VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

MOBILNÍ APLIKACE PRO PODPORU EVIDENCE MAJETKU

THESIS TITLE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **DANIEL PONÍŽIL**
AUTHOR

VEDOUCÍ PRÁCE **prof. Ing. ADAM HEROUT, Ph.D.**
SUPERVISOR

BRNO 2023

Abstrakt

Cílem bakalářské práce bylo vytvořit mobilní aplikaci, která by usnadnila proces evidování nového majetku do evidenčního softwaru. Aplikace byla vytvořena ve frameworku React Native s využitím platformy Expo. Samotný vývoj a testování pak probíhalo se zaměřením na platformu Android. Aplikace umožňuje naskenovat sériová čísla nového majetku pomocí čárových kódů a QR kódů. K naskenovaným číslům se vybere druh vytvářeného objektu a vyplní hodnoty jednotlivých vlastností. Pro vyplnění hodnot vlastností lze také využít technologii OCR. Objekty se pak uložením vytvoří v evidenčním systému. Aplikace vychází z provedených analýz příběhu naskladnění nového majetku, nedostatků aktuálního řešení a konkurenčních produktů.

Abstract

The goal of the bachelor's thesis was to create a mobile application that would simplify the registration process of new assets into an inventory software. The application was developed using the React Native framework with the use of the Expo platform. The development and testing process was mainly focused on the Android platform. The application allows scanning serial numbers of new assets using barcodes and QR codes. The type of object being created is selected for the scanned numbers, and the values of individual properties are filled in. OCR technology can also be used to fill in the values of properties. The objects are then created in the inventory system by saving. The application is based on analyses of the story of stocking new assets, deficiencies of the current solution, and competitive products.

Klíčová slova

mobilní aplikace, React Native, multiplatformní, Android, evidování majetku, správa majetku, čárové kódy, QR kódy, OCR

Keywords

mobile application, React Native, cross-platform, Android, asset registration, asset management, barcodes, QR codes, OCR

Citace

PONÍŽIL, Daniel. *MOBILNÍ APLIKACE PRO PODPORU EVIDENCE MAJETKU.* Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Adam Herout, Ph.D.

MOBILNÍ APLIKACE PRO PODPORU EVIDENCE MAJETKU

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana prof. Ing. Adama Herouta Ph.D. Další informace mi poskytla firma Alvao. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Daniel Ponížil
16. května 2023

Poděkování

Rád bych poděkoval panu profesoru Adamu Heroutovi, mému vedoucímu, za jeho neocenitelné rady a trpělivost při konzultacích. Dále bych chtěl vyjádřit svůj vděk společnosti Alvao za poskytnutí tématu k mé bakalářské práci. Zde také patří poděkování všem kolegům ze zaměstnání, především tedy Ing. Filipovi Jandorovi, který působil jako hlavní konzultant této práce ze strany firmy. Mým kamarádům studentům bych chtěl poděkovat za jejich pravidelné rady a odpovědi na mé otázky. Nakonec, mé poděkování patří samozřejmě mé rodině a známým, kteří mě během tvorby této práce a celého studia neustále podporovali.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Na kvalitní evidenci a správu majetku se v mnoha oborech a odvětvích klade velký důraz. Správné ukládání dat o majetku a jeho správa jsou klíčovými prvky v mnoha oblastech, jako je podnikání nebo státní správa a jsou základem pro správné fungování IT oddělení v organizacích.

Některá řešení pro evidování majetku mohou být značně nepohodlná pro práci. Použití kancelářských aplikací nebo dokonce papírových záznamů může být velice nepřehledné, navíc s sebou přináší riziko chyb a ztráty důležitých dat o majetku. Takováto řešení navíc často neumožňují jednoduché vyhledávání, orientaci a správu majetku. Naštěstí si podnikatelé a organizace mohou vybrat z několika softwarových produktů, které jim s tímto problémem pomůžou.

Cílem této bakalářské práce je vytvořit moderní a uživatelsky přívětivou mobilní aplikaci, která usnadní proces evidování nového hmotného majetku do jednoho ze softwarových produktů pro správu majetku. Uživatelé mohou jednoduše naskenovat všechna sériová čísla jednoho druhu nového majetku pomocí fotoaparátu svého mobilního zařízení, hromadně jim vyplnit jejich společné informace a následně data uložit do databáze pomocí komunikace přes REST API. Aplikace také obsahuje funkcionalitu optického rozpoznávání znaků, kterou uživatelé mohou využít při hromadném vyplňování informací, což umožňuje pohodlnější a bezchybové vyplňování údajů z různých zdrojů, jako jsou například faktury nebo dodací listy.

Tato bakalářská práce je rozdělena do sedmi hlavních kapitol. Druhá kapitola se zabývá teoretickým základem pro vývoj mobilních aplikací, technologiemi, frameworky a dalšími použitými prvky v této práci, jako optické rozpoznávání znaků (OCR), REST API a komunikace s databází. Třetí kapitola se zaměřuje na problematiku evidování majetku, současné metody, přístupy konkurence k tomuto problému a také čárové kódy a QR kódy a jejich využití při evidování majetku. Čtvrtá kapitola se věnuje analýze požadavků na aplikaci a průzkumu konkurenčních řešení. V páté kapitole je popsán návrh aplikace s uživatelským rozhraním. Šestá kapitola popisuje implementaci aplikace, jako je skenování sériových čísel, práci s daty od databáze a implementaci technologie OCR. Zároveň se také zabývá způsobem testování aplikace a zpracováním výsledků testů. Sedmá kapitola zhodnocuje dosažení jednotlivých cílů této práce a možností jejího budoucího rozvoje. Na závěr je přiložena použitá literatura.

Kapitola 2

Vývoj mobilní aplikace

V této kapitole se popisují jednotlivé technologie použité při vývoji aplikace převážně pro zařízení Android, proč došlo k jejich zvolení a porovnání s dalšími technologiemi, které šlo při práci použít.

2.1 Mobilní aplikace a technologie pro jejich vývoj

Mobilní aplikace jsou softwarové programy určené pro běh na mobilních zařízeních, jako jsou telefony a tablety [30]. Způsob vývoje se pak odvíjí od operačního systému zařízení, na které je aplikace cílená. V této bakalářské práci byla aplikace primárně vyvíjena a laděna pro mobilní telefon s operačním systémem Android[1], nicméně plánem budoucího rozvoje je aplikaci vyladit minimálně také pro iOS[2] zařízení.

Operační systém Android

Android je operační systém vyvinutý společností Google pro využití v dotykových zařízeních, jako jsou například chytré telefony nebo tablety. Kromě toho však Android lze najít i v jiných zařízeních. Používá se totiž v mnoha televizích, autech či chytrých hodinkách.

Systém byl jako první vyvinut společností Android Inc., která byla koupena společností Google v roce 2005. Oficiální vydání na trh proběhlo v roce 2008, nicméně komerčně se na zařízení dostal již o rok dříve. V dnešní době patří Android mezi nejpoužívanější operační systémy na světě.

Pro vývoj Android aplikací lze použít například jazyk Kotlin, který preferuje i samotná společnost Google a je tedy oficiálním jazykem pro vývoj na Androidu [22]. Tento jazyk vyvíjí a udržuje společnost JetBrains[3].

Výhody vývoje pro Android zařízení

- Android je otevřený software (anglicky open-source) a díky tomu má opravdu rozsáhlou komunitu, která zajišťuje kvalitní podporu
- Pro vývoj intuitivních aplikací lze využít pokyny přímo od společnosti Google
- Díky fragmentaci mohou aplikace spouštět 2 činnosti na jedné obrazovce

[1] https://www.android.com
[2] https://www.apple.com/ios
[3] https://www.jetbrains.com

- Je jednodušší uvést aplikaci do obchodu Google Play než například do obchodu Apple App Store

Nevýhody vývoje pro Android zařízení

- Fragmentace může vývojářům dělat potíže s dobrým přizpůsobením pro různé velikosti obrazovky
- Vzhledem k velkému množství zařízení s Androidem je náročné aplikace otestovat a tedy zaručit správnou funkcionalitu na všech zařízeních
- Náročnější vývoj a testování pak může navýšit cenu vývoje dané aplikace

Multiplatformní a nativní vývoj

V **multiplatformním vývoji** jde o vytvoření aplikace spustitelné na více operačních systémech z jednoho zdrojového kódu [32]. Aplikace jsou vytvářeny pomocí multiplatformních rámců neboli frameworků a jsou vytvářeny privátními společnostmi.Mezi tyto nástroje patří například React Native od společnosti Meta, Flutter od Googlu nebo Xamarin od Microsoft.

Mezi hlavní výhody multiplatformního vývoje patří primárně doba potřebná pro vývoj aplikace. Ta je totiž o poznání kratší, než kdyby se aplikace musela programovat pro každý operační systém zvlášť. S tím také souvisí nižší náklady. Další výhodou je zajisté také jednodušší údržba a znovupoužitelnost kódu.

Naopak nevýhodou je pak rychlost běhu vytvořené aplikace. Multiplatformní aplikace jsou obvykle větší, tedy zabírají více místa. Také mohou mít problém s přístupem ke všem hardwarovým funkcím a nativním komponentám.

Nativní přístup k vývoji se zaměřuje na vývoj aplikace zvlášť pro každou platformu, neboli operační systém. Systémy mají své vlastní vývojové nástroje a jazyky, což zajišťuje maximální výkon a kompatibilitu s operačním systémem. Například pro Android se velmi často používá Java nebo Kotlin, naopak u iOS je to pak populární jazyk Swift nebo Objective-C. Pro vývoj také potřebujeme vývojové prostředí IDE (anglicky integrated development environment). Jako integrované vývojové prostředí pro Android aplikace se používá Android Studio a pro iOS aplikace Xcode.

Výhodou nativního vývoje je, že aplikace zpravidla dodávají skvělou uživatelskou zkušenost, neboli UX (anglicky user experience). Jsou také výkonnější, protože se vytváří a optimalizují vždy pro určitou platformu. Oproti multiplatformním aplikacím mají také lepší zabezpečení dat a škálovatelnost, tedy schopnost aplikace růst a zvládat větší zatížení. Nativní aplikace je také jednodušší publikovat do jednotlivých obchodů, jako je Google Play nebo Apple App Store.

Větší náklady na vývoj jsou pak jasnou nevýhodou, pokud je potřeba vytvořit nativní aplikaci zároveň pro Android i pro iOS. Vývoj zabere delší čas a kód aplikace není jeden, ale je rozdělen do projektů podle cílových platforem.

2.2 React Native, Expo a důvody zvolení

V této části je popsána základní teorie pro zvolený framework React Native a platformu Expo.

2.2.1 React Native

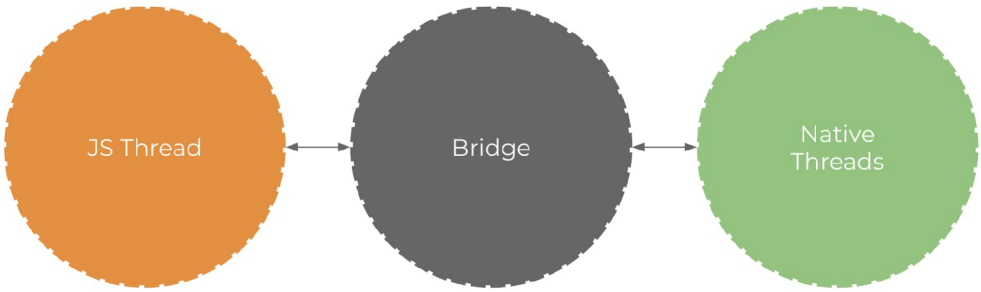
React Native[4] je populární framework, který byl vyvinut a zveřejněn společností Meta v roce 2015. Za hlavní cíl tohoto frameworku je umožnit vývojářům vytvářet nativní mobilní aplikace z jednoho zdrojového kódu kombinací JavaScriptu a Reactu pro zařízení s operačním systémem Android a iOS. Samotný React je jednou z nejlepších knihoven JavaScriptu pro vytváření uživatelských rozhraní. Primárním zdrojem pro nadcházející teorii byla dokumentace React Native [25].

Základní prvky architektury React Native

Do architektury React Nativu patří následující hlavní části: Běhové prostředí JavaScriptu (anglicky JavaScript runtime environment), Most (anglicky Bridge), Nativní moduly a komponenty [16].

1. **Běhové prostředí JavaScriptu** je zodpovědné za vykonávání JavaScriptového kódu aplikace. Provádí veškerou logiku aplikace a může ovlivnit, jak běží.

2. **Most** je klíčová část architektury, jejímž cílem je umožnit komunikaci mezi JavaScriptovou a nativní částí cílové platformy. Most je tedy koncept, který poskytuje obousměrnou a asynchronní komunikaci mezi těmito dvěma částmi [10].



Obrázek 2.1: Obrázek znázorňuje komunikaci mezi JavaScriptovým a nativním vláknem pomocí mostu. Obrázek je převzat ze stránek od HackerNoon[9].

3. **Nativní moduly** jsou specifické pro každou platformu a umožňují vývojářům přistupovat k rozhraní API (zkratka pro Application Programming Interface) nativní platformy, které není ve výchozím nastavení v JavaScriptu k dispozici a je tedy dostupné jen na dané platformě iOS nebo Android. Umožňují například použití knihoven z Objective-C či Javy, aniž by bylo potřeba je znova psát v JavaScriptu [26].

4. **Nativní komponenty** v React Native jsou základní stavební prvky uživatelského rozhraní, které jsou podporovány operačními systémy Android a iOS. React Native

[4] https://reactnative.dev

umožňuje vytvářet tyto komponenty pomocí JavaScriptu a Reactu. Během běhu aplikace React Native používá odpovídající Android a iOS pohledy (anglicky views) pro tyto komponenty, což zajišťuje výsledný vzhled a chování aplikace srovnatelné s aplikacemi vytvořenými pomocí nativních jazyků. Tento přístup tak umožňuje vývojářům vytvářet kvalitní aplikace s výhodami dobrého výkonu a univerzálního kódu pro obě platformy [23].

React Native obsahuje základní sadu již připravených nativních komponent, které se dají okamžitě využít. Jedná se o základní komponenty (anglicky Core Components) React Native a některé z nich jsou:

- View – Kontejner podporující rozložení pomocí flexboxu[5], styly, dotykové ovládací prvky a prvky přístupnosti. Obdobou je neposouvající div u webů.
- Text – Komponenta, která zobrazuje a stylizuje text, zvládne také některé dotykové události
- Image – Komponenta sloužící k zobrazení obrázků
- ScrollView – Skrolovací kontejner obsahují jednu či více komponent nebo view kontejnerů
- TextInput – Komponenta umožňující uživateli zadat text a ten dál zpracovat

React Native také vývojářům dovoluje vytvářet své vlastní nativní komponenty pro své potřeby. Tomu napomáhá obrovská komunita, takže je z čeho vybírat.

Je také nutné podotknout, že v době vypracovávání této Bakalářské práce se pracuje na nové architektuře, která prozatím ještě není stabilní. V nové architektuře se bude pracovat s jinými typy nativních modulů a komponent pro dosažení obdobných výsledků. Budou to Turbo Native Modules[6] a Fabric Native Components[7]

Další důležité prvky React Native

Stavy (anglicky states) a vlastnosti (anglicky props) jsou dva způsoby dat, které řídí komponentu. Stavy slouží pro správu dynamických dat v komponentách. Pokud v komponentě potřebujeme používat data, která se mění, používáme pro to stavy. Stavy by se měly pravidelně inicializovat v konstruktoru třídní komponenty nebo pomocí `useState` a pro změnu stavu se volá `setState`. Pokud dojde ke změně stavu komponenty, tak ji React Native opět vykreslí. K hodnotám uloženým ve stavech můžeme přistupovat pomocí `this.state` u třídních komponent a nebo pomocí `useState`. Jako příklad, kdy stav můžeme nastavit, lze uvést získání dat z uživatelského vstupu (například pomocí komponenty TextInput) nebo ze serveru. Pro správu toku dat lze také využít různé nástroje, jako jsou kontejnery Redux nebo MobX. Ty se pak používají pro modifikaci stavů místo přímého volání `setState`.

Styly (anglicky styles) se používají pro stylování a uspořádání komponent na obrazovce. Všechny základní komponenty v React Native přijímají vlastnost (anglicky prop) s názvem `style`. Názvy a hodnoty stylů obvykle připomínají fungování CSS (kaskádové styly) u webů a použít je můžeme několika způsoby. Jedním z nich je

[5] https://reactnative.dev/docs/flexbox
[6] https://reactnative.dev/docs/the-new-architecture/pillars-turbomodules
[7] https://reactnative.dev/docs/the-new-architecture/pillars-fabric-components

použití inline styly, které najdeme definované v komponentách pod vlastností `style`. Druhou variantou je pak použít tzv. StyleSheet, který nám umožňuje definovat více stylů na jednom místě a mimo nějakou komponentu. Díky tomu bude náš kód čitelnější a jednou definovaný styl pak můžeme použít ve více komponentách. Další variantou jak komponenty mohou získat styl je zdědit je z rodičovské komponenty. Výhodou také je, že můžeme vytvářet styly pro konkrétní platformy. Můžeme využít modul zvaný Platform module[8] a použít Platform.OS nebo metodu Platform.select pro nastavení červeného pozadí pro iOS zařízení a zelené pozadí pro Android zařízení.

Pro uspořádání a rozvržení komponent na obrazovce se používá Flexbox (celým názvem Flexible Box Layout), což je jednoduchý systém pro manipulaci s komponentami a jeho chování je stejné jako u Flexboxu v CSS. Kromě umístění a velikosti můžeme u komponent upravovat také například barvu, okraje, stíny (vizuální efekty) a další. V následujícím výpisu 2.1 můžete vidět příklad nastavení barvy pozadí podle specifické platformy a využití StyleSheet.

```
1  import {Platform, StyleSheet} from 'react-native';
2
3  const styles = StyleSheet.create({
4    container: {
5      flex: 1,
6      ...Platform.select({
7        ios: {
8          backgroundColor: 'red',
9        },
10       android: {
11         backgroundColor: 'green',
12       },
13       default: {
14         // other platforms, web for example
15         backgroundColor: 'blue',
16       },
17     }),
18   },
19 });
```

Výpis 2.1: V ukázce můžeme vidět nastavení různých stylů pozadí podle platformy, kterou získáme z `Platform.select`. Abychom mohli takto styly nastavit, musí být na začátku importovány jejich odpovídající moduly. Kód je přejat z dokumentace React Native [27]

Fast refresh je velmi oblíbená funkce React Native, která umožňuje vývojářům vidět změny okamžitě (například změna barvy pozadí). Funguje to tak, že pokud upravíme třeba barvu pozadí komponenty, funkce Fast refresh aktualizuje a znova vykreslí jen tuto komponentu. Pokud ale upravíme modul s exporty, které nejsou komponenty v Reactu, Fast Refresh pak aktualizuje nejen tento modul, ale také moduly, které ho importují. Dá se tedy říct, že Fast Refresh zjistí, kam až změna zasahuje a podle toho aktualizuje potřebné komponenty a moduly [24].

[8] https://reactnative.dev/docs/platform-specific-code

2.2.2 Expo

Expo[9] je otevřená (anglicky open-source) platforma postavená okolo React Native umožňující vytvářet nativní aplikace pro Android, iOS a web pomocí JavaScriptu a Reactu. Poskytuje nám soubor nástrojů, které ještě více zrychlují a zjednodušují proces vývoje v React Native. Následují důležité prvky platformy Expo:

Expo CLI je nástrojem příkazového řádku a je kostrou rozhraní mezi vývojářem a ostatními nástroji Expa [7]. Umožňuje nám vytvořit nový React Native projekt se základní konfigurací pro zahájení vývoje. Spouští také vývojový server, přes který mohou vývojáři přistupovat k aplikaci a testovat okamžitě úpravy, které v aplikaci provedou. Expo CLI se dá rovněž použít pro instalaci a aktualizaci balíčků použitím příkazu `npx expo install`, jež slouží jako náhrada `npm install`, což je základní příkaz pro instalaci balíčků pro JavaScript.

Expo SDK je sada vývojových nástrojů (anglicky software development kit) poskytující přístup k zařízení a systémovým funkcím, tedy fotoaparátu, gyroskopu, GPS a dalším. Nástroje lze použít po nainstalování výše zmíněným příkazem `npx expo install` [8].

Expo Go je mobilní aplikace usnadňující vývojářům testování vyvíjené aplikace. Vývojář se z této aplikace připojí ke svému projektu a vyvíjenou aplikaci si přes Expo Go jednoduše otevře ve svém mobilním zařízení. Po úpravě kódu vyvíjené aplikace se změna hned projeví na mobilním zařízení. Pokud by vývojář nechtěl používat fyzický telefon k testování vyvíjených aplikací, může projekt spojit s emulátorem (simulace mobilního zařízení na počítači) z Android Studia[10] nebo Xcode[11].

Expo managed workflow a bare workflow jsou dva základní způsoby vývoje. S **managed workflow** se při programování používá jen JavaScript či TypeScript a nástroje Expa se postarají o vše ostatní. Používá se přitom nástroj Expo CLI na počítači a vývojový klient na mobilním zařízení (například Expo Go). Jelikož se o většinu věcí stará Expo, není potřebné, případně až na nějaké ladění, používat Android studio nebo Xcode. Je vhodný pro vývojáře chtějící rychle začít s vývojem.

Bare workflow umožňuje vývojářům přistupovat k nativním modulům a konfiguracím bez jakýchkoliv omezení a má tak v rukou naprostou kontrolu a komplexnost (složitost), která s tím přichází. Je tedy vhodný právě pro ty vývojáře, jejichž priorita je velká flexibilita a možnosti přizpůsobení prostředí.

2.2.3 Podobné technologie a porovnání

Flutter

Flutter[12] je otevřený (anglicky open-source) framework, který vyvinula společnost Google. Stejně jako React Native se používá při vyvíjení nativně kompilovaných multiplatformních aplikací pro Android, iOS a web z jednoho zdrojového kódu. Flutter je postaven na programovacím jazyku Dart[13], což je jazyk též vyvíjený společností Google.

[9] https://expo.dev
[10] https://developer.android.com/studio
[11] https://developer.apple.com/xcode
[12] https://flutter.dev
[13] https://dart.dev

Flutter oproti React Native využívá vlastní sadu UI komponent. Ke komunikaci s nativními komponentami cílové platformy se zde nepoužívá `bridge`, ale kód Dartu je přímo kompilován do nativního kódu cílové platformy. Díky tomu jsou Flutter aplikace výkonnější a většinou se blíží výkonu nativních aplikací. Jelikož se však jedná o mladší framework, jeho komunita není zdaleka tak silná jako u React Native, což může znamenat menší podporu při řešení některých netradičních problémů [4].

Xamarin

Xamarin[14] je otevřená (anglicky open-source) platforma od Microsoftu pro vytváření aplikací na Android, iOS a Windows a je založená na .NET a C#. Mezi jednotlivými platformami umožňuje vývojářům sdílet až 90 % svých aplikací. Xamarin nám nabízí k použití spoustu nástrojů a funkcionalit, z nichž zajímavý je určitě vizuální návrhář pro tvorbu uživatelského rozhraní a integrace s vývojovým prostředím Visual Studio.

Xamarin, stejně jako Flutter nabízí oproti React Native lepší výkonost. Je to způsobeno tím, že Xamarin umožňuje přímý přístup k nativnímu rozhraní API (zkratka pro Application Programming Interface). Nevýhodou pak může být cena, kdy použití zdarma platí jen pro jednotlivce či malé projekty, ale pro používání platformy Xamarin pro větší projekty je potřeba již zakoupit produkt Microsoft Visual Studio IDE. Xamarin má také menší komunitu vývojářů v porovnání s React Native [11].

Ionic

Ionic[15] je otevřený (anglicky open-source) framework pro vývoj hybridních aplikací. Tyto aplikace se vytváří pomocí webových technologií, tedy HTML, CSS a JavaScript, což může potěšit především ty vývojáře, kteří se věnovali tvorbě webů a chtějí si vyzkoušet vývoj mobilních aplikací. Aplikace se vytváří pro web a nasazení pro Android a iOS se dělá pomocí obálky (anglicky wrapper) WebView.

Výhodou Ionic je, že je možné ho integrovat s různými JavaScriptovými frameworky, jako je React, Angular nebo Vue. Jelikož uživatelské rozhraní tvoří webové technologie, bude aplikace vypadat stejně na všech zařízeních. V porovnání s React Native Ionic ale neposkytuje nativní zážitek a s tím souvisí i horší výkonost oproti nativním aplikacím [5].

2.2.4 Proč byl zvolen React Native

Vybrání určitého frameworku záleží na konkrétních požadavcích projektu a preferencích. Ke zvolení React Native došlo, protože aplikace této bakalářské práce bude sloužit jako rozšíření již existující aplikace z firmy, ve které pracuji a právě tato aplikace byla v React Native napsána. Zároveň by se dalo říct, že vytvářená aplikace není nějak významně náročná a cílená na výkonost, tedy zvolení frameworku Flutter nebo Xamarin nebylo potřeba.

2.3 Optické rozpoznávání znaků OCR

OCR neboli optické rozpoznávání znaků (anglicky Optical Character Recognition) je specifický proces, který nám umožňuje extrahovat tištěný nebo ručně psaný text z dokumentů (například fotografie nebo naskenované dokumenty) do textové podoby, která se pak může

[14] https://dotnet.microsoft.com/en-us/apps/xamarin
[15] https://ionicframework.com

dál zpracovávat. Textový výstup může být například ve formátu JSON, tak jak to dělá OCR od Microsoftu [21].

2.3.1 OCR technologie funguje prostřednictvím následujících kroků:

1. **Předzpracování obrazu** – V této první fázi se převedou dokumenty na obrázky. Provádí se to za účelem přesného strojového zobrazení a také pro odstranění nežádoucích nedostatků. Následně probíhá transformace na černobílé zobrazení, kde se cílí na zlepšení kontrastu mezi znaky a pozadím. Poté systém OCR segmentuje obraz na jednotlivé části, tedy tabulky, text a vloženou grafiku.

2. **Rozpoznávání znaků** – Pomocí umělé inteligence se analyzují jednotlivé části obrazu, tedy tmavé části oproti bílým za účelem získání znaků a číslic. Umělá inteligence k tomu může přistupovat několika způsoby, a to buď zaměřením na jedno písmeno, frázi nebo odstavec. Pro co nejlepší rozpoznání vzorů je umělá inteligence trénovaná z řady jazyků, textových formátů a rukopisů, tedy porovnává získané znaky z obrázku s již naučenými znaky. K rozpoznávání pomáhají specifické vlastnosti znaků, kam patří například počet šikmých, křížících se nebo zakřivených čar v písmenu.

3. **Následné zpracování** – V tomto posledním kroku se opravují nedostatky v rozpoznaném textu. Mezi to patří například oprava pravopisných chyb nebo správného formátování. Proto je vhodné také naučit umělou inteligenci nějaký rozsáhlý slovníček pojmů.

Jednotlivé kroky fungování OCR byly přejaty z článku „What is OCR (Optical Character Recognition): Overview, and More“ [33].

2.3.2 OCR v Azure Cognitive Services od Microsoftu

Azure Cognitive Services[16] je soubor cloudových služeb od Microsoftu umožňující programátorům jednoduše použít umělou inteligenci ve svých aplikacích. Do těchto služeb nespadá jen optické rozpoznávání znaků OCR (anglicky Optical Character Recognition), ale také rozpoznávání obličejů, rozpoznávání řeči a další.

Samotné OCR je pak součástí služeb Computer Vision (česky počítačové zpracování obrazu), kde tedy kromě extrakce textu (OCR) můžeme najít rozpoznávání obrázků (například detekování objektů), prostorovou analýzu (například pohyb lidí v prostoru v reálném čase) a rozpoznávání tváře. Přístup k těmto službám může být pomocí volání rozhraní API (zkratka pro Application Programming Interface). V této práci jsem ze služby Computer Vision verze 3.2 využil operaci *Read* a operaci *Get Read Result*. Mohl jsem také využít přímo operaci *OCR*, nicméně formát odpovědi z operace *Get Read Result* byl lepší [18].

Read

Operace *Read* funguje nad metodou *POST*. Api této operace je přizpůsobeno pro obrázky s velkým počtem textů, smíchanými jazyky a smíchané druhy dokumentů. Operace se provádí asynchronně a funguje tak, že při zavolání se mu v těle dotazu předá obrázek v binární podobě (lze také předat odkaz na obrázek online, který si pak operace sama zpracuje) a operace zpátky vrátí v hlavičce odpovědi místo, kde se náš obrázek zpracovává. K tomuto

[16] https://azure.microsoft.com/cs-cz/products/cognitive-services

místu pak přistupujeme z nadcházející operace *Get Read Result*. U operace *Read* můžeme dotaz poslat s některými parametry, ty však ale nejsou povinné. Patří sem třeba parametr language (česky jazyk – pro přímé zvolení jazyka), pages (česky stránky – používá se pro vícestránkové dokumenty) nebo readingOrder (česky pořadí čtení – pro určení pořadí, ve kterém se mají texty extrahovat).

Get Read Result

Tato operace slouží pro získání zpracovaných dat, které jsme poslali v operaci *Read*. Operace *Get Read Result* se podívá na místo, které jsme dříve získali z odpovědi od operace *Read*. Zpracovaná data z obrázku najdeme v těle odpovědi ve formátu `json`. Každé objevené slovo pak obsahuje souřadnice, které ho ohraničují, samotný text slova a úroveň spolehlivosti (jak si byla služba jistá s vyhodnocením daného slova nebo řetězce).

Mezi obě operace je pak vhodné vložit časovou pauzu, protože pokud by byly odeslány oba dotazy ve stejný čas, mohlo by dojít k situaci, kdy poslaný obrázek nebo dokument ještě není zpracován a tak nelze získat zpracovaná data. Je tedy dobré si ověřit odpověď druhé operace a v případě ještě nezpracovaného obrázku poslat dotaz znova o chvíli později.

2.4 Databáze a REST API

V této práci byla využita databáze od Microsoftu v Azure a pro komunikaci mezi databází a mobilní aplikací je využito REST API (celým názvem Representational State Transfer Application Programming Interface).

2.4.1 Databáze

Databáze je organizovaný soubor strukturovaných dat, zpravidla uložených v počítačovém systému v elektronické podobě. Správu databáze obvykle zajišťuje systém správy databází DBMS (zkratka pro Database management system). DBMS společně s daty a napojenými aplikacemi tvoří databázový systém, jež zkráceně nazýváme databáze. Obecně nám dávají možnost data spravovat, modifikovat, kontrolovat atp.

Nejběžnějším typem databáze jsou relační databáze, které ukládají data v tabulkách tvořených z řádků a sloupců, což umožňuje efektivně manipulovat s daty. Mezi relační databáze patří například MySQL, Microsoft SQL Server, Oracle Database a pro dotazování a zápis dat využívají strukturovaný jazyk SQL. Jiným typem databáze může být pak tzv. NoSQL databáze (nerelační), kam můžeme zařadit MongoDB nebo Couchbase. Tyto databáze neukládají data v tabulkách, nýbrž v nestrukturovaných nebo polo-strukturovaných formátech, kam spadají dokumenty (připomíná formát json u MongoDB) a grafy [19].

SQL databáze Azure Microsoft

Azure SQL Database[17] je relační databáze poskytující lehké napojení na další cloudové služby poskytované v rámci Microsoft Azure. Kromě automatických aktualizací a zálohování zajišťuje kvalitní zabezpečení dat, dostupnost, dynamické škálování výkonu a úložiště podle aktuálních potřeb aplikace, tedy platí se jen za to, co je potřeba. Azure SQL Database je také kompatibilní s mnoha standardními nástroji pro správu a vývoj databází, jako jsou SQL Server Management Studio nebo Visual Studio.

[17] https://azure.microsoft.com/en-us/products/azure-sql/database

2.4.2 REST API

REST API je aplikační rozhraní, které umožňuje komunikaci mezi aplikacemi na základě architektonického stylu REST. Používá standardní metody HTTP, což jsou metody GET, POST, PUT, DELETE, CONNECT a další. Poskytuje srozumitelný formát pro výměnu dat, nejčastěji ve formátu JSON nebo XML. Vývojáři tak pomocí REST API mohou jednoduše integrovat do své aplikace další služby, které by ve své aplikaci nedokázali implementovat nebo by to bylo velmi náročné a nevhodné. V aplikaci této bakalářské práce používám REST API pro komunikaci s databází a využití OCR pro analýzu textu v obrázcích.

Základy teorie pro REST API jsou převzaty z článku od společnosti Red Hat [13].

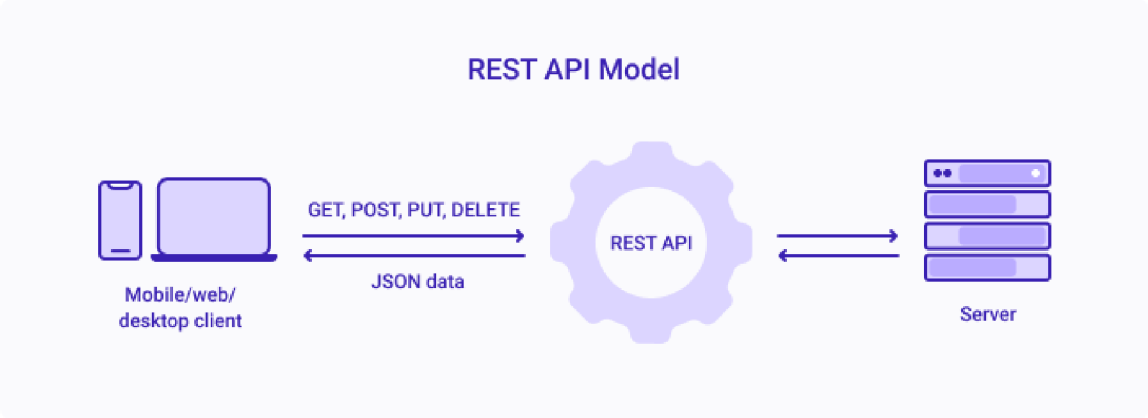
API

API (zkratka pro Application Programming Interface) je sada pravidel, nástrojů a protokolů pro vytváření a integraci aplikačního softwaru. Definuje, jakým způsobem by měly být strukturovány požadavky a odpovědi, které probíhají mezi dvěma zařízeními. Tedy pokud vývojář chce, aby aplikace komunikovala s nějakým cílovým systémem a získala od něj nějaké informace, rozhraní API mu pomůže tento požadavek předat.

REST

REST (zkratka pro Representational State Transfer) je soubor architektonických omezení používaných při vyvíjení webových služeb. Použití REST je jednoduché, protože staví na funkcích internetového protokolu HTTP (zkratka pro Hypertext Transfer Protocol) a tedy jeho konstrukce jsou velmi známé. Interakce založené na RESTu sdělují svůj stav pomocí číselných stavových kódů HTTP (200, 401, 500,...).

Na následujícím obrázku 2.2 si můžete prohlédnou model REST API při komunikaci mezi klientskou a serverovou částí.



Obrázek 2.2: Obrázek znázorňuje komunikaci mezi klientskými zařízeními (telefon, počítač) a cílovým serverem, na který jsou posílané dotazy. Obrázek je převzat ze stránek od Hevo Data [17].

Kapitola 3

Evidence majetku a závislost na technologiích

Zaměření bakalářské práce je na podporu vytvoření nového majetku v evidenci. Tato kapitola obsahuje základní informace o evidování majetku, na jaké druhy lze majetek rozdělit, inventuře a správě majetku. Na konci kapitoly jsou pak informace o čárových kódech a QR kódech a jak se dají při evidovaní majetku využít.

3.1 Evidování majetku

Evidování majetku je druh aktivity, který umožňuje jednotlivcům, malým podnikům nebo i velkým organizacím spravovat a sledovat vlastní majetek. Do tohoto důležitého procesu spadá například možnost majetek identifikovat, vyhledávat a správně distribuovat na potřebná místa (zaměstnance, místnosti a další), tedy poskytnout efektivní řízení a kontrolu majetku.

3.1.1 Rozdělení majetku

Jednotlivých druhů majetku, který lze evidovat, je opravdu velké množství a je tedy vhodné si jej rozdělit do dvou kategorií na majetek hmotný a nehmotný.

> **Hmotný majetek** jsou fyzické, neboli hmatatelné položky, jako je technické vybavení (počítače, monitory, projektory), nábytek, přepravní prostředky, nemovitosti a další. Do evidence u těchto položek můžeme například zahrnout jejich identifikaci, popis, druh, umístění či sledovat stav (zdraví položky).
>
> **Nehmotný majetek** zahrnuje takové prostředky, na které si sice nejde sáhnout, ale i přes to mohou tvořit důležitou a hodnotnou část celkového majetku jednotlivce nebo společnosti. Do těchto nehmotných zdrojů spadá software, licence, patenty, protokoly v elektronické podobě (například protokoly o předání majetku) a další. U této kategorie pak můžeme do evidence zahrnout opět identifikaci dané položky, popis a u některých i třeba platnost či datum vystavení a podpisu.

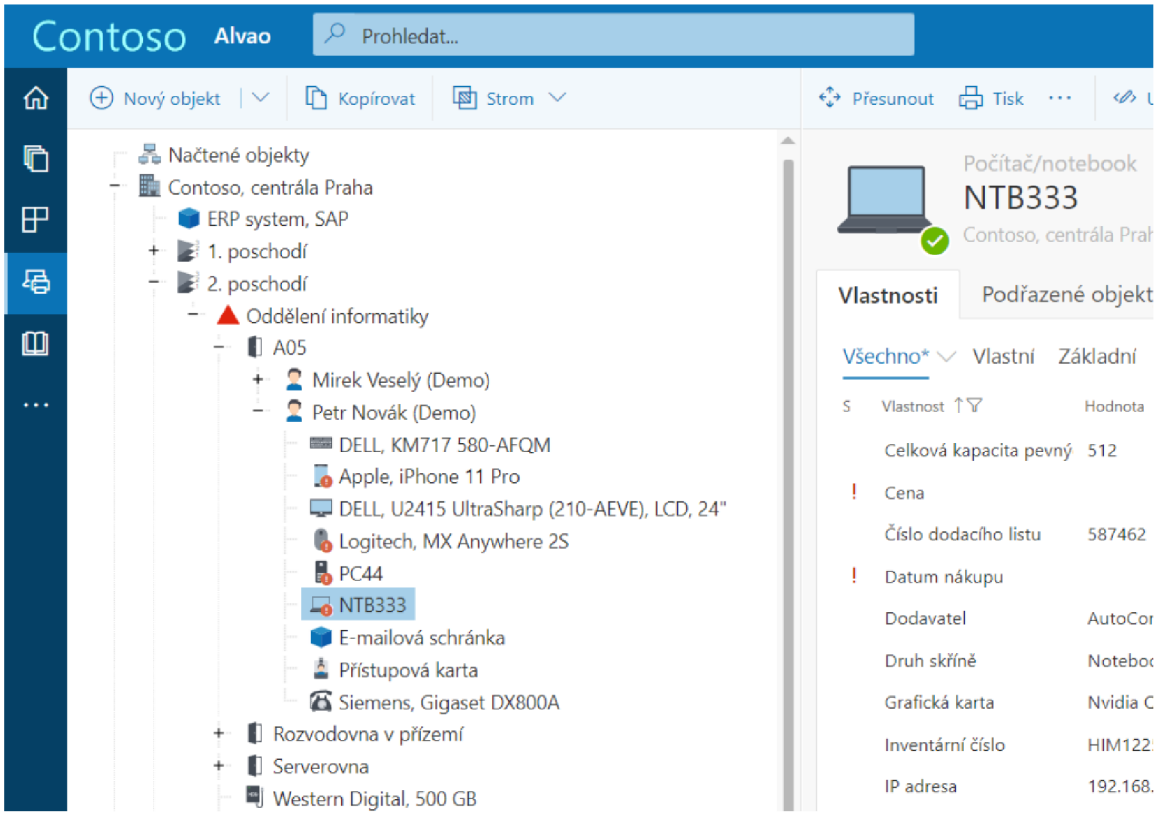
Aplikace této bakalářské práce se zabývá naskladněním nového majetku pomocí sériových čísel. Dá se tedy říct, že primárně cílí na možnost vytvoření hmotných objektů v evidenci, a to technického vybavení či nějakého nábytku.

3.1.2 Způsoby evidování majetku

Existuje více způsobů, jakými lze majetek evidovat a uchovávat o něm potřebné informace. Tím levnějším a lehce dostupným způsobem může být využití tabulkových procesorů (anglicky spreadsheets), kam patří `Excel` nebo `Google Spreadsheet`. Tento systém evidování může být dostatečný pro malé množství majetku. Pro větší počty je však nevhodný, protože není dostatečně přehledný, hůře se v něm vyhledává a spravuje majetek (majetek změní majitele a je potřeba aktualizovat evidenci). Mohou pak tedy vznikat chyby v evidenci, které pak mohou vést ke ztrátám majetku a peněz.

Druhým mnohem kvalitnějším způsobem evidence majetku je využití softwaru k tomu přímo určenému. Znatelně usnadňuje monitorování a kontrolu majetku a i když tento software je zpravidla zpoplatněn, může ve skutečnosti firmě nějaké peníze ušetřit. Také šetří čas, zabraňuje ztrátám majetku, nákupu nepotřebných duplikátů a pomáhá se vyhnout účetnickým rizikům [14].

Následující snímek poskytuje ukázku softwaru pro evidování majetku.



Obrázek 3.1: Obrázek ukazuje systém pro evidování majetku od společnosti Alvao. Obrázek byl přejat ze stránek společnosti [1].

3.1.3 Správa majetku a inventura

Správa zahrnuje několik činností, které jsou součástí životního cyklu majetku. Mezi tyto činnosti se již na začátku řadí samotné plánování nákupu a chystání rozpočtu k tomu určenému. Dále sem pak spadají opravy, údržby, následná likvidace a nahrazení majetku. V rámci správy se řeší také pojištění některých položek a případné reklamace.

Inventura je činnost, jejímž cílem je pravidelná fyzická kontrola a ověřování skutečného stavu majetku. Potvrzujeme tím například umístění položek v jednotlivých místnostech, kdo má určitou položku ve vlastnictví nebo za ni zodpovídá. Také tím lze kontrolovat aktuální zdravotní stav položek, tedy jestli nejsou nějakým způsobem poškozené. Inventura tedy pomáhá zodpovědným osobám v organizacích udržovat přehled o celkovém majetku. V organizacích se inventura zpravidla dělá jednou ročně (zapůjčené notebooky), u dražšího majetku to pak může být častěji.

3.2 Čárové kódy a QR kódy a jejich využití při evidování majetku

Čárové kódy (anglicky barcodes) a QR kódy jsou 2 druhy systémů, pomocí kterých můžeme ukládat a reprezentovat nějaká data. Dají se snadno číst pomocí fotoaparátu na mobilním zařízení. Hlavní odlišností mezi těmito systémy je, že čárové kódy umožňují ukládat data pouze v jednom rozměru, zatímco QR kódy ve dvou rozměrech. Informace pro tuto část byly čerpány z článku `What is the Difference Between a Barcode and a QR Code?` [3].

3.2.1 Čárové kódy

Čárový kód je obrázek nejčastěji obdélníkového tvaru tvořený sérií černých rovnoběžných čar různé šířky a bílých mezer. Běžný uživatel se s jejich využitím může setkat nejčastěji v obchodech, kdy se například na pokladně identifikuje zboží (zjištění ceny) naskenováním jeho čárového kódu nejčastěji ve formě nálepky.

Tak jak obchody používají čárové kódy pro identifikaci zboží, využívají je některé organizace pro identifikaci vlastního majetku. Čárový kód je nalepen na zařízení (monitor, počítač, projektor a další) a obsahuje jeho jednoznačnou identifikaci, nejčastěji tedy inventární nebo sériové číslo.

3.2.2 QR kódy

Jak již bylo zmíněno, QR kódy obsahují informace ve dvou rozměrech. Díky tomu mohou ukládat větší množství informací oproti klasickým čárovým kódům. QR kódy si mohou vygenerovat organizace i jednotlivci a přiřadit ho ke svým výrobkům či službám. S QR kódy se běžný uživatel může setkat například na reklamních plakátech, kde v kódu může být uložena cílová adresa webové stránky, na kterou reklama směruje.

QR kódy mohou obsahovat opravdu různé druhy informací, jako jsou texty, webové adresy, mailové adresy a další. Mohou být tedy použity pro uložení inventárního nebo sériového čísla konkrétního zařízení. Z toho důvodu se organizace mohou pro využití QR kódů k identifikaci zařízení rozhodnout, i přes to, že by volba čárových kódů byla zdaleka dostačující.

Kapitola 4

Analýza požadavků a konkurence

V této kapitole jsem se zaměřil na analýzu požadavků na aplikaci. K tomu bylo potřeba podívat se na uživatelské příběhy, jak se aktuálně majetek eviduje a v čem to není ideální. Potom jsem popsal, jakým způsobem k evidování majetku přistupuje konkurence a ukázal mobilní aplikace s podobnou funkcionalitou, na kterou u aplikace této práce cílíme.

4.1 Analýza požadavků na aplikaci

Klíčovým krokem pro správnou analýzu požadavků je identifikace uživatelských příběhů, díky kterým máme přehled o tom, jak uživatel postupuje v určitých situacích, a také kde a jakým způsobem by s aplikací pracoval.

Informace pro uživatelské příběhy vychází z konzultací ze zaměstnání a interních analýz společnosti Alvao[1].

4.1.1 Příběh naskladnění nového majetku a způsoby evidování

Organizace si objedná nový majetek, který po příchodu dostane na starost správce majetku. Jeho cílem je zapsat tento nový majetek do evidence. Majetek se pravidelně nakupuje hromadně a příkladem jedné takové objednávky může být 10 stejných monitorů, počítačů apod. Tento typ produktů již zpravidla obsahuje sériové číslo od výrobce, a to nejčastěji ve formátu čárových kódů nebo QR kódů.

Kromě sériového čísla se při naskladnění může pracovat s inventárním číslem. To buď přiřadí správce majetku rovnou při naskladnění nebo se o to postará až při zpracování faktury účetní oddělení. K identifikaci majetku se IT oddělení může rozhodnout používat i evidenční čísla místo inventárních čísel. Inventární číslo nebo evidenční číslo se pak v podobě štítku vytiskne a nalepí na dané zařízení.

Aktuální možnosti evidování

O novém majetku se obvykle eviduje několik údajů, jako je druh objektu, výrobce, model, sériové číslo, inventární číslo (případně evidenční číslo), dodavatel, číslo dodacího listu, datum nákupu a další. Existuje několik postupů, jak správci mohou postupovat.

1. Správce majetku postupně zapíše všechen nový majetek společně s jeho vlastnostmi do excelovského sešitu. Z něj se vytvoří soubor s daty ve formátu `csv` a následně se importuje do evidence, kde je možné s vytvořenými objekty dále pracovat.

[1] https://www.alvao.com/cs

2. V softwaru pro evidování majetku správce postupně po jednom zapisuje nové objekty. Stejné objekty případně rozkopíruje a ručně vyplní unikátní vlastnosti, jako je sériové číslo.

3. Správce majetku naskenuje sériové kódy pomocí jejich čárových kódů a nahraje je do evidence, kde z nich pak vytvoří objekty s potřebnými vlastnostmi.

Problémy těchto způsobů

- V prvních dvou případech musí správce majetku opisovat data z faktury ručně. V případě faktury v elektronické podobě může data postupně vykopírovat. Tento způsob může být velice nepohodlný a při přepisování velmi náchylný na vznik chyb.
- U třetí varianty pak pro skenování čárových kódů potřebuje správce specializovaný hardware, a tak pracuje s více zařízeními (první práce s čtečkou, potom na počítači s evidenčním softwarem).

Řešení a požadavky na aplikaci

Hledání řešení těchto problémů nám tvoří základní požadavky na mobilní aplikaci, která díky těmto nedostatkům v rámci této bakalářské práce vzniká.

- Mobilní aplikace nám umožňuje naskladnění majetku pomocí načtení jeho sériového čísla, které je na majetku ve formě čárového kódu nebo QR kódu.
- Aplikace umožňuje vybrání druhu objektu, neboli šablony, podle které se načte aktuální seznam vlastností vytvářeného objektu.
- Jednotlivé vlastnosti lze vyplnit ručně. Zároveň však je možné pro jejich vyplnění využít funkci optického rozpoznávání znaků OCR (anglicky Optical Character Recognition), kdy se pomocí kamery například naskenuje text na štítku, kde je název modelu, ten se převede na text a vloží do dané vlastnosti.
- Aplikace takto zvládne naskladnit více kusů jednoho druhu objektu, tedy pro každé naskenované sériové číslo monitoru se v evidenci vytvoří jeho odpovídající záznam s jeho odpovídajícím sériovým číslem a vlastnostmi, které mají všechny naskenované monitory stejné.

4.2 Přístup konkurence k evidování majetku a podobné aplikace

Při prozkoumávání konkurenčních řešení jsem analyzoval jak české firmy, tak i zahraniční. Obecné metody naskladnění majetku byly velice podobné tomu, tak jak to děláme u nás. Tedy importy objektů ze souboru ve formátu `csv`, ruční vytvoření objektů po jednom a občas nějaký import naskenovaných sériových čísel. Mezi českými společnostmi jsem také nenašel žádné informace o tom, že by měly vlastní mobilní aplikaci pro naskladnění majetku a jeho manipulaci.

Zde byl však znatelný rozdíl u zahraničních organizací, kde poměrně velký počet z nich měl nějakou mobilní aplikaci. Některé z těchto mobilních aplikací mimo jiné zvládali vytvoření nových objektů a manipulaci s nimi. U vytvoření majetku z aplikace to pak standardně

probíhalo postupným vyplněním informací (název, cena atd.) a přiřazením čárového kódu nebo QR kódu sériového čísla (u některých aplikací mohlo být sériové číslo nahrazeno nějakým jiným identifikačním číslem), které mohlo být naskenováno přímo mobilním zařízením nebo muselo být vyplněno ručně.

Na co jsem však při analýze zahraničních organizací nenarazil, byla možnost vytvořit pomocí mobilní aplikace více stejných objektů s různými sériovými čísly a stejnými vlastnostmi. Je ale nutné říct, že tyto produkty jsou obvykle komerční. Tedy neměl jsem kompletní přístup k jejich produktům a můj průzkum byl omezen na jejich volně dostupné informace a demo videa. Je tedy možné, že ve skutečnosti tuto funkcionalitu některé produkty mají. V následujícím úseku bude ukázka podobných aplikací.

4.2.1 Sortly: Inventory Simplified

Aplikace *Sortly: Inventory Simplified* je jedna ze zaběhlých aplikací pro správu majetku. Je vytvořena a spravována americkou společností Sortly[2] a v obchodě Google Play dosáhla více než 100 tisíc stažení. Kromě Google Play ji lze také najít v obchodě Apple App Store. Aplikace působí uživatelsky přívětivě a svým uživatelům umožňuje vytvářet nové položky v evidenci, přiřazovat k nim fotografie, kategorizovat je, nastavovat jim umístění a další. Sortly také umožňuje využívat čárové kódy a QR kódy pro sledování a úpravu položek.

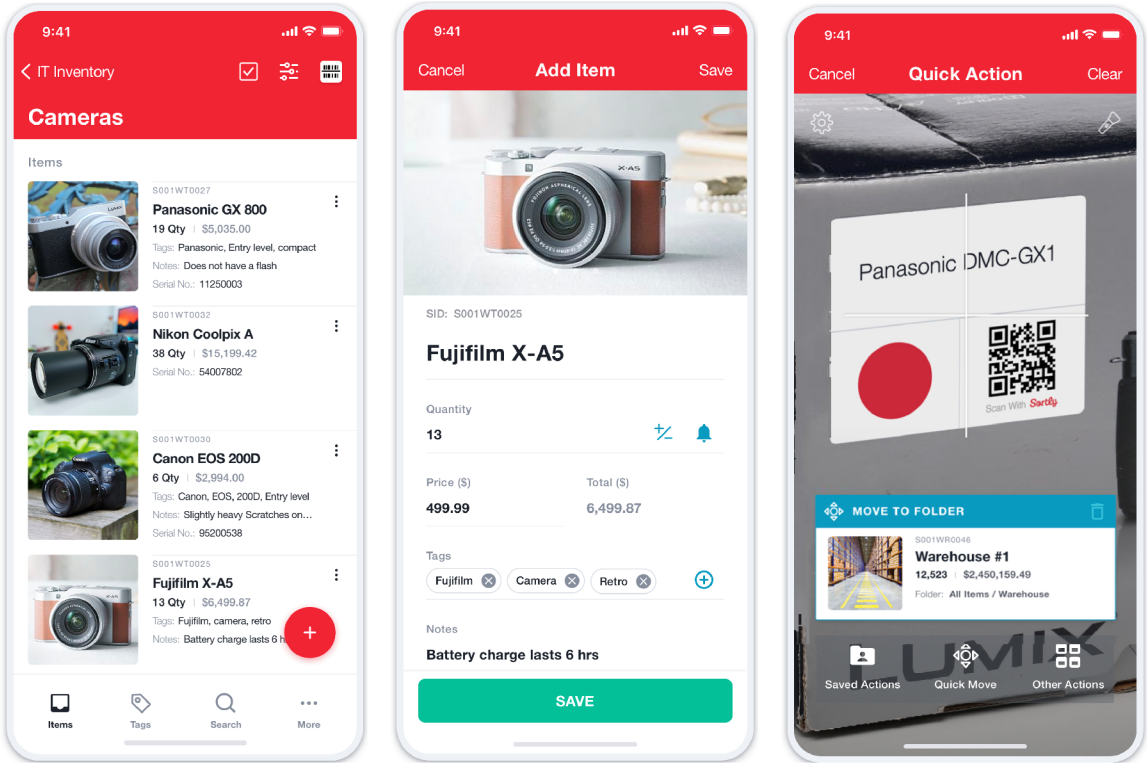
Aplikace má velice pěkně zpracované grafické uživatelské rozhraní. V seznamu produktů na obrázku 4.1a zobrazují základní informace o daném produktu, což ulehčuje práci s nimi. Jediná věc, co mi nepřijde ideální, je využití dalších dvou barev (zelené a modré) k určitě hlavní červené barvě. Tuto kombinaci lze vidět na obrázku 4.1b.

4.2.2 Itemit

Itemit[3] je britská společnost a mimo jiné i pomocí jejich mobilní aplikace poskytují systém pro evidenci a sledování majetku. Aplikace je k dispozici pro platformu iOS i Android a dává možnost uživatelům vytvářet nové položky v evidenci, pořídit a přiřadit k nim fotografii atd. Aplikace umožňuje také použití čárových kódů a QR kódů pro sledování a aktualizaci položek.

Uživatelské rozhraní aplikace je velmi pěkně zpracované a čisté. Bílé a světle šedé pozadí s tmavými texty je doplněno o červené prvky, které krásně zvýrazňují důležité části aplikace sloužící zpravidla pro nějakou akci. V čem ale aplikace trochu zaostává je poměrně malé množství informací, které lze při vytváření objektu vyplnit. Navíc pokud uživatel chce vyplnit sériové číslo, musí ho zadat ručně, a to i přes to, že aplikace podporuje čtení čárových kódů a QR kódů. Ukázku této aplikace můžete vidět na následujících obrázcích 4.2.

[2] https://www.sortly.com/
[3] https://itemit.com/



(a) Seznam produktů (b) Detail produktu (c) Skenování kódů

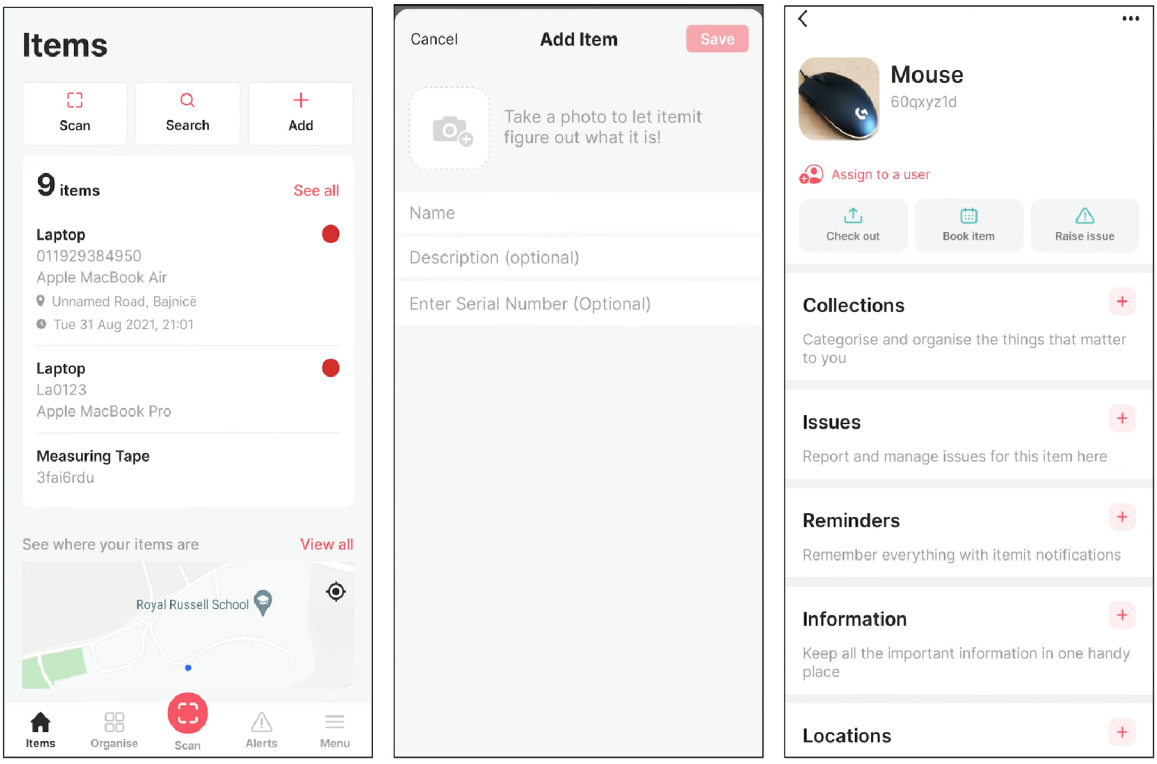
Obrázek 4.1: **Aplikace Sortly:** obrázky ukazují základní uživatelské rozhraní aplikace. Na prvním obrázku se nachází seznam produktů, kdy u každého produktu ukazují jeho fotografii a základní informace. Na druhém obrázku je ukázka detailu produktu při upravování/vytváření. Na třetím pak rychlá akce pro přesun produktů s využitím skenování čárových kódů a QR kódů. Obrázky aplikace byly přejaty ze stránky GetApp [12].

4.2.3 Asset Panda

Asset Panda[4] je další americkou společností, která poskytuje mobilní aplikaci v rámci svého systému pro evidenci a správu majetku. Díky teto mobilní aplikaci mohou uživatelé snadno přidávat nový majetek a aktualizovat jeho záznamy. Aplikace zajisté jako předchozí aplikace podporuje skenování čárových kódů a QR kódů a tak lze lehce přidávat a aktualizovat jednotlivé položky v evidenci.

Aplikace Asset Panda je poměrně uživatelsky přívětivá, rozhodně ale její kvalita grafického zpracování nedosahuje takových kvalit v porovnání s předchozími aplikacemi. Na druhou stranu mohu pochválit opravdu velké množství informací, které se dají u jednotlivých objektů vyplnit. To lze také vidět na obrázku 4.3b

[4] https://www.assetpanda.com/

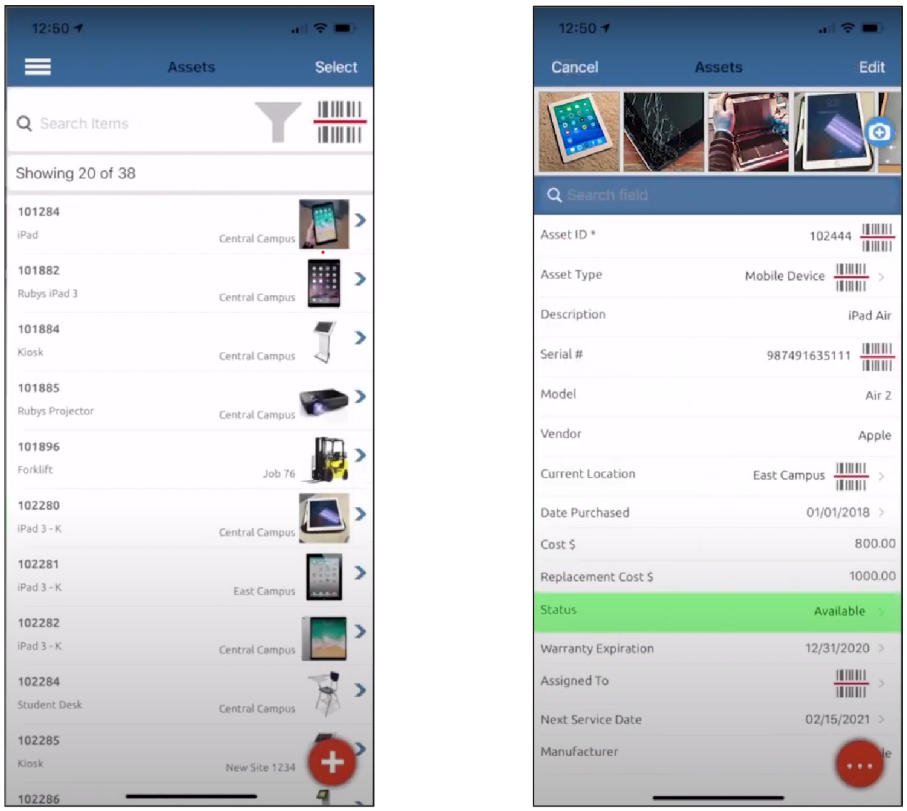


(a) Hlavní stránka (b) Vytváření položky (c) Detail položky

Obrázek 4.2: **Aplikace Itemit:** obrázky znázorňují uživatelské rozhraní aplikace Itemit. Hlavní stránku lze vidět na prvním obrázku a obsahuje akční tlačítka pro vyvolání nějaké akce (skenování, vyhledávání, vytvoření nové položky), ukázku některých položek a mapu, kde se nachází. Druhý obrázek ukazuje proces vytváření nové položky s možností přidání obrázku a vyplněním základních informací. Třetí pak ukazuje podrobný přehled jedné položky a možnost vytváření akcí souvisejících s danou položkou (například nahlášení problému). Obrázky byly přejaty z videa firmy Itemit, které demonstrovalo použití této aplikace [15]

4.2.4 Zhodnocení podobných aplikací

Ukázané aplikace byly dohromady uživatelsky přívětivé a grafické zpracování převážně u prvních dvou bylo na velmi kvalitní úrovni. Všechny aplikace uměly nějakým způsobem manipulovat s čárovými kódy a QR kódy při správě majetku v evidenci. Co však ani jedna z aplikací nemá je podpora vytváření více stejných objektů s různými sériovými čísly, tedy uživatelé tyto objekty musí vytvářet po jednom. Přidání této funkcionality by určitě posunulo tyto aplikace zase o krok dál.



(a) Seznam objektů (b) Detail objektu

Obrázek 4.3: **Aplikace Asset Panda:** obrázky opět poskytují ukázku uživatelského rozhraní aplikace. První obrázek ukazuje seznam veškerých objektů s možností vyhledávání v nich a pár akčních prvků pro skenování kódů či vytvoření nového objektu. Druhý pak ukazuje pohled na jeden objekt s jeho podrobnými informacemi. Obrázky byly přejaty z demonstračního videa od Asset Panda [20].

Kapitola 5

Návrh aplikace

Poté, co jsme si v minulé kapitole 4 představili požadavky na vytvářenou aplikaci si v této kapitole představíme návrh uživatelského rozhraní aplikace.

5.1 Rozdělení obrazovek

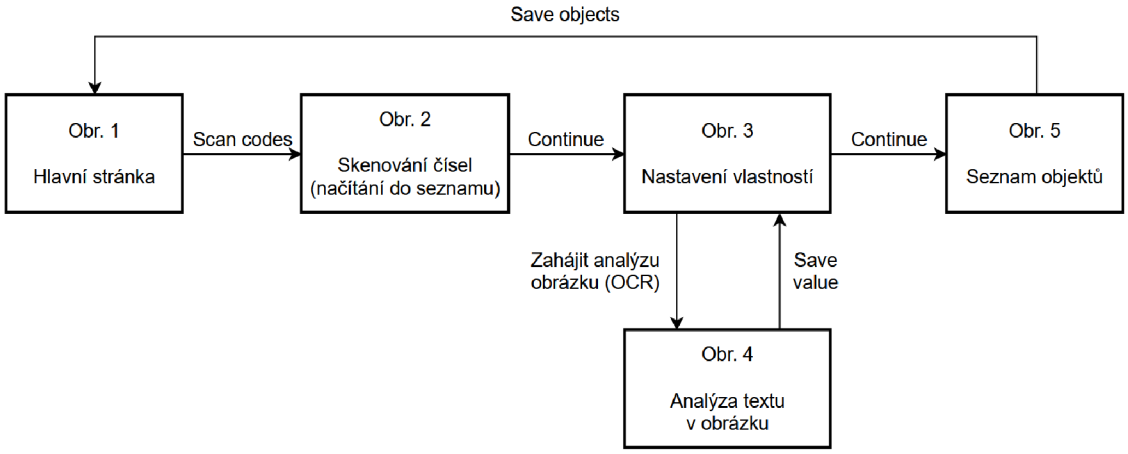
Pro vytvoření návrhu uživatelského rozhraní bylo potřeba si určit, jaké aktivity budou na jednotlivých obrazovkách aplikace probíhat. Tyto aktivity vychází ze scénáře naskladnění nového majetku. Manipulace s aplikací by v tomto procesu vypadala následovně:

1. Správce majetku otevře mobilní aplikaci, dostane se na hlavní stránku a zahájí proces pro vytváření nového majetku – obrazovka č. 1.
2. Postupně se skenují sériová čísla, zobrazují se detaily o naskenovaných číslech a mohou se naskenovaná čísla odstranit – obrazovka č. 2.
3. Vybírá se šablona objektů. Podle zvolené šablony se načtou vlastnosti vytvářených objektů, jejíchž hodnoty se dají vyplňovat – obrazovka č. 3.
4. Jednotlivé hodnoty lze vyplnit také využitím technologie OCR. Z této obrazovky bude možné vyfotit fotografii, poslat ji na analýzu rozpoznání textu a z výsledné odpovědi pak vybrat a uložit novou hodnotu do vlastnosti – obrazovka č. 4.
5. Zobrazení finálního seznamu naskenovaných objektů s vyplněnými vlastnostmi včetně zobrazení jejich unikátního sériového čísla a s možností potvrdit nové objekty a uložit je do databáze – obrazovka č. 5.

Graficky je tento proces manipulace s mobilní aplikací při naskladnění nového majetku znázorněn na obrázku 5.1.

Součástí aplikace jsou také obrazovky pro přihlášení do aplikace pomocí AAD od Microsoftu[1] a zadání adresy REST API. Tato funkcionalita již byla vytvořena v jiné aplikaci, jejímž autorem je Ing. Martin Ročkár. Ten aplikaci vytvořil v rámci jeho diplomové práce [31]. Pro potřeby této bakalářské práce byla tato funkcionalita přejata a použita z toho důvodu, že nedávalo smysl tuto část znova implementovat a navíc nesouvisí s hlavním cílem této práce.

[1] https://learn.microsoft.com/cs-cz/azure/active-directory/hybrid/connect/plan-connect-user-signin



Obrázek 5.1: Obrázek znázorňuje scénář manipulace s aplikací, kterou bude provádět správce majetku při procesu naskladnění nového majetku a uložení jeho vlastností do evidence.

5.2 Návrh uživatelského rozhraní

Návrh uživatelského rozhraní probíhal formou připomínající iterativní proces. Podle definovaných požadavků na aplikaci a rozdělení funkcionalit na jednotlivé obrazovky se vytvořil prvotní návrh. Následnými konzultacemi se zpracovaly chybné prvky a nedostatky, podle kterých se pak původní návrh přepracoval. Složitější obrazovky (například obrazovka pro skenování kódů nebo editaci pomocí technologie OCR) pak provedly tento cyklus průměrně 3x.

Jako první zde ukážu pár prvních návrhů a ukážu, v čem byly špatné a jak následné konzultace vedly k výslednému řešení. Po té popíšu finální návrh aplikace.

5.2.1 Ukázka prvních návrhů

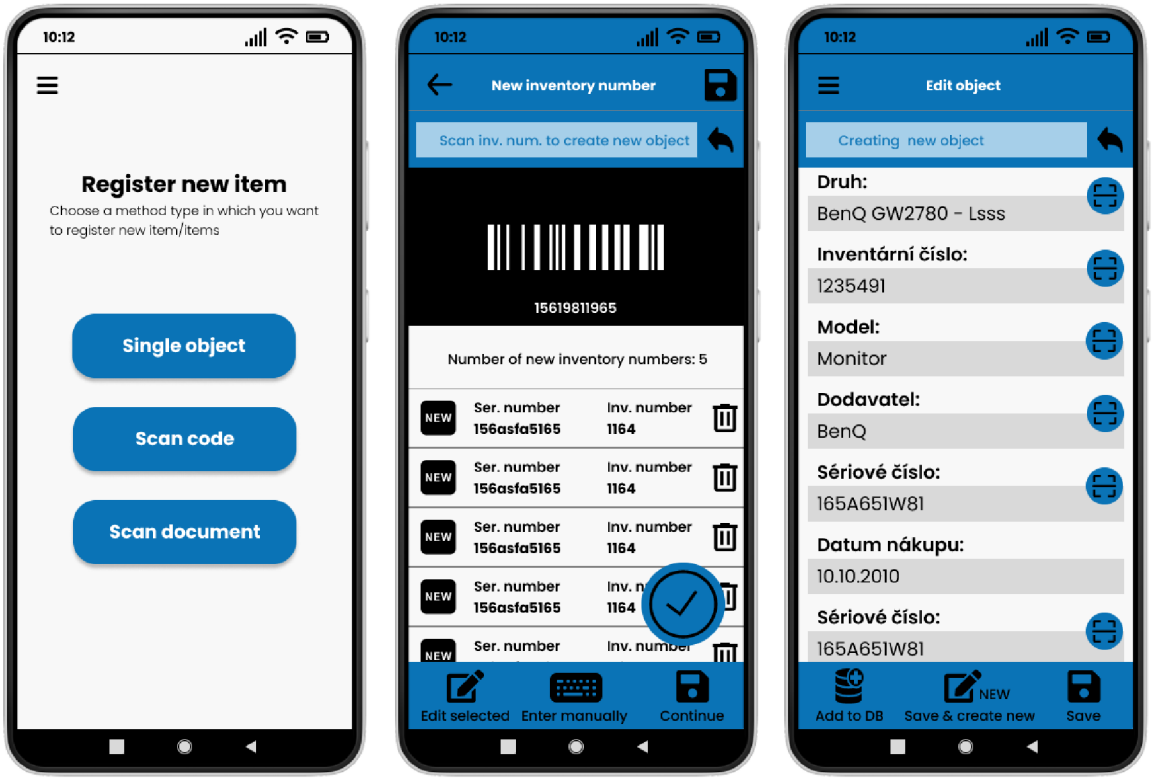
První návrhy měli dva hlavní problémy. Prvním z nich bylo, že se návrhy vytvářeli již v době, kdy ještě nebyla přesně definovaná funkcionalita cílové aplikace. Druhou pak bylo jasné nedodržení standardního rozložení uživatelského rozhraní, na které jsou uživatelé v aplikacích běžně zvyklí. Kombinací těchto dvou velkých problémů se pak první návrhy znatelně odlišují od reálných finálních návrhů.

Na hlavní stránku, kterou můžeme vidět na obrázku 5.2a, se uživatel dostane po přihlášeni do aplikace. Funguje jako takový rozcestník pro zahájení klíčových aktivit v aplikaci. Jak můžeme vidět na obrázku, v tuto chvíli se ještě pracovalo se třemi variantami aktivit. Volba vytvořit samostatně jeden objekt (v návrhu `Single object`) však mohla být provedena v rámci našeho základního scénáře použití aplikace definovaného v sekci *Rozdělení na stránky* 5.1. V návrhu ji šlo tedy sjednotit do jedné aktivity pod názvem `Scan code`. Aplikace taky měla obsahovat možnost skenování faktury nebo dodacího listu, nicméně z toho rychle sešlo, protože se jedná o velice náročné řešení na implementaci (mohlo by to dle mého názoru být jako samostatné téma pro bakalářskou nebo diplomovou práci).

V momentě zahájeni procesu naskladnění kliknutím na `Scan code` na hlavní stránce by se uživatel dostal na stránku skenování čárových kódů a QR kódů. Tato obrazovka, kterou můžete vidět na obrázku 5.2b, je asi nejhorším návrhem této bakalářské práce. Hlavička na-

vigace na vrchu obrazovky ikonu uložení pro pokračování na další obrazovku, což absolutně nedává smysl. K tomu se ještě v navigaci objevovaly dvě různé barvy na jednobarevném pozadí. Sekce pod tím tam absolutně nedává smysl a sám nevím, jak jsem něco takového mohl vůbec dát ke kontrole. Okno fotoaparátu pro skenování netvoří ani třetinu obrazovky a hned pod ním se nachází seznam naskenovaných objektů. V této části návrhu se ještě pracovalo s variantou skenování jak sériového čísla, tak i inventárního čísla. Od toho se již v pozdějších návrzích odstoupilo. Spodní menu je perfektní ukázkou nedodržení pravidel uživatelského rozhraní. Nefunguje zde totiž pro přesun mezi klíčovými částmi aplikace, ale jako menu s akčními prvky pro vyvolání nějaké aktivity. Mělo poskytovat funkcionalitu pro zadání čísla ručně, úpravu zvoleného čísla a pokračování na další obrazovku, což bylo navíc hloupě skryto pod ikonkou uložení. Fajfka nad spodním menu pak měla fungovat jako tlačítko aktivující fotoaparát pro sejmutí kódu z obrazovky.

Třetí obrazovka sloužící pro úpravu vlastností objektů na obrázku 5.2c pak opakuje podobné chyby jako obrazovka skenování kódů. Navigace v horní části obsahuje špatnou ikonku pro přesun zpátky a pod navigací se opět vyskytuje nesmyslná část. Spodní menu pak opět chybně obsahuje akční prvky pro danou stránku místo pro přesun mezi důležitými částmi aplikace.



(a) Hlavní obrazovka (b) Skenování kódů (c) Editace objektů

Obrázek 5.2: První obrázek z této trojice ukazuje prvotní návrh hlavní stránky vytvářené aplikace. Tento návrh pracoval se třemi možnými scénáři naskladnění nového majetku. Na druhém obrázku je návrh obrazovky pro skenování kódů a na třetím pak návrh pro úpravu vlastností nově vytvářených objektů. Oba tyto návrhy nedodržely základní strukturu uživatelského rozhraní, na kterou jsou uživatelé zvyklí.

Ponaučení z prvních návrhů

Hlavním ponaučením pro následnou úpravu návrhů bylo dodržení určité standardní struktury uživatelského rozhraní, na kterou jsou uživatelé u běžných aplikací zvyklí. Po vyvarování těchto chyb byly nové návrhy mnohonásobně lepší a další úpravy, které následovaly, již byly poměrně dobré.

5.2.2 Finální návrhy aplikace

Obrazovka 1 – Hlavní stránka

Po postupných úpravách nám na hlavní stránce, kterou můžeme vidět na obrázku 5.3a, zůstalo akční tlačítko `Scan code`, které nám spustí proces vytváření nového majetku. Navíc se zde objevilo vyhledávací pole, které by mělo sloužit k vyhledávání již vytvořených objektů v databázi (tato funkcionalita však není cílem této práce a nebude implementována). Na pravé straně vyhledávacího pole je ikonka pro skenování. Kliknutí na ni by mělo otevřít fotoaparát pro skenování čárových kódů a QR kódů. Například naskenované sériové číslo by pak mělo vyhledat objekt s odpovídajícím číslem v evidenci. Kliknutí na ikonku tří vodorovných čárek v levém horním rohu by pak mělo otevřít boční menu, tak jak jde vidět na třetím obrázku 5.3b.

Boční menu obsahuje malou ikonu aplikace firmy Alvao[2] a umožňuje přesun mezi některými částmi aplikace, jako je nastavení nebo profil. Pomocí `Sign out` se pak dá z aplikace odhlásit.

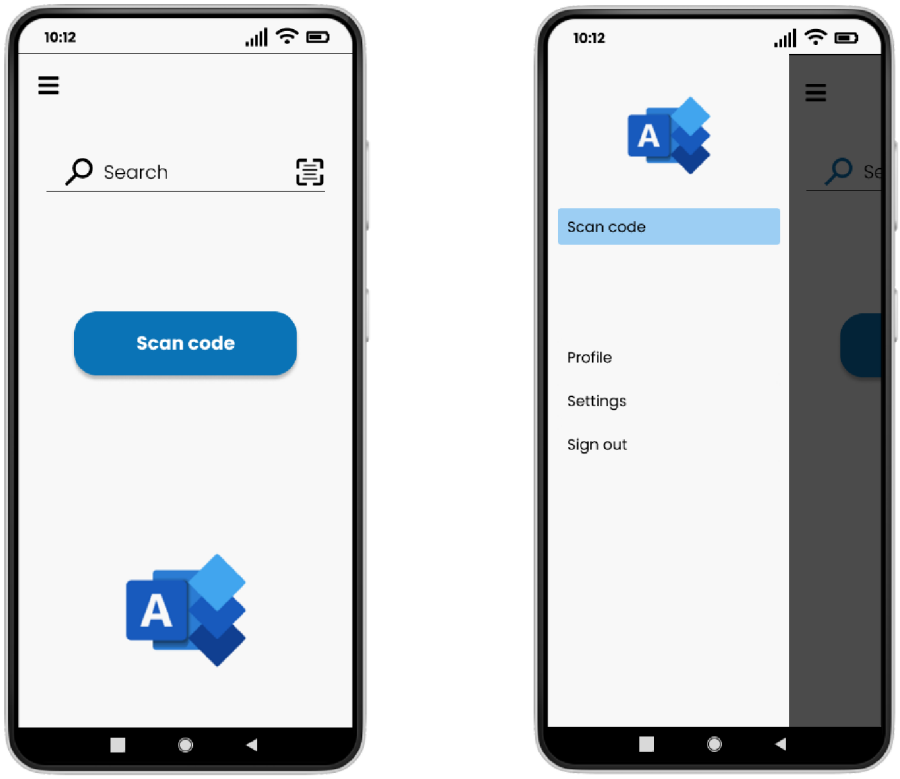
Obrazovka 2 – Skenování čárových kódů a QR kódů

Obrázky 5.4 ukazují finální verzi návrhu obrazovky pro skenování sériových čísel pomocí čárových kódů a QR kódů. V navigaci na horní části obrazovky zůstal jen nadpis a šipka zpět pro přechod na předchozí obrazovku. Zároveň byly také dodrženy barvy těchto elementů. Fotoaparát pro skenování čísel je nyní přes většinu obrazovky. Díky tomu má uživatel větší přehled o tom, co vidí a skenuje. Kolečko s číslem v pravém horním rohu fotoaparátu ukazuje aktuální počet naskenovaných čísel. Po kliknutí na toto kolečko se otevře modální okno s naskenovanými čísly a možností jednotlivá čísla odstranit, což lze vidět na druhém obrázku 5.4b. Pro kvalitnější zobrazení modálního okna je v době jeho otevření ztmavené pozadí. Tlačítkem `Close` jde pak modální okno zavřít. Po ukončení skenování čísel lze na další stránku přejít pomocí tlačítka `Continue`.

Obrazovka 3 a 4 – Úprava vlastností nově vytvářených objektů

Pro naskenovaná sériová čísla, například 10 stejných monitorů, musím jako první určit druh objektu, který nám definuje, jaké vlastnosti bude daný objekt mít. Proto jako první na obrazovce editace objektů bude pole se seznamem šablon určujících druh objektů. Podle zvolené šablony druhu objektu se pak zobrazí odpovídající vlastnosti pro vytvářený objekt. Tyto vlastnosti je pak možné vyplnit, což lze vidět na obrázku 5.5a. Pro vyplnění jednotlivých vlastností je pak možné využít analýzu textů z obrázků pomocí technologie OCR (anglicky Optical Character Recognition). Na obrazovku s analýzou obrázků se dostaneme kliknutím na ikonku skenování na pravé straně u jednotlivých vlastností. Navigace v horní části obrazovky pak opět obsahuje jen nadpis a šipku zpět pro návrat na předchozí stránku. Pomocí tlačítka `Continue` pak můžeme přejít stránku s finálním seznamem vytvářených objektů.

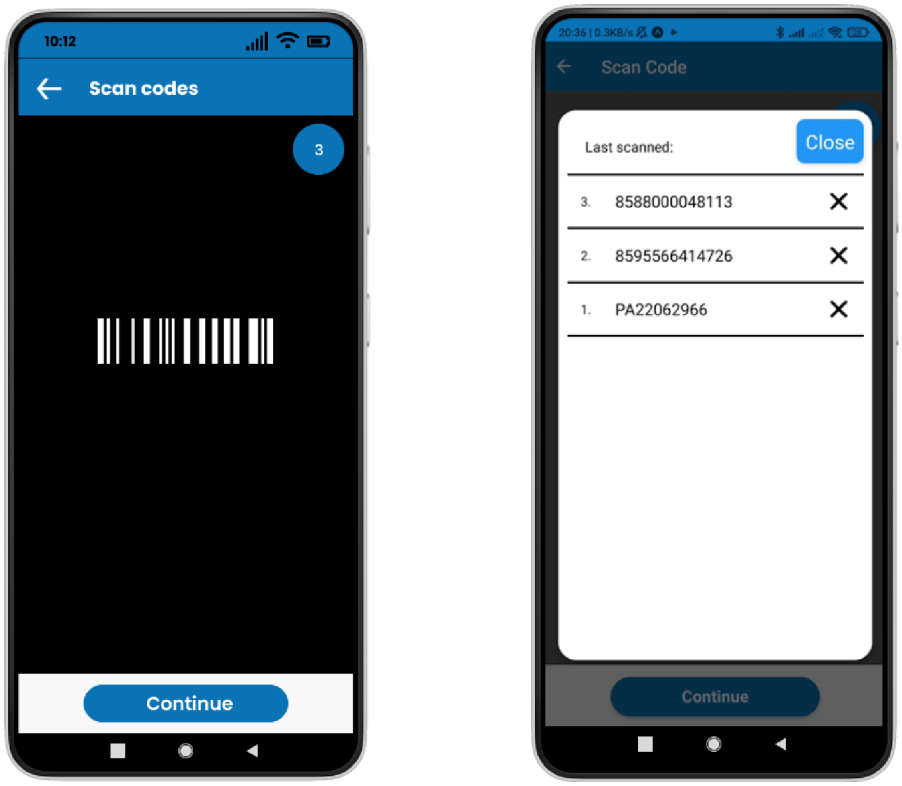
[2] https://www.alvao.com/cs/



(a) Hlavní stránka (b) Boční menu

Obrázek 5.3: Na prvním obrázku je ukázka hlavní obrazovky po přihlášení do aplikace. Obsahuje velké tlačítko `Scan code` pro spuštění procesu naskladnění majetku. Obsahuje také vyhledávací pole se skenováním kódů pro vyhledávání odpovídajících objektů v evidenci. Druhý obrázek ukazuje návrh bočního menu, které se zobrazí po kliknutí na ikonku menu na hlavní stránce. Zde se dá například z aplikace odhlásit.

Po přechodu na obrazovku pro úpravu vlastností pomocí analýzy textu v obrázku se zobrazí fotoaparát. Po vyfocení fotografie se provede analýza textu a jednotlivá slova se na fotografii označí a je možné je vybírat. Návrh zvýraznění textu a jeho vybírání si můžete prohlédnout na obrázku 5.5b. Vybírání vlastností bylo navrženo způsobem, že by si uživatel mohl ze seznamu nad analyzovaným obrázkem vybrat vlastnost, jejíž hodnotu chce vyplnit. V době návrhu dávalo smysl pojmenovat tuto obrazovku jako `Přiřazení vlastností` (v obrázku `Assign properties`). Lepší by však byl název naznačující analýzu obrázku, tedy `OCR Analysis` nebo `Image Analysis`. Tlačítko `Continue` zde pak mělo fungovat jako návrat na obrazovku Editace objektů. To však aktuálně neshledávám jako ideální.

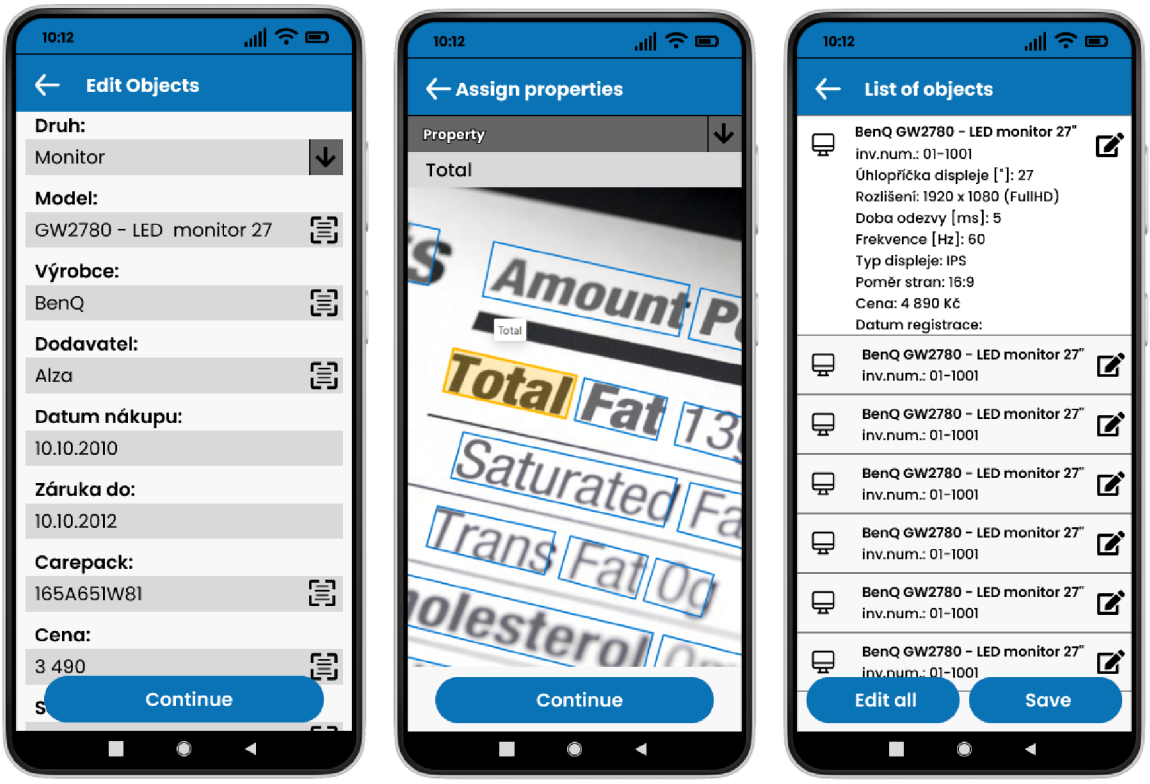


(a) Skenování kódů (b) Modální okno

Obrázek 5.4: Tyto obrázky znázorňují finální verzi návrhu obrazovky pro skenování čárových kódů a QR kódů. Jedná se o poměrně čisté řešení. Fotoaparát pro skenování čísel je přes celou obrazovku a naskenovaná čísla je možné si zobrazit v modálním okně, které se objeví po kliknutí na modré kolečko s číslem v pravém horním rohu fotoaparátu. Číslo v kolečku ukazuje aktuální počet naskenovaných čísel. Tlačítko `Continue` slouží pro přesun na další obrazovku procesu naskladnění majetku.

Obrazovka 5 – Seznam naskenovaných objektů

Závěrečnou obrazovkou procesu naskladnění nového majetku je obrazovka s finálním seznamem naskenovaných objektů. Na ukázku návrhu této obrazovky se můžete podívat na obrázku 5.5c. Počet zobrazených objektů se rovná počtu naskenovaných sériových čísel. Na objekt v seznamu lze také kliknout, čímž se zobrazí detailní přehled všech vlastností daného objektu. Ve variantě tohoto návrhu lze vidět možnost spustit editaci jednoho objektu kliknutím na ikonku znázorňující úpravu, která je na pravé straně každého objektu. Každý objekt v seznamu pak má ikonku na levé straně, která odpovídá druhu objektu (monitor, projektor, počítač atd.). Navigace klasicky umožňuje přechod na předchozí obrazovku a obsahuje nadpis. Dole na obrazovce seznamu objektů se pak nachází dvě tlačítka. Tlačítko `Edit all` by uživatele přesunulo zpátky na editaci vlastností všech objektů. Je zde však poměrně zbytečné, protože stejnou akci můžeme udělat šipkou zpět v navigaci. Druhé tlačítko `Save` pak uloží objekty s jejich vlastnostmi do databáze.



(a) Úprava vlastností (b) Skenování kódů (c) Editace objektů

Obrázek 5.5: První obrázek ukazuje hlavní obrazovku pro úpravu vlastností objektů. Vlastnosti, které je potřeba vyplnit se zobrazí po zvolení druhu, neboli šablony objektu. Kliknutím na ikonku skenování u jednotlivých vlastností pak otevřeme obrazovku s fotoaparátem. Po vyfocení fotografie se provede analýza textu a jednotlivé texty se označí. To je znázorněno na druhém obrázku. Po přiřazení hodnot k vlastnostem se můžeme vrátit na předchozí stránku pomocí tlačítka `Continue`. Třetí obrázek pak ukazuje obrazovku se všemi naskenovanými objekty, kde počet se odvíjí od počtu naskenovaných sériových čísel. Z této obrazovky je možné opět spustit editaci všech objektů tlačítkem `Edit all` nebo jen jednoho pomocí ikonky úpravy na pravé straně u každého objektu. Použitím tlačítka `Save` se pak objekty uloží do databáze.

Kapitola 6

Implementace a testování aplikace

6.1 Implementace

Aplikace byla vytvořena s pomocí frameworku React Native s využitím platformy Expo, což je nástavba okolo React Native Frameworku. Základní popis tohoto frameworku a platformy Expo se nachází v sekci 2.2.

6.1.1 Struktura aplikace

Celkovou strukturu aplikace netvoří jen část implementovaná v mobilní aplikaci v React Native. Důležitou součástí celkové aplikace je také databáze. Ta je umístěna v cloudovém prostředí Azure od Microsoftu. Komunikace mezi mobilní aplikací a databází je realizována pomocí REST API (celým názvem Representational State Transfer Application Programming Interface). Základní informace k REST API a databázi jsou uvedeny v sekci 2.4.

Mobilní aplikace

Mobilní aplikace je složena z komponent. Hlavním souborem aplikace je soubor `App.js`. Tento soubor je většinou součástí každého nového Expo projektu a je to zpravidla první soubor, ve kterém programátoři začínají pracovat. V aplikaci byl tento soubor použit pro vytvoření navigace, která nám umožňuje přesun mezi obrazovkami.

Dále aplikace v základu obsahuje například konfigurační soubory `app.json` a také `babel.config.js`, soubor `package.json` s informacemi o balíčcích, složku `node_modules` obsahující knihovny používané v projektu a složku `assets` pro ukládání obrázků a ikon.

V projektu je samozřejmě možné pro aplikaci vytvářet nové komponenty. Vytvořil jsem pro to složku `screens`, která obsahuje komponenty implementující funkcionalitu jednotlivých obrazovek. Vytvořil jsem 5 komponent pro obrazovky, které reflektují rozdělení aplikace tak, jak je uvedeno v návrhu aplikace v části 5.1. Součástí této složky jsou i další komponenty. Ty však byly přejaty z mobilní aplikace diplomové práce od Ing. Martina Ročkára [31] a slouží hlavně k přihlášení do aplikace a zadání adresy REST API. Z této aplikace byly přejaty také struktury pro posílání dotazů na databázi. Ty jsou obsaženy ve složce `helpers` spolu s dalšími pomocnými soubory ještě pro výše zmíněné přihlášení do aplikace a zadání adresy REST API.

Databáze

V této práci používám Azure SQL Server od Microsoftu. Novou databázi jsem v rámci této práce nemusel vytvářet. Použita byla jedna z testovacích databází, kterou jsem si získal v rámci zaměstnání. Jedná se opravdu o robustní databázi a její strukturu si lze prohlédnout v dokumentaci [2] produktu společnosti Alvao.

REST API

REST API je použito mobilní aplikací pro komunikaci s databází. Funkcionalita potřebná pro mnou vytvářenou mobilní aplikaci již však byla implementována v rámci firemních produktů. REST API bylo prostřednictvím Visual Studia[1] nasazeno jako webová aplikace v prostředí Azure od Microsoftu. Implementace REST API je provedena v .NET Frameworku 4.8[2] a je navržena podle OData standardu[3].

6.1.2 Navigace v aplikaci

Navigace je klíčovým prvkem pro orientaci v aplikaci a pro přesun mezi jednotlivými obrazovkami. V řešení byla navigace implementována pomocí knihovny `React Navigation`[4], která nám poskytuje různé typy navigace. Od typu navigace se pak odvíjí její způsob použití. V aplikaci je použitý tzv. `Stack Navigator` a `Drawer Navigator`.

Pro použití navigace je potřeba ji první do projektu nainstalovat. K instalaci můžeme použít `npx expo install react-native-screens react-native-safe-area-context` ve složce projektu. Výhodou Expo řízeným projektem je, že pro instalaci knihoven můžeme použít `npx expo`. To se nám postará o instalaci kompatibilních verzí knihoven. Kód aplikace pak potřebujeme obalit navigačním kontejnerem, který v kódu musíme importovat následujícím způsobem `import { NavigationContainer } from '@react-navigation/native'`

Stack Navigator

Stack Navigator [29] je nejběžnějším typem navigace z knihovny React Navigation. Umožňuje aplikaci přecházet mezi různými obrazovkami způsobem, že nová obrazovka je umístěna na vrchol zásobníku. Pokud se vracíme z nějaké obrazovky zpět na předchozí, je to jako kdybychom opouštěnou obrazovku ze zásobníku vysunuli.

Pro použití Stack Navigatoru je potřeba ho první nainstalovat a poté v kódu naimportovat `import { createStackNavigator } from '@react-navigation/stack'`. V rámci aplikace můžeme pracovat s více navigátory a tedy k odlišení používám instanci, ve které jsme je vytvořili. Instance je vytvořena příkazem `const Stack = createStackNavigator()`.

Použití navigace v aplikaci reflektuje ukázka 6.1. `Stack.Navigator` je komponenta definující daný navigátor. Komponenta `Stack.Screen` definuje jednotlivé obrazovky v navigátoru. Nastavením parametru `component` určujeme cílovou obrazovku (vytvořené obrazovky jsou také komponenty). V `options` pak můžeme definovat například nadpis, který se objeví v hlavičce obrazovky a případně další nastavení.

[1] https://visualstudio.microsoft.com/cs/
[2] https://dotnet.microsoft.com/en-us/download/dotnet-framework/net48
[3] https://www.odata.org/
[4] https://reactnavigation.org/docs/getting-started

```
1   import { createStackNavigator } from '@react-navigation/stack';
2
3  <Stack.Navigator>
4    <Stack.Screen options={{ ... }} name="Main" component={Home} />
5    <Stack.Screen options={{ ... }} name="ScanCode" component={ScanCode} />
6    ...
7    <Stack.Screen options={{ ... }} name="Ocr" component={Ocr} />
8  </Stack.Navigator>
```

Výpis 6.1: Ukázka znázorňuje použití Stack Navigatoru v aplikaci. Komponenta `Stack.Navigator` definuje navigátor a `Stack.Screen` pak definuje jednotlivé obrazovky navigace

Drawer Navigator

Drawer Navigator [28] je typ navigačního menu, které můžeme vysunout a zasunout z boční strany obrazovky pomocí gest. Na vysunutí je na hlavní stránce v aplikaci implementována také ikonka. Pro přidání do projektu je potřeba ho nainstalovat a následně v kódu naimportovat pomocí `import { createDrawerNavigator } from '@react-navigation/drawer'`.

V kódu aplikace je Drawer Navigator definován v sekci `return` hlavní funkce `MyApp`. Svoje stylové vlastnosti a některé chování ale přebírá z komponenty DrawerContent.

Základní zobrazení

To co uživatel vidí po spuštění aplikace se odvíjí od toho, zda je nebo není přihlášen a také zda má nastavenou URL adresu REST API. Pokud uživatel není přihlášen, zobrazí se mu stránka s přihlášením. Přihlášení v aplikaci probíhá prostřednictvím AAD ověřování (celým názvem anglicky Azure Active Directory Authentication). Po přihlášení se uživatel dostane na hlavní stránku, ze které může zahájit skenování sériových čísel pro naskladnění nového majetku. Pokud by uživatel neměl nastavenou adresu REST API, zobrazí se mu stránka, na které ji může vyplnit.

Základní kostra aplikace (přihlášení, většina navigace, kostra dotazů na databázi) byla přejata z již hotové mobilní aplikace od Ing. Martina Ročkára [31].

6.1.3 Skenování sériových čísel

Skenování sériových čísel je první aktivitou procesu naskladnění nového majetku. Obrazovka (komponenta) s touto funkcionalitou je naprogramována v souboru `scanCode.js` a uživatel se na ni dostane z hlavní stránky po kliknutí na tlačítko `Scan codes`. O skenování čísel se stará modul `Expo BarCodeScanner`, který podporuje jak čárové kódy i QR kódy.

Expo BarCodeScanner

Expo BarCodeScanner [6] jsem si vybral hlavně z toho důvodu, že se jedná přímo o Expo knihovnu a při její instalaci do projektu pomocí `npx expo install expo-barcode-scanner` je zajištěna kompatibilita této knihovny s aktuálním projektem. Také tato knihovna podporuje opravdu velký počet kódových formátů. Pro použití v kódu je pak samozřejmě nutné ji takto importovat `import { BarCodeScanner } from 'expo-barcode-scanner'`.

Jako první aplikace ověří, zda má povolen přístup k fotoaparátu zařízení. Pokud přístup není povolen, požádá o něj. Pokud je oprávnění uděleno, uživatel může skenovat. Po přechodu na tuto obrazovku je skenování hned aktivní až do doby, než se naskenuje první kód. Po naskenování se totiž změní stav stavové proměnné `scanned` na true. Zahájit opět skenování lze kulatým tlačítkem s nápisem *Scan*, které se aktivně objevuje na obrazovce podle stavu skenování.

Naskenovaný kód se uloží do seznamu `scannedCodes`, který byl deklarovaný pomocí stavové proměnné tímto způsobem `const [scannedCodes, setScannedCodes] = useState([])`. Při ukládání kódu do seznamu se navíc ověří, zda již daný kód nebyl naskenován. Pokud byl, tak se kód znovu neuloží a skenování dál pokračuje.

Následující první kód 6.2 ukazuje klíčovou komponentu `BarCodeScanner` pro skenování. Pokud dojde úspěšně k naskenování kódu, zavolá se funkce `handleBarCodeScanned`, jejíž demonstraci lze vidět ve druhém kódu 6.3. Jejím cílem je ověřit, že kód již nebyl naskenován, pozastavit skenování dalších kódů, uložení naskenovaného kódu do seznamu a zavoláním funkce pro aktualizaci počtu naskenovaných kódů.

```
1   <BarCodeScanner
2       onBarCodeScanned={scanned ? undefined : handleBarCodeScanned}
3       style={styles.scannerSize}
4   />
```

Výpis 6.2: Tento kód znázorňuje komponentu `BarCodeScanner`, která skenuje kódy a po úspešném naskenování, pokud je parametr `scanned` false, se zavolá funkce handleBarCodeScanned

```
1   const handleBarCodeScanned = ({ type, data }) => {
2       if (!scannedCodes.includes(data)) {
3           setScanned(true);
4           setScannedCodes([data, ...scannedCodes]);
5           countScannedCodes();
6       }
7   };
```

Výpis 6.3: Funkce `handleBarCodeScanned` první projde celý seznam již naskenovaných kódů a ověří, že se nově naskenovaný kód v seznamu ještě nevyskytuje. Pokud v seznamu ještě není, přeruší skenování, uloží nový kód a zavolá aktualizaci počtu naskenovaných kódů.

Zobrazení čísel z naskenovaných kódů

V aplikaci je na obrazovce *Scan Code* také možné zobrazit si seznam naskenovaných kódů. K tomu jsem použil komponentu `Modal`, díky které lze prezentovat nějaká data nad aktuálně zobrazeným obsahem. Toto modální okno kromě naskenovaných kódů ukazuje pořadí, v jakém byly naskenovány a je možné zde jednotlivé kódy ze seznamu odstranit. Aby se kód dal odstranit, je k tomu potřeba znát jeho index. Index jednotlivým kódům přiřazuje funkce `map()` při vykreslování.

Odstranění potom provede funkce `deleteCode()`, která si první zkopíruje celý seznam do pomocné proměnné. Z ní smaže kód podle cílového indexu a seznam zpátky uloží do původní stavové proměnné. Tento proces je ukázán v kódu 6.4.

```
1    const deleteCode = (index) => {
2        let codesCopy = [...taskItems];
3        codesCopy.splice(index, 1);
4        setScannedCodes(codesCopy);
5        countScannedCodesAfterDelete();
6    }
```

Výpis 6.4: Funkce `deleteCode` odstraní kód podle cílového indexu v pomocné proměnné a nový seznam nahraje zpět do stavové proměnné.

6.1.4 Vlastnosti objektů

Nastavení vlastností při vytváření nových objektů je jednou z hlavních funkcí této aplikace. Vlastnosti objektů se odvíjí od typu vytvářených objektů. Například jednou z vlastností objektu typu monitor může být délka diagonály. Tuto vlastnost ale nebude mít objekt typu stolní počítač.

Typy jednotlivých objektů jsou uloženy v databázi společně s jejich specifickými vlastnostmi. Postup pro zobrazení vlastností objektů na obrazovce `Set Object Properties` je následující.

1. Po přechodu na tuto obrazovku se pošle z mobilní aplikace dotaz na databázi pro získání všech typů objektů v seznamu šablon.
2. Uživatel si ze seznamu vybere cílový typ objektu
3. Po výběru typu objektu se pošle druhý dotaz pro získání vlastností odpovídajících danému typu objektu.
4. Jakmile jsou data z databáze získána, zobrazí se vlastnosti na obrazovce a je možné je vyplňovat.

Kostra kódů pro dotazování na data v databázi byla přejata z aplikace diplomové práce od Ing. Martina Ročkára [31]. Mým cílem zde tedy bylo správně připravit a manipulovat s daty.

Odpovědí druhého dotazu je pole objektů definující jednotlivé vlastnosti. Z jednotlivých objektů v tomto poli si vytáhnu některé informace o vlastnostech (id, jméno, hodnota) a uložím si je do nového pole. Z pole pak odfiltruji vlastnosti sériové a inventární číslo. Inventární číslo se při naskladnění v aktuálním scénáři nepoužívá a se sériovým číslem pracujeme bokem. Ukázku vybrání specifických vlastností a odfiltrování inventárních a sériových čísel můžete vidět v ukázce 6.5. Jak vypadají příchozí data z druhého dotazu od serveru pak v kódu 6.6.

```
1    setProperties(() => {
2        return res.value.map((item) => ({ id: item.id, name: item.name,
             value: null }))
3    });
4    setProperties((current) => current.filter((item) =>
5    item.name !== 'Inventory number' && item.name !== 'Serial number'));
```

Výpis 6.5: Uložení základních informací vlastností a vyfiltrování sériového a inventárního čísla

```
Array [
  Object {
    "_links": Object {
      "parent": Object {
        "href": "objects/407",
      },
    },
    "id": 2832,
    "name": "Monitor type",
    "nameOrder": 3,
    "value": null,
  },
  Object {
    ...
]
```

Výpis 6.6: Ukázká přichozích dat po dotazu na získání vlastností podle zvoleného typu objektu.

Editace v textových polích

Jednotlivé vlastnosti jsou jako objekty uloženy v poli `properties`. V seznamu zobrazených vlastností lze hodnoty přímo vyplnit v komponentách `TextInput`, kdy po upravení nějaké hodnoty se zavolá funkce `setObjectProperty`. Ta vytváří nové pole postupným kopírováním objektů z pole `properties` až do chvíle, než nenarazí na nově upravenou vlastnost. Pro danou vlastnost zaktualizuje hodnotu a pokračuje s ukládáním do nového pole. Výsledným polem pak nahradí původní pole v proměnné `properties`.

Editace s využitím OCR

V aplikaci jde jednotlivé hodnoty vlastností vyplnit i s využitím technologie OCR. Funkcionalita je implementována na obrazovce s názvem `Image Analysis` v komponentě `Ocr.js`. Tuto variantu zde popíšu trochu více.

1. Po přechodu na obrazovku se uživateli otevře fotoaparát přes celý displej. Namíří fotoaparát na text, který chce analyzovat a pomocí tlačítka `Scan` sejme fotografii.
2. Aplikace převede fotografii na binární data a odešle je na Computer Vision API s žádostí o operaci `Read`. Odpovědí této služby bude mimo jiné adresa místa, na kterém najdeme výslednou analýzu obrázků.
3. Po počkání tří sekund se odešle nový dotaz GET pro operaci `Get Read Result` na adresu, kterou jsme získali z odpovědi v minulém kroku.
4. Odpovědí od služby pak bude data ve formátu `json` obsahující informace o zanalyzovaném obrázku.
5. V aplikaci pak podle navrácených informací vykreslím obdélníky okolo jednotlivých slov, na které uživatel může kliknout a uložit je do upravované vlastnosti.

Podrobnější teoretické informace k OCR lze najít v sekci 2.3. Při implementaci mi dost problémů způsobila snaha vykreslit ohraničení kolem jednotlivých slov použitím 4 různě velkých čar. Bohužel jsem měl problémy s modulem pro vytváření vlastních tvarů a tak jsem se musel spokojit s řešením klasických obdélníků. Obrázek pro analyzování nesmí být focen pod nevhodným úhlem.

Uložení objektů do databáze

Uložení objektů do databáze probíhá podobným způsobem jako získávání vlastností podle typu objektů. V tomto případě se ale data posílají do databáze. Jako první se pošle dotaz na vytvoření objektu podle jeho typu. Odpovědí od databáze je pak id nově vytvořeného objektu. V druhém dotazu pak posíláme vlastnosti objektu. Podle počtu naskenovaných sériových čísel se pak tento proces vykoná víckrát.

Data pro odeslání musím samozřejmě připravit. Vzhledem k tomu, že naskenovaná sériová čísla do této doby udržuji v separovaném poli, musím před odesláním vlastností sériové číslo přidat k ostatním vlastnostem. K tomu slouží funkce `createNewObject` a lze ji vidět v ukázce 6.7. Vytvoří nové pole `structuredData`, do kterého postupně cyklem ukládá jméno a hodnotu vlastnosti z pole properties (to obsahuje všechny vlastnosti). Na konec pole navíc přidá vlastnost sériové číslo (v kódu Serial number) s hodnotou. Potom již zavolá funkci `createObjects` pro vytvoření objektu v databázi.

```
1   const createNewObject = () => {
2       listDataSource.map(element => {
3           var structuredData = {
4               value: properties.map((test) => (
5                   { name: test.name, value: test.value }
6               ))
7           }
8           structuredData.value.push({ "name": 'Serial number', "value":
                element.serialNumber });
9
10          createObjects(structuredData);
11      });
12  }
```

Výpis 6.7: Ukázka funkce `createNewObject`, která postupně vytvoří nové pole s připravenými daty k odeslání do databáze. Na konec pole navíc přidá vlastnost Serial number.

6.1.5 Zhodnocení implementace oproti návrhu

Výsledná implementace poměrně dodržela původní návrhy. Nicméně je zde pár drobností, které se od návrhů odlišují.

Hlavní obrazovka, boční menu a obrazovka pro skenování kódů

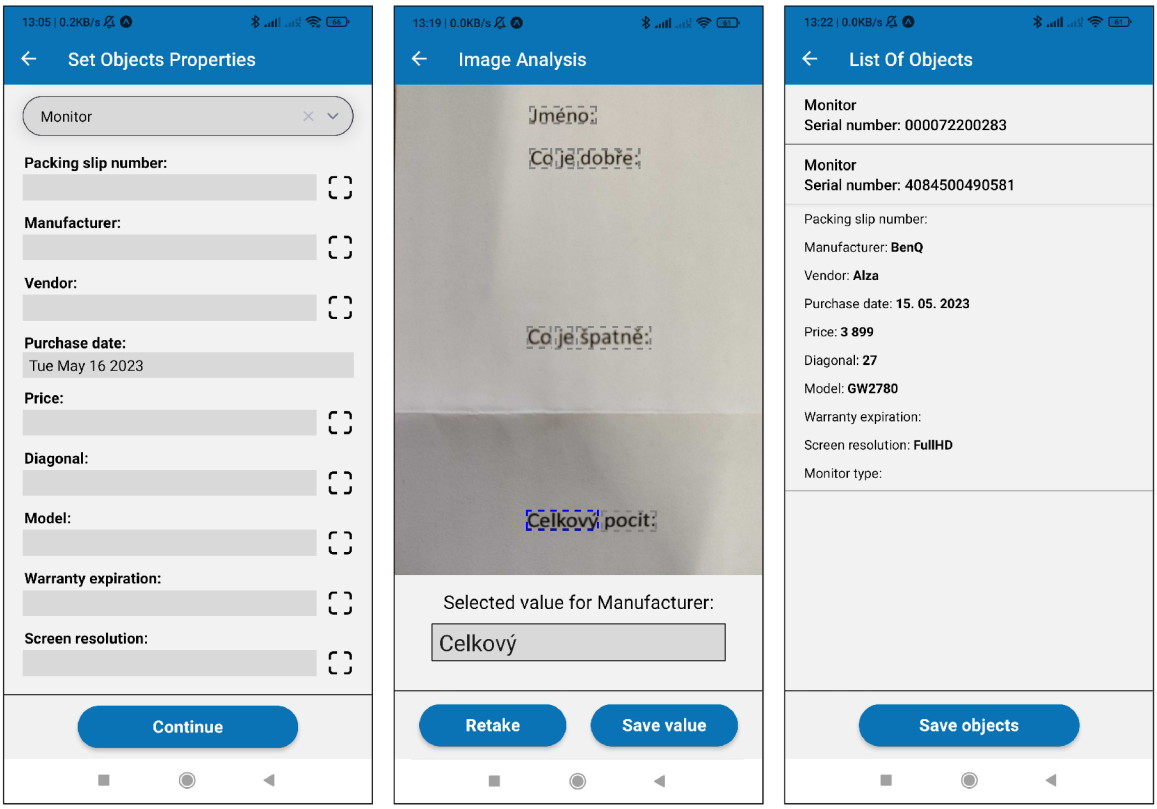
Hlavní obrazovka v podstatě odpovídá návrhu. Co se týče vyhledávacího pole, tato funkcionalita nebyla implementována, protože to nebylo cílem této práce. V bočním výsuvném menu pak zůstalo jen tlačítko pro odhlášení a hlavní obrazovku. Obrazovka pro skenování čárových kódů a QR kódů odpovídá finálnímu návrhu.

Obrazovka pro nastavení vlastností

Tato obrazovka se oproti návrhu lehce liší. Pole pro vybrání typu objektu jsem znatelně odlišil od ostatních textových polí, které slouží pro vyplnění hodnot vlastností. Také jsem se rozhodl změnit název této obrazovky na `Set Objects Properties`, který podle mě více reflektuje hlavní cíl této obrazovky. Tyto změny můžete vidět na obrázku 6.1a.

Obrazovka s využitím technologie OCR

Oproti původnímu návrhu zde došlo k pár úpravám. Pole ukazující vybíranou vlastnost jsem přesunul dolů. To bylo hlavně z důvodu problémů, které jsem měl se zobrazením rámečků okolo analyzovaného textu. Důležité bylo také přidat tlačítko `Retake`, které spustí znovu fotoaparát. Tlačítko `Save value` pak nahradilo původní tlačítko `Continue` z důvodu, že to dávalo větší smysl. Po provedeném závěrečném testování, které se nachází v sekci 6.2 jsem také upravil nadpis této obrazovky, aby i méně znalí uživatelé lépe pochopili, k čemu obrazovka slouží. Tuto implementovanou verzi lze vidět na obrázku 6.1b.



(a) Editování vlastností (b) OCR analýza obrázku (c) Seznam objektů

Obrázek 6.1: Obrázky ukazují reálný vzhled aplikace po provedení implementace. První obrázek zobrazuje obrazovku pro úpravu vlastností nově vytvářených objektů. Druhý potom úpravu vlastností pomocí technologie OCR. Třetí obrázek ukazuje obrazovku s finálním seznamem objektů před uložením do databáze.

Obrazovka se seznamem objektů

Na této obrazovce zmizely oproti původnímu návrhu ikonky vytvářených objektů. Ikony by měli být sjednocené s používaným evidenčním produktem. V evidenčním produktu navíc ikony můžou být upravovány a měněny. Pro implementování funkcionality pro automatickou aktualizaci a stahování ikon z evidenčního systému jsem neměl prostředky a ani časové možnosti.

V původním návrhu se pracovalo s funkcí editování jen jednoho objektu ze všech nově evidovaných objektů. Tuto funkci by bylo hezké v aplikaci mít, bohužel mi na to nezůstal časový prostor. Z obrazovky také zmizelo tlačítko `Edit all`, protože editovat všechny objekty lze návratem na předchozí stránku. Reálný vzhled této obrazovky je zobrazen na obrázku 6.1c.

6.2 Testování aplikace

V této části jsou popsány způsoby, jakými byla aplikace testována a zhodnocení výsledků závěrečného testování.

6.2.1 Způsoby testování aplikace

Jedním ze způsobu testování aplikace bylo tzv. průběžné testování, které probíhalo během vývoje této aplikace. Tento způsob umožnil včas odhalit spoustu chyb a problémů. Testování jsem prováděl z části samostatně, kdy jsem pravidelně nově implementovanou funkcionalitu ověřoval. Testování ale probíhalo také formou konzultací.

Výborným zdrojem zpětné vazby byla konzultace s vedoucím této bakalářské práce s panem prof. Heroutem. Díky jeho zpětné vazbě se výrazně posunula například implementace obrazovky pro skenování kódů, která po prvním kole jejího vývoje byla opravdu nedostatečná. Kromě konzultací s prof. Heroutem jsem aktuální řešení konzultoval také s některými kolegy ze zaměstnání a s tátou, který se vývoji mobilních aplikací již dlouhou dobu věnuje.

Ukončení vývoje mobilní aplikace jsem zakončil trochu větším testováním, na kterém se podílelo 8 uživatelů. Většinu z nich tvořili zejména kolegové ze zaměstnání. Jejich profesní pozice zahrnovaly role testerů, programátorů a vedení společnosti. Zpracování závěrečného testování se nachází v následující části.

6.2.2 Zpracování a zhodnocení testování

Uživatele zapojené do testování bychom mohli rozdělit do dvou skupin. Jedna skupina již předem disponovala poměrně dobrými informacemi o aplikaci (například konzultovali aplikaci již v průběhu vývoje a návrhů), a tak při testování byla pro ně aplikace dostatečně pochopitelná. Naopak druhá skupina o aplikaci měla opravdu minimální informace a s jejím ovládáním si museli poradit převážně sami. Hodnocení uživatelské zkušenosti touto skupinou pak bylo o poznání nižší než u první skupiny. Během procesu testování jsem uživatele při používání aplikace pozoroval a v případě potřeby poskytl vysvětlení nebo nápovědu, jak aplikaci používat. Po objasnění příběhu použití aplikace a vysvětlení drobných nejasností i druhá skupina zvládla aplikaci použít.

Testovací poznatky

Následuje bodový výpis pozitivních a negativních poznatků. Uvedené poznatky obě skupiny viděly buď podobně nebo se navzájem nevylučovaly.

Pozitivní poznatky:

- Po zahájení procesu naskladnění nového majetku přechodem na obrazovku skenování kódů aplikace rovnou skenuje
- Aplikace funguje rychle a nezasekává se
- Aplikace bude mít reálné využití v praxi
- Upravit vlastnost jde pomocí technologie OCR
- Až na obrazovku se skenováním kódů je uživatelské rozhraní aplikace

Negativní poznatky:

- Při skenování kódů se skenuje celá plocha (vhodné by bylo, aby aplikace skenovala jen označenou část uprostřed obrazovky)
- Bylo by vhodné upravit signalizaci naskenovaného kódu a aspoň trochu sjednotit chování s analýzou obrázků pomocí OCR
- Chybí potvrzující informace pro uživatele o uložení objektů
- Obrazovka s implementací OCR by mohla podporovat upravení více vlastností bez nutnosti z obrazovky odejít a vrátit se zpět po kliknutí na úpravu jiné vlastnosti (případně při opětovném přechodu na tuto obrazovku nechat zachovaný poslední stav)
- Při probíhající analýze obrázku technologií OCR by aplikace tuto skutečnost mohla nějakým způsobem uživateli signalizovat
- Název obrazovky *OCR Analysis* nemusí být pro všechny pochopitelný
- Retake, neboli opětovné vyfocení obrázku pro OCR analýzu vyvolá pád aplikace

Zhodnocení testování

Výslednou aplikaci uživatelé ohodnotili pozitivně s tím, že je ale potřeba se zaměřit na opravu výše zmíněných nedostatků. Nejvíce pak při hodnocení pochválili možnost využití technologie OCR pro analýzu textu v obrázku a vyplnění hodnoty vlastnosti vybráním slova po provedení analýzy. Naopak největším nedostatkem a prostorem pro zlepšení je podle nich neideální uživatelské rozhraní na obrazovce skenování čárových kódů.

Po provedeném testování se mi do výsledné aplikace ještě podařilo opravit funkci `Retake`, která umožňuje znova vyfotit obrázek pro jeho analýzu. Zároveň jsem také přidal indikaci právě probíhající analýzy pomocí točícího kolečka. Jako poslední se mi podařilo přidat hlášku s informací o vytvoření objektů v databázi. Na další opravy již nezbyl čas.

Aplikace stále obsahuje některé nedostatky a chyby, které je potřeba vyřešit. Po dokončení bakalářského studia mám v plánu opravit chyby v aplikaci a dál s vývojem pokračovat, protože je potřeba udělat ještě kus práce před tím, než bude aplikace vypuštěna zákazníkům.

Kapitola 7

Závěr

Evidování majetku je nezbytnou součástí jakékoliv organizace. Ačkoliv je možné evidovat majetek pomocí klasických kancelářských aplikací, jako je Excel, pro komplexnější potřeby je vhodné použít specializovaný software. V této bakalářské práci jsem se zaměřil na vývoj mobilní aplikace, která by usnadnila proces vytváření nového majetku v evidenčním softwaru.

Pro vytvoření práce bylo potřeba se seznámit s vývojem mobilních aplikací převážně pro platformu Android. Dále jsem se zabýval evidencí majetku v organizacích, zejména tedy procesem naskladnění nového majetku. Následně jsem provedl analýzu požadavků na aplikaci, přičemž jsem vycházel z uživatelského příběhu naskladnění nového majetku a současných omezení tohoto procesu. Současně s analýzou požadavků jsem se věnoval průzkumu konkurenčních řešení procesu evidování nového majetku. Zajímavým zjištěním bylo, že se mi nepodařilo najít nějakou mobilní aplikaci, která by podporovala hromadné evidování nového majetku, což potvrzuje relevanci mé práce.

V další fázi jsem se zaměřil na návrh uživatelského rozhraní aplikace, které jsem průběžně konzultoval a upravoval. Po té jsem prvky uživatelského rozhraní implementoval do aplikace a zároveň implementované části průběžně testoval a konzultoval. Důležité části uživatelského rozhraní a aplikace jsou popsány v tomto dokumentu. Součástí zadání práce bylo také vytvořit plakátek aplikace a demonstrační video. To se nachází na přiloženém paměťovém médiu.

Zadání bakalářské práce bylo splněno. Mobilní aplikace zvládne naskenovat sériová čísla nového majetku pomocí čárových kódů a QR kódů, vyplnit potřebné vlastnosti a vytvořit odpovídající objekty v evidenčním softwaru. Práce mi umožnila si poprvé vyzkoušet a naučit se proces vývoje mobilních aplikací. Také bylo velice zajímavé v aplikaci implementovat možnost využití technologie OCR.

S výslednou aplikací jsem poměrně spokojený a mám v plánu po dokončení studia na ní pokračovat. Rád bych opravil chyby, které se v aplikaci objevují a poupravil uživatelské rozhraní obrazovky pro skenování kódů. Dále bych chtěl do aplikace implementovat nové funkcionality, jako je vyhledávání majetku již uloženého v evidenci a možnost ho upravovat. Proces naskladnění bych chtěl také ulehčit přidáním možnosti OCR analýzy dodacího listu, která by z něj automaticky vytáhla a doplnila data do vlastností odpovídajících objektů. Nakonec by také bylo dobré do aplikace přidat možnost provádět fyzické inventury.

Literatura

[1] ALVAO. *Evidence IT majetku | ALVAO* [online]. [cit. 2023-05-15]. Dostupné z: https://www.alvao.com/cs/it-asset-management.

[2] Alvao. *ALVAO database* [online]. 2023 [cit. 2023-05-07]. Dostupné z: https://doc.alvao.com/en/11.1/alvao-asset-management/implementation/customization/database.

[3] Business, P. for. *What is the Difference Between a Barcode and a QR Code?* [online]. 19. října 2022 [cit. 2023-05-07]. Dostupné z: https://pwpblog.wpengine.com/difference-between-barcode-and-qr-code-kb.

[4] Deshpande, C. *Flutter vs. React Native: Which One to Choose in 2023?* [online]. Simplilearn, 25. dubna 2023 [cit. 2023-04-30]. Dostupné z: https://www.simplilearn.com/tutorials/reactjs-tutorial/flutter-vs-react-native.

[5] Dhaduk, H. *React Native vs Ionic: Which Framework is best and Why?* [online]. Simform, 20. února 2023 [cit. 2023-04-30]. Dostupné z: https://www.simform.com/blog/react-native-vs-ionic/.

[6] Expo Team. *BarCodeScanner - Expo Documentation* [online]. [cit. 2023-05-14]. Dostupné z: https://docs.expo.dev/versions/latest/sdk/bar-code-scanner.

[7] Expo Team. *Expo CLI - Expo Documentation* [online]. 2023 [cit. 2023-04-29]. Dostupné z: https://docs.expo.dev/more/expo-cli.

[8] Expo Team. *Reference - Expo Documentation* [online]. 2023 [cit. 2023-04-29]. Dostupné z: https://docs.expo.dev/versions/latest.

[9] Frachet, M. *JS threads communicates with the native ones through the bridge* [online]. HackerNoon, 10. listopadu 2017 [cit. 2023-04-26]. Dostupné z: https://hackernoon.imgix.net/hn-images/1*JT_Smf1u3fJTBY8ev9WAzg.png.

[10] Frachet, M. *Understanding the React Native bridge concept | HackerNoon* [online]. HackerNoon, 10. listopadu 2017 [cit. 2023-04-26]. Dostupné z: https://hackernoon.com/understanding-react-native-bridge-concept-e9526066ddb8.

[11] Gakure, J. *Xamarin Vs React Native For Mobile Apps – A Comparison* [online]. OpenReply, 6. ledna 2023 [cit. 2023-04-30]. Dostupné z: https://blog.openreplay.com/xamarin-vs-react-native-for-mobile-apps--a-comparison/.

[12] GetApp. *Sortly Pricing, Features, Reviews & Alternatives | GetApp* [online]. [cit. 2023-05-10]. Dostupné z: https://www.getapp.com/operations-management-software/a/sortly-pro.

[13] Hat, R. *What is a REST API?* [online]. 8. dubna 2020 [cit. 2023-05-02]. Dostupné z: https://www.redhat.com/en/topics/api/what-is-a-rest-api.

[14] Itemit. *How To Create An Asset Register?* [online]. 24. srpna 2021 [cit. 2023-05-07]. Dostupné z: https://itemit.com/how-to-create-an-asset-register.

[15] Itemit. *A Quick Demo of itemit* [online]. 12. října 2021 [cit. 2023-05-10]. Dostupné z: https://www.youtube.com/watch?v=W5ejuzuJ4K0&ab_channel=itemit.

[16] Kaneko, J. *Understanding React Native Architecture - DEV Community* [online]. DEV Community, 27. srpna 2020 [cit. 2023-04-26]. Dostupné z: https://dev.to/goodpic/understanding-react-native-architecture-22hh.

[17] Kulkarni, A. *API vs REST API Simplified: 6 Critical Differences* [online]. Hevo Data, 19. října 2021 [cit. 2023-05-02]. Dostupné z: https://hevodata.com/learn/api-vs-rest-api/.

[18] Microsoft. *Microsoft Cognitive Services* [online]. 2022 [cit. 2023-05-02]. Dostupné z: https://westus.dev.cognitive.microsoft.com/docs/services/computer-vision-v3-2/operations/5d9869604be85dee480c8750.

[19] Oracle. *What Is a Database | Oracle* [online]. 27. září 2021 [cit. 2023-05-02]. Dostupné z: https://www.oracle.com/database/what-is-database.

[20] Panda, A. *Asset Panda Web and Mobile Demo* [online]. 15. dubna 2021 [cit. 2023-05-10]. Dostupné z: https://www.youtube.com/watch?v=IaIR_Ker6jI&ab_channel=AssetPanda.

[21] PatrickFarley, luzhang06, koudaiii, sanjeev3, hmacgregor1 et al. *OCR – optické rozpoznávání znaků - Azure Cognitive Services* [online]. Microsoft, 15. března 2023 [cit. 2023-04-30]. Dostupné z: https://learn.microsoft.com/cs-cz/azure/cognitive-services/computer-vision/overview-ocr.

[22] Praveenruhil. *Introduction to Android Development - GeeksforGeeks* [online]. GeeksforGeeks, 20. dubna 2023 [cit. 2023-04-25]. Dostupné z: https://www.geeksforgeeks.org/introduction-to-android-development.

[23] React Native Team. *Core Components and Native Components · React Native* [online]. 12. ledna 2023 [cit. 2023-04-26]. Dostupné z: https://reactnative.dev/docs/intro-react-native-components.

[24] React Native Team. *Fast Refresh · React Native* [online]. 12. ledna 2023 [cit. 2023-04-27]. Dostupné z: https://reactnative.dev/docs/fast-refresh.

[25] React Native Team. *Introduction · React Native* [online]. 17. března 2023 [cit. 2023-04-29]. Dostupné z: https://reactnative.dev/docs/getting-started.

[26] React Native Team. *Native Modules Intro · React Native* [online]. 12. ledna 2023 [cit. 2023-04-26]. Dostupné z: https://reactnative.dev/docs/native-modules-intro.

[27] React Native Team. *Platform Specific Code · React Native* [online]. 12. ledna 2023 [cit. 2023-04-27]. Dostupné z: https://reactnative.dev/docs/platform-specific-code.

[28] React Navigation Team. *Drawer Navigator* [online]. [cit. 2023-05-14]. Dostupné z: https://reactnavigation.org/docs/drawer-navigator.

[29] React Navigation Team. *Stack Navigator* [online]. [cit. 2023-05-14]. Dostupné z: https://reactnavigation.org/docs/stack-navigator.

[30] Rouse, M. *What is a mobile application?* [online]. Techopedia, 7. srpna 2020 [cit. 2023-04-08]. Dostupné z: https://www.techopedia.com/definition/2953/mobile-application-mobile-app.

[31] Ročkár, M. *Vývoj multiplatformovej aplikácie pre spoločnosť ALVAO s. r. o.* [online]. 2022 [cit. 2023-05-07]. Dostupné z: https://is.mendelu.cz/zp/index.pl?prehled=vyhledavani;podrobnosti_zp=71765;zp=71765;dinfo_jazyk=2;lang=cz.

[32] Schmitt, J. *Native vs cross-platform mobile app development | CircleCI* [online]. CircleCI, 24. srpna 2022 [cit. 2023-04-13]. Dostupné z: https://circleci.com/blog/native-vs-cross-platform-mobile-dev/#c-consent-modal.

[33] Simplilearn. *OCR – optické rozpoznávání znaků - Azure Cognitive Services* [online]. Simplilearn, 23. února 2023 [cit. 2023-04-30]. Dostupné z: https://www.simplilearn.com/what-is-ocr-optical-character-recognition-article.