit/vision/text-recognition/v2

[5] What Is OCR (Optical Character Recognition)?. *AWS* [online]. 2023 [cit. 2023-05-09]. Dostupné z: https://aws.amazon.com/what-is/ocr/

[6] How to scan documents on your iPhone or iPad. *Apple Support* [online]. 2023 [cit. 2023-05-09]. Dostupné z: https://support.apple.com/en-us/HT210336

[7] Hi, I'm Cliff Weitzman. *Speechify* [online]. 2023 [cit. 2023-05-09]. Dostupné z: https://speechify.com/about/?landing_url=https%3A%2F%2Fspeechify.com%2Ftext-to-speech-ios%2F

[8] COHEN, Jason. 4 Companies Control 67% of the World's Cloud Infrastructure. *PCMag* [online]. 2021 [cit. 2023-05-09]. Dostupné z: https://www.pcmag.com/news/four-companies-control-67-of-the-worlds-cloud-infrastructure

[9] Amazon Textract. *Amazon Web Services* [online]. 2023 [cit. 2023-05-09]. Dostupné z: https://aws.amazon.com/textract/

[10] Anthem Enables Intelligent Claims Processing Using Amazon Textract. *Amazon Web Services* [online]. 2021 [cit. 2023-05-09]. Dostupné z: https://aws.amazon.com/solutions/case-studies/anthem/

[11] Fujitsu. *Forbes* [online]. 2023 [cit. 2023-05-09]. Dostupné z: https://www.forbes.com/companies/fujitsu/

[12] Fujitsu delivers up to 99.9 percent scanning accuracy with Azure AI form and character recognition. *Microsoft Customer Stories*[online]. 2022 [cit. 2023-05-09]. Dostupné z: https://customers.microsoft.com/en-us/story/1504311236437869486-fujitsu-document-scanning-azure-form-recognizer

[13] OCR (Optical Character Recognition) with world-class Google Cloud AI. *Google Cloud* [online]. [cit. 2023-05-09]. Dostupné z: https://cloud.google.com/use-cases/ocr

[14] Document AI. *Google Cloud* [online]. [cit. 2023-05-09]. Dostupné z: https://cloud.google.com/document-ai

[15] Picture what the cloud can do: How the New York Times is using Google Cloud to find untold stories in millions of archived photos. *Google Cloud* [online]. 2018 [cit. 2023-05-09]. Dostupné z: https://cloud.google.com/blog/products/ai-machine-learning/how-the-new-york-times-is-using-google-cloud-to-find-untold-stories-in-millions-of-archived-photos

[16] HILLIARD, Wesley. At 2 billion iPhones sold, Apple continues to redefine what customers want. *AppleInsider* [online]. 2021 [cit. 2023-05-09]. Dostupné z: https://appleinsider.com/articles/21/09/22/at-2-billion-iphones-sold-apple-continues-to-redefine-what-customers-want

[17] LISHCHUK, Ruslana. 83+ Apple Statistics in 2022: Sales and Usage. *MacKeeper* [online]. 2022 [cit. 2023-05-09]. Dostupné z: https://mackeeper.com/blog/apple-statistics/

[18] COSTELLO, Sam. The History of iOS, from Version 1.0 to 16.0. *Lifewire* [online]. 2022 [cit. 2023-05-09]. Dostupné z: https://www.lifewire.com/ios-versions-4147730

[19] LEFEBVRE, Rob. The iPadOS Versions Guide. *Lifewire* [online]. 2022 [cit. 2023-05-09]. Dostupné z: https://www.lifewire.com/what-is-ipados-4691712

[20] FINGAS, Roger. Apple Watch runs 'most' of iOS 8.2, may use A5-equivalent processor. *AppleInsider* [online]. 2015 [cit. 2023-05-09]. Dostupné z: https://appleinsider.com/articles/15/04/23/apple-watch-runs-most-of-ios-82-may-use-a5-equivalent-processor

[21] RITCHIE, Rene. History of the App Store. *IMore* [online]. 2017 [cit. 2023-05-09]. Dostupné z: https://www.imore.com/history-app-store

[22] The apps you love. From a place you can trust. *Apple* [online]. 2023 [cit. 2023-05-09]. Dostupné z: https://www.apple.com/app-store/

[23] BAGGOTT, Alex. Every Apple App Store fee, explained: How much, for what, and when. *AppleInsider* [online]. 2023 [cit. 2023-05-09]. Dostupné z: https://appleinsider.com/articles/23/01/08/the-cost-of-doing-business-apples-app-store-fees-explained

[24] What is an API?. *IBM* [online]. [cit. 2023-05-09]. Dostupné z: https://www.ibm.com/topics/api

[25] MAYO, Benjamin. Apple announces new Xcode, 'Swift' programming language. *9to5Mac*[online]. 2014 [cit. 2023-05-09]. Dostupné z: https://9to5mac.com/2014/06/02/apple-announces-new-xcode-swift-programming-language/

[26] MAYO, Benjamin. Apple's Swift programming language is now open source. *9to5Mac*[online]. 2015 [cit. 2023-05-09]. Dostupné z: https://9to5mac.com/2015/12/03/apple-swift-programming-language-open-source/

[27] About Swift. *Swift Documentation* [online]. 2023 [cit. 2023-05-09]. Dostupné z: https://docs.swift.org/swift-book/documentation/the-swift-programming-language/aboutswift/

[28] A Swift Tour. *Swift Documentation* [online]. 2023 [cit. 2023-05-09]. Dostupné z: https://docs.swift.org/swift-book/documentation/the-swift-programming-language/guidedtour/

[29] Structures and Classes. *Swift Documentation* [online]. 2023 [cit. 2023-05-09]. Dostupné z: https://docs.swift.org/swift-book/documentation/the-swift-programming-language/classesandstructures/

[30] Enumerations. *Swift Documentation* [online]. 2023 [cit. 2023-05-09]. Dostupné z: https://docs.swift.org/swift-book/documentation/the-swift-programming-language/enumerations/

[31] Concurrency. *Swift Documentation* [online]. 2023 [cit. 2023-05-09]. Dostupné z: https://docs.swift.org/swift-book/documentation/the-swift-programming-language/concurrency/

[32] Protocols. *Swift Documentation* [online]. 2023 [cit. 2023-05-09]. Dostupné z: https://docs.swift.org/swift-book/documentation/the-swift-programming-language/protocols/

[33] Error Handling. *Swift Documentation* [online]. 2023 [cit. 2023-05-09]. Dostupné z: https://docs.swift.org/swift-book/documentation/the-swift-programming-language/errorhandling/

[34] Generics. *Swift Documentation* [online]. 2023 [cit. 2023-05-09]. Dostupné z: https://docs.swift.org/swift-book/documentation/the-swift-programming-language/generics/

[35] Build Your First App in Swift and SwiftUI. *AppCoda* [online]. [cit. 2023-05-09]. Dostupné z: https://www.appcoda.com/learnswift/swiftui-basics.html

[36] Creating and Combining Views. *Apple Developer* [online]. 2023 [cit. 2023-05-09]. Dostupné z: https://developer.apple.com/tutorials/swiftui/creating-and-combining-views

[37] View configuration. *Apple Developer* [online]. 2023 [cit. 2023-05-09]. Dostupné z: https://developer.apple.com/documentation/swiftui/view-configuration

[38] Stack Data Structure. *Programiz* [online]. [cit. 2023-05-09]. Dostupné z: https://www.programiz.com/dsa/stack

[39] HUDSON, Paul. What's the difference between @ObservedObject, @State, and @EnvironmentObject?. *HACKING WITH SWIFT* [online]. 2021 [cit. 2023-05-09]. Dostupné z: https://www.hackingwithswift.com/quick-start/swiftui/whats-the-difference-between-observedobject-state-and-environmentobject

[40] UIKit vs. Swiftui. *Bluewhale* [online]. 2023 [cit. 2023-05-09]. Dostupné z: https://bluewhaleapps.com/blog/uikit-vs-swiftui

[41] UIKit. *Apple Developer* [online]. 2023 [cit. 2023-05-09]. Dostupné z: https://developer.apple.com/documentation/uikit

[42] About App Development with UIKit. *Apple Developer* [online]. 2023 [cit. 2023-05-09]. Dostupné z: https://developer.apple.com/documentation/uikit/about_app_development_with_uikit

[43] UIKit integration. *Apple Developer* [online]. 2023 [cit. 2023-05-09]. Dostupné z: https://developer.apple.com/documentation/swiftui/uikit-integration

[44] Vision. *Apple Developer* [online]. 2023 [cit. 2023-05-09]. Dostupné z: https://developer.apple.com/documentation/vision

[45] Detecting Objects in Still Images. *Apple Developer* [online]. 2023 [cit. 2023-05-09]. Dostupné z: https://developer.apple.com/documentation/vision/detecting_objects_in_still_images

[46] Recognizing Text in Images. *Apple Developer* [online]. 2023 [cit. 2023-05-09]. Dostupné z: https://developer.apple.com/documentation/vision/recognizing_text_in_images

[47] Core Data. *Apple Developer* [online]. 2023 [cit. 2023-05-09]. Dostupné z: https://developer.apple.com/documentation/coredata

[48] BOHON, Cory. Xcode Through the Years. *MartianCraft* [online]. 2022 [cit. 2023-05-09]. Dostupné z: https://martiancraft.com/blog/2022/01/xcode-through-the-years/

[49] SF Symbols 4. *Apple Developer* [online]. 2023 [cit. 2023-05-09]. Dostupné z: https://developer.apple.com/sf-symbols/

[50] Virtual whiteboard for sketching hand-drawn like diagrams. *GitHub* [online]. 2023 [cit. 2023-05-09]. Dostupné z: https://github.com/excalidraw/excalidraw

[51] Pricing. *Figma* [online]. 2023 [cit. 2023-05-09]. Dostupné z: https://www.figma.com/pricing/

[52] GEMAYEL, Toni. How to wireframe. *Figma* [online]. 2019 [cit. 2023-05-09]. Dostupné z: https://www.figma.com/blog/how-to-wireframe/

[53] Designing for iOS. *Apple Developer* [online]. 2023 [cit. 2023-05-09]. Dostupné z: https://developer.apple.com/design/human-interface-guidelines/designing-for-ios

[54] SOMMERVILLE, Ian. Softwarové inženýrství. Brno: Computer Press, 2013. ISBN 978-80-251-3826-7.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

API	Application Programming Interface
AWS	Amazon Web Services
CSV	Comma-separated values
GCP	Google Cloud Platform
GUI	Graphical User Interface
ICR	Intelligent Character Recognition
IDE	Integrated Development Environment
IWR	Intelligent Word Recognition
Lo-Fi	Low fidelity
MVC	Model-View-Controller
OCR	Optical Character Recognition
TTS	Text-to-speech
UI	User interface
UX	User experience
WWDC	World Wide Developer Conference

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH