UNIVERZITA PARDUBICE

FAKULTA EKONOMICKO-SPRÁVNÍ

BAKALÁŘSKÁ PRÁCE

Vývoj a zabezpečení mobilní aplikace

2024 **Pavel Plaček**

Univerzita Pardubice
Fakulta ekonomicko-správní
Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Pavel Plaček**
Osobní číslo:	**E21590**
Studijní program:	**B0688A140004 Informatika a systémové inženýrství**
Specializace:	**Informační a bezpečnostní systémy**
Téma práce:	**Vývoj a zabezpečení mobilní aplikace**
Zadávající katedra:	**Ústav systémového inženýrství a informatiky**

Zásady pro vypracování

Cílem práce je naprogramovat a zabezpečit mobilní aplikaci.

Osnova:
– Charakterizovat postup a nástroje pro vývoj mobilních aplikací v současnosti.
– Navrhnout ukázkovou mobilní aplikaci.
– Naprogramovat ukázkovou aplikaci dle stanoveného postupu.

Rozsah pracovní zprávy: **cca 35 stran**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

BIGELOW, Stephen. *Operating system (OS)* [online]. Dostupné z: https://www.techtarget.com/whatis/definition/operating-system-OS
HUB, Miloslav. *Bezpečnost a ochrana informací v prostředí internetu*. Pardubice: Univerzita Pardubice, 2013. ISBN 978-80-7395-701-8
META PLATFORMS. *React* [online]. Dostupné z: https://react.dev/
UDEMY. *The Complete React Native + Hooks Course* [online]. Dostupné z: https://www.udemy.com/

Vedoucí bakalářské práce: **RNDr. Ing. Oldřich Horák, Ph.D.**
Ústav systémového inženýrství a informatiky

Datum zadání bakalářské práce: **1. září 2023**
Termín odevzdání bakalářské práce: **30. dubna 2024**

L.S.

prof. Ing. Jan Stejskal, Ph.D. v.r.
děkan

Ing. et Ing. Martin Lněnička, PhD. v.r.
garant studijního programu

V Pardubicích dne 1. září 2023

PROHLÁŠENÍ

Prohlašuji:

Práci s názvem *Vývoj a zabezpečení webové aplikace* jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne

Pavel Plaček

PODĚKOVÁNÍ

Tímto bych chtěl poděkovat vedoucímu práce panu RNDr. Ing. Oldřichovi Horákovi, Ph.D. za cenné rady týkající se této bakalářské práce.

ANOTACE

Bakalářská práce je zaměřena na vývoj a zabezpečení mobilní aplikace na platformě Android. V úvodu je zmíněna motivace k výběru tématu, další kapitoly postupně představují pohled na operační systémy se zaměřením na klasické počítače až po mobilní platformy.

Následuje popis vývojových prostředků, které byly v práci využity a důležitost zabezpečení. V praktické části je řešena příprava a vlastní vývoj aplikace. Práce je zakončena závěrečným shrnutím.

KLÍČOVÁ SLOVA

Operační systém, mobilní aplikace, React Native, Visual Studio, Android Simulator, Android, technologie pro webový vývoj, zabezpečení, biometrická autentizace.

TITLE

Mobile application development and security.

ANNOTATION

The bachelor thesis focuses on the development and security of a mobile application on the Android platform. The motivation for the choice of the topic is mentioned in the introduction, the following chapters gradually present a view of operating systems, focusing on traditional computers to mobile platforms.

This is followed by a description of the development tools that were used in the thesis and the importance of security. The practical part deals with the preparation and actual development of the application. The thesis concludes with a final summary.

KEY WORDS

Operating system, mobile apps, React Native, Visual Studio, Android Simulator, Android, web development technologies, security, biometric authentication.

OBSAH

SEZNAM ILUSTRACÍ A TABULEK

SEZNAM ZKRATEK A ZNAČEK

API	Application Programming Interface
CLI	Command Line Interactive
CMD	Command Prompt
CSS	Cascading Style Sheets
GPS	Global Positioning System
GUI	Graphic User Interface
HDMI	High-Definiton Multimedia Interface
IMB	International Business Machines Corporation
JSON	JavaScript Object Notation
JSX	JavaScript XML
OS	Operační Systém
PCIe	Peripheal Component Interconnect Express
UI/UX	User Interface/User Expirience
USB	Universal Serial Bus
VGA	Video Graphic Array
XML	Extensible Markup Language

ÚVOD

S mobilními aplikacemi se dostáváme do styku téměř dennodenně. Jsou součástí života většiny lidí. Obliba mobilních aplikací neustále roste díky jejich širokému použití. Mohou se používat v rámci marketingu (propagace firem, integrace produktů a služeb do kompaktního rozhraní), v rámci osobní potřeby každodenního života (monitoring zdraví, elektronická pošta) až po herní aplikace, tím pádem zábavu.

Tato práce se zabývá vývojem mobilní aplikace pomocí programovacího jazyka *React Native*. Bude se jednat o aplikaci kompatibilní s operačním systémem *Android* a vyvíjena bude ve vývojovém prostředí *Microsoft Visual Studio* a ve vývojovém prostředí *Android Studio*. Uvedu zde i informace ohledně dalšího, velmi populárního operačního systému, a to *macOS* a *iOS* od firmy Apple a související vývojové prostředí pro vývoj mobilních aplikací v prostředí macOS, *XCode*.

Programování aplikací pro operační systém Android a iOS je za pomocí programovacího jazyka React Native dosti podobné, ne-li stejné, tudíž je vhodné si uvědomit, že se programátor nemusí učit striktně dva programovací jazyky a dvě různé syntaxe kódu. Kdyby tomu tak nebylo, musel by se programátor naučit například programovací jazyk *Kotlin* pro vývoj na zařízení s Androidem a programovací jazyk *Swift* pro zařízení s iOS. S React Native se nám změní jen rozdílné vývojové prostředí dle našich preferencí či operační systém, se kterým budeme při vývoji spolupracovat.

Cílem práce je naprogramovat a zabezpečit mobilní aplikaci.

1 Operační systém

Operační systém (zkráceně OS) je program, který po prvotním nahrání do počítače spouštěcím programem spravuje všechny ostatní aplikační programy v počítači. Aplikační programy využívají operační systém tak, že prostřednictvím definovaného aplikačního programového rozhraní (API) zadávají požadavky na služby. Kromě toho mohou uživatelé komunikovat s operačním systémem přímo prostřednictvím uživatelského rozhraní, například rozhraní příkazového řádku (CLI) nebo grafického uživatelského rozhraní (GUI). [1]

Historie operačních systémů

V dřívějších dobách operační systémy jako takové neexistovaly a programátor musel komunikovat s počítačem pomocí strojového kódu (na úrovni 1–0). S vývojem HW začaly vznikat první programovací jazyky. Programovací jazyky mohou být buď vázány na *konkrétní hardware* (např. Assembler), nebo jsou na hardware *nezávislé* (tzv. vyšší programovací jazyky). Nejstarším vyšším programovacím jazykem je *Short Code* z roku 1949, později vznikl Fortran (1956, vyvinut IBM), COBOL (1959), BASIC (1965, později standardní jazyk pro PC), Pascal (1971), C (1972) atd. [2]

S dalším rozvojem však bylo třeba programu, který by sám zvládal základní funkce systému a ulehčil programátorům práci. Počátkem 60. let tak pomalu začaly vznikat operační systémy a v polovině 60. let (s příchodem minipočítačů) již vyvstává potřeba takových operačních systémů, jaké známe dnes. Mezi nejznámější a nejrozšířenější platformy patří především operační systém *Windows* od firmy Microsoft a operační systémy na bázi *UNIXu*. Mezi nejznámější operační systémy na bázi UNIXu patří *Linux* a *Mac OS X*. Významným prvkem ve vývoji operačních systémů bylo vytvoření *grafického uživatelského rozhraní* (GUI –graphics unit interface). [2]

Služby operačního systému

Další definice operačního systému jej charakterizuje jako:

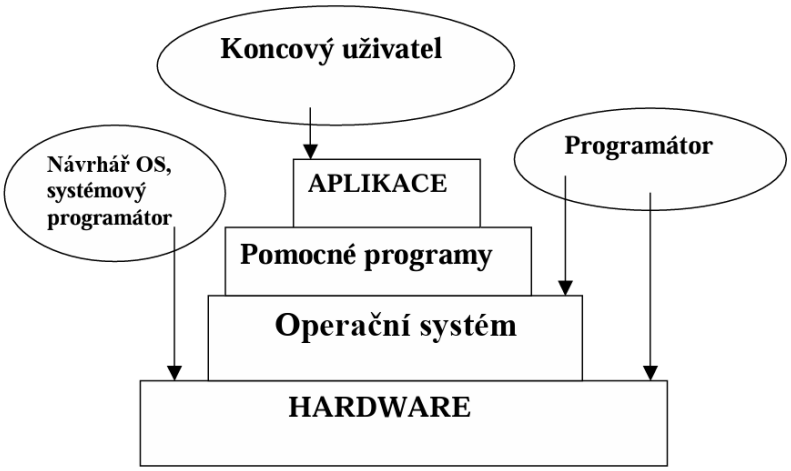
- **Správce prostředků** – spravuje a přiděluje zdroje systému.
- **Řídicí program** – řídí provádění uživatelských programů a operací I/O zařízení.
- **Jádro** – trvale běžící program – všechny ostatní programy lze chápat jako aplikační. [2]

Všechny tyto definice operačního systému oddělují striktně jednotlivé komponenty výpočetního systému, jak je zřejmé z ukázky v *Příloze A*.

Z předchozích definic vyplývají oblasti služeb operačních systémů, které jsou zřejmé z *Přílohy B*.

Struktura operačního systému

Cílem operačního systému je rovněž zajištění pohodlnosti používání počítače, tzn. že operační systém je správcem rozhraní *člověk/stroj* a správcem rozhraní *proces/operační systém*. [2]



Obrázek 1, struktura operačního systému

Zdroj: převzato a upraveno dle [2]

Operační systém z hlediska rozhraní člověk/stroj typicky poskytuje služby pro vytváření programů a jejich spouštění, běh procesů, obsluha I/O zařízení, řízení přístupu k souborům a k detekci chyb. [2]

Dalším cílem operačního systému je zajištění dostatečné výkonnosti a efektivity. Operační systém je tak správcem systémových prostředků (procesorů, paměti, vstupních a výstupních zařízení a souborů) a správcem jejich užívání. Operační systém řídí přístup k vstupním a výstupním zařízením a souborům, provádí správu paměti a určuje, který program bude používat procesor. Operační systém musí zajistit schopnost vývoje, tj. doplňování a vývoj hardware a doplňování nových služeb do počítačového systému. Operační systém musí být schopen reagovat na inovace v technických prostředcích, na nové komponenty počítačů. [2]

Operační systémy, z hlediska své struktury, lze popsat architekturou monolitickou, víceúrovňovou, virtuální a klient-server. [2]

Uživatelské rozhraní

Každý operační systém vyžaduje uživatelské rozhraní, které uživatelům a správcům umožňuje interakci s operačním systémem za účelem nastavení, konfigurace, a dokonce i řešení problémů s operačním systémem a jeho základním hardwarem. Existují dva základní typy uživatelského rozhraní: *CLI* a *GUI*. [1]

CLI neboli okno v terminálovém režimu poskytuje textové rozhraní, kde se uživatelé spoléhají na tradiční klávesnici při zadávání konkrétních příkazů, parametrů a argumentů souvisejících s konkrétními úlohami. GUI neboli pracovní plocha poskytuje vizuální rozhraní založené na ikonách a symbolech, kde se uživatelé spoléhají na gesta poskytovaná zařízeními s lidským rozhraním, jako jsou touchpady, dotykové obrazovky a myši. [1]

Grafické uživatelské rozhraní nejčastěji používají běžní nebo koncoví uživatelé, kteří se zajímají především o manipulaci se soubory a aplikacemi, například dvojklikem na ikonu souboru otevřou soubor v jeho výchozí aplikaci. Rozhraní CLI zůstává oblíbené mezi pokročilými uživateli a správci systému, kteří musí pravidelně zpracovávat řadu velmi podrobných a opakujících se příkazů, například vytvářet a spouštět skripty pro nastavení nových osobních počítačů pro zaměstnance. [1]

Operační systém může také podporovat rozhraní *API*, která aplikacím umožňují využívat funkce operačního systému a hardwaru (např. API Windows a klávesnice). [1]

Role operačního systému v softwarovém vývoji

Operační systém přináší počítačovému softwaru a vývoji softwaru silné výhody. Bez operačního systému by každá aplikace musela obsahovat vlastní uživatelské rozhraní a také rozsáhlý kód potřebný k obsluze všech nízkoúrovňových funkcí základního počítače, jako je diskové úložiště, síťová rozhraní atd. Vzhledem k obrovskému množství dostupného základního hardwaru by to značně zvětšilo velikost každé aplikace a znemožnilo by to vývoj softwaru. [1]

Mezi nejběžnější a nejznámější operační systémy, se kterými se dostáváme do styku denně, jsou například *Microsoft Windows*, *macOS* od firmy Apple. Dále mobilní operační systémy, a to *Google Android* či *iOS* od firmy Apple. [3] V následujících kapitolách jsou tyto operační systémy popsány.

1.1 Windows

Operační systém Windows je počítačový program, který spravuje všechny prostředky počítače a poskytuje služby aplikacím, které „nad“ ním běží. Tento operační systém byl vyvinut společností Microsoft a vydán v roce 1985 pod názvem *Windows 1.0*. Od té doby se operační systém Windows neustále vyvíjel a stal se jedním z nejoblíbenějších operačních systémů na světě. [4]

Operační systém Windows je navržen pro provoz na různých typech hardwaru, včetně stolních počítačů, notebooků, serverů a mobilních zařízení. Systém Windows používá grafické uživatelské rozhraní (GUI), které umožňuje uživatelům komunikovat s počítačem pomocí ikon, tlačítek a vizuálních nabídek. Systém Windows má také mnoho funkcí, jako je například možnost multitaskingu, která umožňuje současný běh několika aplikací, a také možnost plug-and-play, která uživatelům usnadňuje připojení dalších zařízení, jako jsou tiskárny, skenery a fotoaparáty. [4]

Historie Windows

Historie operačního systému Windows sahá do roku 1981, kdy firma IBM uvedla na trh první PC spolu se 16bitovým operačním systémem *MS-DOS* (Microsoft Disk Operating System). Pro další vývoj počítačového průmyslu a šíření operačních systémů od Microsoftu se ukázalo jako rozhodující, že firma IBM umožnila výrobu klonů svých počítačů – velmi rychle se pak rozšířily do celého světa. [4]

Další verze Windows

Po systému Windows 1.0 s prvním GUI přicházely další verze. Jsou to *Windows 2.0, Windows 3.0, Windows 95*, *Windows 98, Windows 2000, Windows XP, Windows Vista, Windows 7, Windows 8, Windows 10* a *Windows 11*. Každá verze tohoto operačního systému nabízela různá vylepšení a opravy, včetně zlepšení výkonu, vylepšeného rozhraní, podpory nového hardwaru a softwaru a lepších bezpečnostních funkcí. [4]

Windows 11

Windows 11 se dodávají na trh v několika verzích, například *Windows 11 Home* (určen pro domácí použití), *Windows 11 Pro* (pro malé firmy), *Windows 11 Enterprise* (pro velké firmy) či *Windows 11 IoT* (pro vestavěná zařízení). [5]

Novinky oproti Windows 10

Nejdůležitější novinky: Nové grafické prostředí (GUI), které bylo více sjednoceno, nový vzhled přihlašovací plochy a nabídky Start, vylepšený systém aktualizací, přepracovaný katalog Microsoft Store, možnost spouštění Android aplikací, snadnější práce s okny atd. [5]

Vybrané aplikace systému Windows 11

Pozn., všechny popisované aplikace jsou dostupné v nabídce *Start.*

Výčet aplikací: Kalendář**,** Pošta (tato aplikace má společné okno s aplikací Kalendář, lze mezi nimi přecházet) Počasí a Přehrávač médií, Microsoft Store, Fotky, Malování 3D**,** To Do**,** Editor videa Clipchamp. Naopak zmizeli aplikace Cortana či Internet Explorer. [5]

Funkce operačního systému Windows

Funkce operačního systému Windows jsou klíčové pro spouštění různých aplikací a programů na počítačích nebo noteboocích. [4]

Mezi některé funkce operačního systému Windows patří:

- **Správa prostředků počítače:** Hlavní funkcí operačního systému Windows je správa a organizace počítačových zdrojů, jako je procesor, paměť RAM a pevný disk. Operační systém Windows pomocí těchto prostředků provádí různé úlohy, jako je otevírání aplikací, přístup na internet a tisk dokumentů. [4]
- **Poskytování rozhraní:** poskytnutí grafického uživatelského rozhraní (GUI). [4]
- **Zajištění kompatibility:** usnadnění instalace a používání aplikací a programů, lze spouštět programy a aplikace určené pro různé verze operačního systému Windows. [4]
- **Usnadnění nastavení sítě:** OS obsahuje nástroje usnadňující nastavení sítě. [4]
- **Usnadnění zabezpečení:** bezpečnostní funkce, jako je antivirová ochrana, ochrana proti malwaru a brána firewall, nástroje pro konfiguraci zabezpečení sítě a řízení přístupu k ochraně důležitých dat uživatele. [4]
- **Správa správy souborů:** správa souborů (ukládání dat, konfigurace přístupových práv a vyhledávání souborů). [4]

Název Windows

Pro zajímavost zde uvedu, proč se tomuto systému říká Windows (česky Okna). V době, kdy Microsoft Windows nebyl uveden na trh, používali všichni uživatelé Microsoft operační systém MS-DOS. Společnost Microsoft dala většině svých produktů jedno slovo; vyžadovala nové slovo, které by mohlo reprezentovat její nový operační systém s grafickým uživatelským rozhraním. [6]

Společnost Microsoft se rozhodla nazvat jej Windows, protože má schopnost provádět několik úloh a spouštět aplikace současně. Dalším důvodem, proč jej nazval Windows, byla skutečnost, že nebylo možné použít ochrannou známku na běžný název, jako je Windows. Jeho oficiální název byl Microsoft Windows. [6]

1.2 MacOS

macOS (dříve známý jako Mac OS X) je operační systém vyvinutý společností Apple Inc. pro řadu osobních počítačů a pracovních stanic Mac. Zkratka „macOS“ znamená „Macintosh Operating System“. Poprvé byl představen v roce 2001 jako nástupce klasického operačního systému Mac OS X. Od té doby prošel mnoha aktualizacemi a vylepšeními, aby se stal propracovaným a uživatelsky přívětivým operačním systémem, jakým je dnes. [7]

Popis macOS

macOS je postaven na architektuře založené na UNIXu a poskytuje uživatelům stabilní, spolehlivé a uživatelsky přívětivé prostředí. Operační systém je navržen tak, aby byl intuitivní a snadno použitelný, klade důraz na jednoduchost a eleganci. Uživatelské rozhraní systému macOS je čisté, přehledné a estetické, takže je příjemné ho používat jak pro běžné, tak pro profesionální uživatele. [7]

Kromě elegantního designu a uživatelsky přívětivého rozhraní je macOS známý také svou stabilitou a zabezpečením. Díky svým robustním bezpečnostním funkcím pomáhá operační systém chránit data a soukromí uživatelů, a je tak oblíbenou volbou pro jednotlivce i organizace. [7]

Historie Mac OS

Operační systém byl uveden na trh v roce 1984 a slouží k provozu řady osobních počítačů Macintosh. Macintosh předznamenal éru systémů s grafickým uživatelským rozhraním a inspiroval společnost Microsoft Corporation k vývoji vlastního grafického

rozhraní, operačního systému Windows. Současný operační systém macOS však za mnohé vděčí jiným počítačovým projektům, z nichž některé nevznikly pod vedením společnosti Apple. [8]

macOS byl na rozdíl od jiných OS stavěn na GUI. Dříve OS nepoužívali GUI. Marketing společnosti Apple se při uvedení počítače Macintosh zaměřil především na intuitivní a snadné používání operačního systému. Namísto zadávání příkazů a adresářových cest na textové výzvy se uživatelé pohybovali ukazatelem myši a vizuálně procházeli *Finderem* – řadou virtuálních složek a souborů reprezentovaných ikonami. Většina počítačových operačních systémů nakonec převzala model grafického uživatelského rozhraní. [8]

Mac OS X

Pozdější verze operačního systému Mac OS přinesly významné funkce, jako je sdílení souborů na internetu, procházení sítě a podporu více uživatelských účtů. V roce 1996 Apple převzal konkurenční společnost *NeXT Computers*, kterou založil *Steve Jobs*, a v roce 2001 uvedl na trh zcela přepracovaný systém Mac OS X. Tato verze běžela na jádře UNIXu a nabízela pokročilé funkce, jako je ochrana paměti a preemptivní multitasking, spolu s novým uživatelským rozhraním Aqua a grafickou lištou Dock. [8]

Aktualizace systému Mac OS X přinášely další inovace, včetně automatického zálohování a správce Dashboard pro malé praktické aplikace zvané widgety. Od roku 2007 Apple představil řadu mobilních zařízení, jako iPhone a iPad, a zdůraznil schopnost propojení s operačním systémem Mac OS X. Další funkce propojení, jako je možnost přijímat telefonní hovory a rychlé sdílení dat, byly postupně integrovány do těchto systémů. V roce 2016 Apple přejmenoval systém na *macOS*, aby lépe odpovídal ostatním platformám, jako je iOS a watchOS. [8]

Struktura a architektura operačního systému Mac OS

Důležitou součástí počítače Mac je *firmware*. Firmware je úroveň programování, která existuje přímo nad hardwarovou vrstvou. Není součástí samotného operačního systému. Firmware Macu je první uložený program, který se spustí po zapnutí počítače Mac. Jeho úkolem je kontrolovat procesor, paměť, diskové jednotky a porty počítače, zda neobsahují chyby. Ekvivalent firmwaru Macu pro PC se nazývá *BIOS*, což je zkratka pro základní vstupně-výstupní systém. Druhý program zvaný zavaděč načte operační systém Mac OS X za předpokladu, že firmware nehlásí žádné chyby. [9]

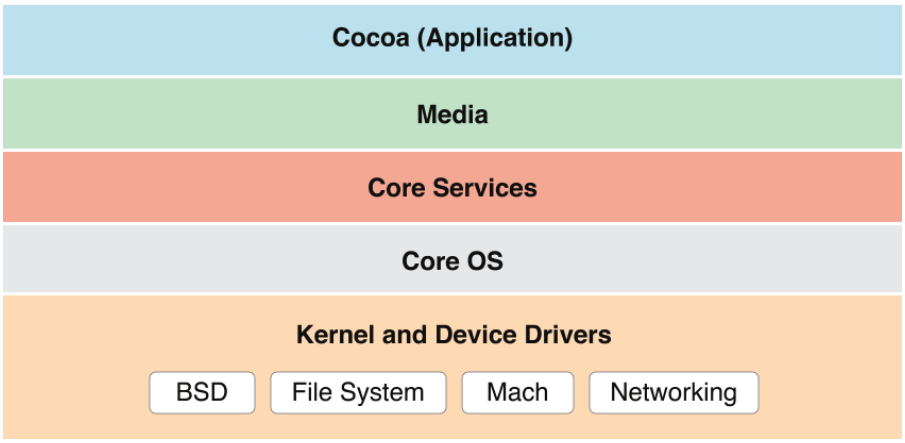
Nejnižší vrstva systému OS X zahrnuje jádro, ovladače a části systému BSD (Berkeley Software Distribution) a je založena především na technologiích open source. Systém OS X rozšiřuje toto nízkoúrovňové prostředí o několik základních infrastrukturních technologií, které usnadňují vývoj softwaru, například připojení k internetu, souborový systém či jádro Mach pro správu paměti, procesů atd. [10]

Technologie a rámce ve vrstvě *Core OS* poskytují nízkoúrovňové služby související s hardwarem a sítěmi. Tyto služby jsou založeny na zařízeních ve vrstvě jádra a ovladačů zařízení. [10]

Technologie ve vrstvě *Core Services* se nazývají základní služby, protože poskytují základní služby aplikacím, ale nemají přímý vliv na uživatelské rozhraní aplikace. Obecně jsou tyto technologie závislé na frameworcích a technologiích ve dvou nejnižších vrstvách systému OS X – tedy ve vrstvě Core OS a ve vrstvě Kernel a Device Drivers. [10]

Vrstva *Media* poskytuje multimediálním aplikacím potřebné funkce a rozhraní, např. formáty obrázků, streamovací protokoly, tvorba animací atd. [10]

Aplikační vrstva *Cocoa* je primárně zodpovědná za vzhled aplikací a jejich reakce na akce uživatele. Kromě toho je ve vrstvě Cocoa implementováno mnoho funkcí, které definují uživatelské prostředí systému OS X, jako je centrum oznámení, režim celé obrazovky a automatické ukládání. [10] Na *Obrázku č. 2* jsou znázorněny jednotlivé vrstvy.



Obrázek 2, vrstvy operačního systému macOS

Zdroj: [10]

XNU

XNU je jádro operačního systému a název XNU je zkratka pro *X is Not UNIX* (česky X není UNIX). Je vyvíjeno firmou Apple a použito jako součást operačních systémů Darwin, macOS (Apple OS X) a iOS.

Jedná se o hybridní jádro unixového typu postavené okolo mikrojádra Mach 3, za použití kódu operačního systému FreeBSD a vlastního API zvaného I/O Kit.

XNU byl původně vyvíjen firmou NeXT pro její operační systém NeXTSTEP. Poté, co Apple skoupil NeXT, byl Mach 2.5 nahrazen Machem verze 3, které se mezitím stalo mikrojádrem, části BSD kódu byly nahrazeny kódem z FreeBSD a Driver Kit byl nahrazen vlastním API zvaným I/O Kit.

Zdrojové kódy XNU (stejně jako celého Darwinu) jsou uvolněny jako open source pod licencí Apple Public Source License 2.0. V současné době je XNU portováno na procesorové platformy x86, x86-64, PowerPC (32 a 64 bit) a ARM (32 a 64 bit). [11]

K datu 2024 je nejnovější verze macOS Sonoma, známá také jako macOS 14. Předešlé verze jsou například macOS 13, Ventura, macOS 12 Monterey či macOS 11, Big Sur. Jednotlivá vydání se liší cca 1 rok od sebe, co se týče vydání jednotlivých verzí. [12]

2 Mobilní operační systémy

Mobilní operační systém je software, který umožňuje smartphonům, tabletům a dalším zařízením spouštět aplikace a programy.

Mobilní operační systém poskytuje rozhraní mezi hardwarovými součástmi zařízení a jeho softwarovými funkcemi. Obvykle se spustí při zapnutí zařízení a zobrazí obrazovku s ikonami nebo dlaždicemi, které zobrazují informace a umožňují přístup k aplikacím. Mobilní operační systémy také spravují mobilní a bezdrátové síťové připojení a přístup k telefonu.

Příkladem mobilních operačních systémů jsou Apple iOS a Google Android. [13]

2.1 Android

Android je software pro mobilní telefony, tablety a rostoucí škálu zařízení, která zahrnují vše od nositelné elektroniky až po zábavu v automobilu. [14]

Android funguje jako překladatel mezi uživatelem a jejich zařízením, umožňující interakci pomocí dotyků, hovorů nebo gest. Software je vyvíjen ve spolupráci se společností Google, která každoročně vydává velké aktualizace. Mezi výrobce, kteří používají Android, patří Samsung, Huawei, Sony, Lenovo, Asus, LG a další. Google udržuje základní prostředí, přidává nové funkce a aktualizuje staré, a zajišťuje dodržování principů open-source. [14]

Historie Androidu

V říjnu 2003 začala historie systému Android s jeho založením v Palo Altu v Kalifornii čtyřmi zakladateli: Rich Miner, Nick Sears, Chris White a Andy Rubin. Původně zamýšleli vytvořit *„chytřejší mobilní zařízení, která si budou lépe uvědomovat polohu a preference svého majitele“*. Později, v roce 2013, Andy Rubin prozradil, že Android OS měl původně vylepšit operační systémy digitálních fotoaparátů, ale společnost se později přeorientovala na mobilní telefony, když se trh s digitálními fotoaparáty začal snižovat.

V roce 2005 společnost Google koupila Android, což změnilo kurz událostí. Google a tým Androidu se rozhodli použít Linux jako základ pro operační systém, což umožnilo nabízet ho výrobcům mobilních telefonů třetích stran zdarma. Tato strategie umožnila Googlu profitovat z poskytování dalších služeb a aplikací. [15]

Maskotem Androidu je zelený robot.

Architektura Android

Android je open source softwarový balík založený na Linuxu, který byl vytvořen pro širokou škálu zařízení a formátů. V *Příloze C* jsou hlavní součásti platformy Android. [16]

2.1.1 Jádro Androidu

Základem platformy Android je linuxové jádro. Například běhové prostředí Androidu (ART) se spoléhá na jádro Linuxu, pokud jde o základní funkce, jako je vlákno a nízkoúrovňová správa paměti. [16]

Použití linuxového jádra umožňuje systému Android využívat klíčové bezpečnostní funkce a výrobcům zařízení vyvíjet hardwarové ovladače pro známé jádro. [16]

Hardwarová abstrakční vrstva (HAL)

HAL je zkratka pro Hardwarovou Abstraktní Vrstvu. Poskytuje standardní rozhraní pro přístup k hardwarovým funkcím zařízení skrze Java API. Každý modul implementuje rozhraní pro specifický typ hardwaru, jako je kamera nebo Bluetooth. Systém Android načte příslušný modul knihovny pro danou hardwarovou komponentu při voláních API. [16]

Běhové prostředí systému Android (Android Runtime)

Na zařízeních s Androidem 5.0 nebo vyšším běží každá aplikace ve vlastním procesu a s vlastní instancí běhového prostředí ART. ART provádí soubory ve formátu DEX, optimalizované pro malou paměť. Nástroje jako d8 kompilují Java zdrojový kód do DEX bajtového kódu, který lze spustit na Androidu. Systém Android obsahuje základní knihovny běhového prostředí, včetně funkcí Java 8. [16]

Nativní knihovny C/C++

Mnoho klíčových součástí a služeb systému Android, například ART a HAL, je vytvořeno z nativního kódu, který vyžaduje nativní knihovny napsané v jazycích C a C++. Platforma Android poskytuje framework Java API, který aplikacím zpřístupňuje funkce některých těchto nativních knihoven. [16]

Rámec Java API (Java API framework)

Celá sada funkcí operačního systému Android je k dispozici prostřednictvím rozhraní API napsaných v jazyce Java. Tato rozhraní API tvoří stavební kameny, které potřebujete k vytváření aplikací pro systém Android, protože zjednodušují opakované použití základních modulárních systémových komponent a služeb, mezi které patří např. následující:

- Bohatý a rozšiřitelný systém zobrazení, který můžete použít k vytvoření uživatelského rozhraní aplikace, včetně seznamů, mřížek, textových polí, tlačítek, a dokonce i vloženého webového prohlížeče.
- Správce zdrojů, který poskytuje přístup k nekódovým zdrojům, jako jsou lokalizované řetězce, grafika a soubory rozvržení.
- Správce oznámení, který umožňuje všem aplikacím zobrazovat vlastní upozornění ve stavovém řádku.
- Správce aktivit, který spravuje životní cyklus aplikací a poskytuje společný navigační zpětný zásobník.
- Zprostředkovatelé obsahu, kteří umožňují aplikacím přístup k datům z jiných aplikací, například z aplikace Kontakty, nebo sdílení vlastních dat.
- Vývojáři mají plný přístup ke stejným rozhraním API frameworku, která používají systémové aplikace systému Android. [16]

Systémové aplikace

Systém Android obsahuje základní aplikace pro e-mail, SMS, kalendáře, prohlížení internetu, kontakty a další. Tyto aplikace mají stejné postavení jako aplikace třetích stran a mohou být nahrazeny uživatelem. Výjimkou je například aplikace Nastavení systému. Systémové aplikace fungují jako uživatelské aplikace a poskytují klíčové funkce, ke kterým mají vývojáři přístup pro své aplikace. Můžeme například použít již nainstalovanou aplikaci SMS pro doručování zpráv místo vytváření vlastní funkcionality. [16]

2.1.2 Android jako open-source

Android je postaven na linuxovém jádře a struktuře GNU/Linux. Android je open-source projekt řízený Googlem pod názvem *AOSP* (Android Open Source Project). Tato platforma je volně dostupná pro komerční i nekomerční účely, což se výrazně liší od uzavřených systémů jako iOS, macOS a Microsoft Windows. [14]

Open source software, jako je Android, umožňuje bezplatné použití bez omezení autorských práv. Android vychází z linuxového jádra, což přispívá k jeho popularitě, jelikož je

volně dostupný pro komerční i nekomerční účely. Nicméně, ne všechny součásti systému Android jsou open source; části jako *Služby Google Play* jsou uzavřené. Tato hybridní povaha Androidu umožňuje jeho široké použití, ale Google má stále kontrolu nad některými službami. Původně navržen jako operační systém pro fotoaparáty, byl Android přepracován pro smartphony poté, co Google v roce 2005 koupil firmu Android. Přestože původně bojoval s konkurencí, přizpůsobil se vývoji trhu a stal se hlavním hráčem díky inovacím ve vývoji mobilních telefonů. Na níže přiloženém obrázku je ukázka prvního mobilního telefonu s OS Andorid, HTC Dream. [17]

Obrázek 3, HTC Dream

Zdroj: [17]

Prostředí Android

Android a iOS mají mnoho podobností, ale i rozdílů. Oba systémy umožňují spouštění aplikací, připojení k Wi-Fi a další základní funkce, ale jejich vzhled a možnosti se mohou lišit. Android je open-source, což umožňuje výrobcům přizpůsobit software podle svých potřeb. Existuje mnoho uživatelských rozhraní od různých výrobců, jako je TouchWiz od Samsungu nebo Xperia od Sony. Android nabízí širokou škálu zařízení od různých výrobců, což znamená větší variabilitu a možnost výběru. To dává uživatelům větší flexibilitu a možnost přizpůsobit si své zařízení podle svých potřeb. [14]

Verze Androidu

„Nejčistší“ verze Androidu je často označována jako „standardní Android“ či „vanilla“ a komunita Androidu ji často preferuje – je to původní software, jak Google zamýšlel. [14]

Každá nová verze Androidu dostala kódové jméno na základě po sobě jdoucích písmen abecedy. Dnes už tomu tak není a jsou číselně označeny. Nejnovější verze je známá jako

Android 15, které je už dostupná, ale ne v celé verzi. Předchozí verze zahrnovaly Android 14, 13, 12, 11, 10. Před Android 10 byly písemné označení verzí po sladkostech. Byly to pro příklad Pie, Oreo, Nougat, Marshmallow, Lollipop, KitKat, Jelly Bean, Gingerbread atd. Po verzi Androidu Pie měla mít další nová verze označení Q, Android Q, ale pro lepší přehlednost se Google rozhodl zrušit písemné značení a přešel na číselné. Android 10 proto, že poslední „písemně“ označená verze byla celým jménem značena jako Android 9.0 Pie, proto se začalo od 10. [18]

Výhody a nevýhody Androidu

Android nabízí širokou škálu produktů od levných telefonů po vlajkové lodě, zatímco u Applu máme na výběr pouze iPhone. Android je snadno přizpůsobitelný vzhledem i funkcemi. Aplikace v Google Play nepodléhají tak přísné kontrole jako u Applu, což může být jak pozitivum, tak negativum. [14]

Naopak mínusové body získává Android za získávání aktualizací. V mnoha případech se zdá, že se výrobci nestarají o poskytování aktualizací softwaru pro zařízení, která nám již prodali. I když poskytují aktualizace, dávají si na čas. To je důvod, proč někteří uvažují o *rootingu*: aktualizace si můžeme stáhnout sami a použít je, místo abychom čekali, až se k tomu výrobce dostane. Aktualizace totiž prochází několika „příčkami“. Prochází samotným Googlem, s daným výrobcem telefonu a následně operátorem, aby se vyladily potřebné sounáležitosti v aktualizacích. Tento dlouhý řetězec událostí je jedním z hlavních důvodů, proč telefony Android nevidí aktualizace tak často nebo tak dlouho jako zařízení iOS. U iOS si Apple vše řídí sám a operátoři zasahují do tohoto procesu minimálně. Jsou ale některé mobilní telefony, které se k tomuto konceptu blíží a těmi jsou telefony Pixel od společnosti Google. [17]

Pro distribuci aplikací s Androidem slouží digitální tržiště Google Play. [14]

2.2 iOS

iOS je multiplatformní operační systém vytvořený a vyvinutý společností Apple Inc. pro iPhone. Tato platforma dříve podporovala také zařízení iPad (do roku 2019) a iPod Touch (do roku 2022). Je založen na operačním systému macOS, který Apple používá pro své počítače Macintosh od roku 2001. Systém iOS byl inspirací pro watchOS, tvOS a iPadOS, které podporují další zařízení Apple. V roce 2022 zůstal iOS druhým nejčastěji instalovaným mobilním operačním systémem na světě. [19]

Apple začal používat označení „i“ napříč svou řadou produktů od roku 1998 s počítačem iMac. Tato předpona byla vysvětlena jako zkratka pro *individual, instruct, inform* a *inspire*. Postupem času se rozšířila na iPhone, iPod Touch a iPad, které běží na operačním systému iOS. S vydáním iOS 13 v roce 2019 byl operační systém pro tablety přejmenován na iPadOS, aby lépe reflektoval rostoucí seznam funkcí specifických pro tablety, jako je podpora doků na domovské obrazovce a režim obraz v obraze. Tím se také zajistilo, že verze iOS a iPadOS budou mít shodné číslo vydání [20]

Mezi hlavní součásti systému iOS patří uživatelské rozhraní a navigace, správa aplikací a App Store, jakož i bezpečnostní funkce a ochrana soukromí. Tyto hlavní součásti spolupracují na vytvoření bezproblémového a příjemného uživatelského prostředí v mobilních zařízeních Apple. [21]

Dále je velkou předností tohoto mobilního operačního systému jeho bezpečnost. Liší se od většiny ostatních operačních systémů tím, že každá aplikace má svůj vlastní ochranný „obal“, který brání ostatním aplikacím, aby s ním manipulovaly. Díky této konstrukci je téměř nemožné, aby virus infikoval aplikace v operačním systému iOS, ačkoli existují i jiné formy malwaru. Ochranný plášť kolem aplikací představuje také omezení, protože zabraňuje aplikacím ve vzájemné přímé komunikaci. [22]

Fungování systému iOS

iOS je vrstvený systém, což znamená, že se skládá z několika vrstev softwarových komponent, které společně zajišťují funkčnost a výkon systému. Vrstvy systému iOS jsou následující:

- **Jádro operačního systému:** Je to nejnižší vrstva systému iOS a poskytuje základní služby a rámce systému, jako je jádro, ovladače, zabezpečení, šifrování, sítě a správa napájení. [23]
- **Základní služby:** Tato vrstva je nad jádrem OS a poskytuje základní služby a rámce systému, jako je souborový systém, databáze, iCloud, Core Foundation, Core Location, Core Motion a další. [23]
- **Média:** Vrstva Media poskytuje služby a rámce pro zvuk, video, grafiku a animace systému, například Audio Toolbox, AV Foundation, Core Audio, Core Graphics, Core Image, Core Text, Core Video a další. [23]
- **Cocoa touch:** Je nejvyšší vrstvou systému iOS a poskytuje služby a rámce pro uživatelské rozhraní a interakci systému, například UIKit, Foundation, Event Kit, Game Kit, Map Kit, Message UI a další. [23]

Verze iOS

Pro přehled verzí si některé zmíníme. První verzí byl OS iPhone OS 1 vydán 29. června 2007. Označení iPhoneOS končí v roce 2009, kdy firma předtavila verzi iPhoneOS 3. Od této doby se operační systém značí iOS a příslušná číslovka, konkrétně verzí 4, kdy iOS 4 byl vydán v červnu 2010. Zkruba po každém uplynutém roce firma Apple představí novou verzi iOS, kdy k dnešnímu datu iPhony používají verzi OS s názvem iOS 17 vydanou v září 2023. [24]

2.3 Mobilní aplikace

Mobilní aplikace jsou softwarové aplikace navržené speciálně pro použití na chytrých telefonech a tabletech. Jejich velikost je obvykle menší než u desktopových aplikací, což odpovídá omezeným prostředkům mobilních zařízení a optimalizaci pro dotykové obrazovky. Mobilní aplikace jsou nedílnou součástí každodenního života a poskytují širokou škálu funkcí od sociálních médií a zábavy po produktivitu a podnikání. Pro firmy se stávají důležitým způsobem komunikace se zákazníky a zaměstnanci, nabízejíce jim pohodlný, personalizovaný a bezpečný přístup k informacím a úkolům na cestách. Mobilní aplikace jsou vytvářeny v různých programovacích jazycích a frameworkách a lze je stahovat z obchodů s aplikacemi, jako je Apple App Store nebo Google Play. Jsou navrženy s ohledem na požadavky, omezení a možnosti mobilních zařízení, a mohou poskytovat širokou škálu funkcí a služeb od her a sociálních médií po bankovnictví a e-mailové klienty. [25]

V *Příloze D* je na obrázku zachycen proces vývoje mobilní aplikace. Od samotné strategie až po nasazení aplikace.

Mobilní aplikace jsou určeny pro konkrétní mobilní operační systémy, jako jsou iOS, Android a Windows Phone. Když je mobilní aplikace stažena a nainstalována do zařízení, uloží se do paměti zařízení a spustí se pomocí operačního systému zařízení. [25]

Při otevření, mobilní aplikace komunikuje s operačním systémem a dalšími vestavěnými komponentami zařízení, jako jsou fotoaparát, GPS a internetové připojení. Tyto informace využívá k poskytování specifických funkcí a služeb. Mezi výhody patří snadná instalace, personalizace a off-line přístup. Nevýhody zahrnují omezenou funkčnost oproti desktopovým aplikacím, problémy s kompatibilitou mezi různými operačními systémy a potenciální bezpečnostní rizika. Také může být obtížné aktualizovat aplikace na nejnovější verze. [25]

3 Vývojové prostředí

IDE je software, který kombinuje různé vývojářské nástroje do jedné aplikace s grafickým uživatelským rozhraním (GUI), včetně editoru kódu, kompilátoru a ladicího programu. Tyto nástroje zahrnují funkce jako editace, sestavování, testování a balení softwaru, což pomáhá zvýšit produktivitu vývojářů. I když IDE není nutné k programování, poskytuje řadu užitečných funkcí, které zjednodušují proces vývoje softwaru. [26]

IDE poskytují širokou škálu funkcí, včetně:

- **Editor:** Pomáhá při psaní kódu tím, že zvýrazňuje syntaxi, nabízí automatické dokončení a kontroluje chyby.
- **Překladač:** Interpretuje kód do strojově specifického kódu, který lze spustit v různých operačních systémech.
- **Ladicí program:** Pomáhá testovat a ladit aplikace a upozorňuje na chyby.
- Vestavěný terminál: Umožňuje interakci s operačním systémem počítače přímo v IDE.
- **Řízení verzí:** Pomáhá sledovat a spravovat změny softwarového kódu.
- **Úryvky kódu:** Umožňují opakované použití často používaných kódových úseků.
- **Rozšíření a zásuvné moduly:** Rozšiřují funkce IDE podle potřeb konkrétního programovacího jazyka.
- **Navigace v kódu:** Poskytuje nástroje pro snadnější procházení a analýzu kódu. [26]

Díky tomu, že IDE poskytují jednotné prostředí pro správu všech aspektů vývojového procesu, mohou pomoci zvýšit produktivitu vývojářů, kvalitu kódu a celkový zážitek z vývoje. [26]

3.1 Visual Studio

Visual Studio je integrované vývojové prostředí (IDE) vyvinuté společností Microsoft pro širokou škálu aplikací, včetně desktopových, webových, mobilních a cloudových aplikací. Podporuje mnoho programovacích jazyků, jako je C#, C++, VB (Visual Basic), Python, JavaScript a další. Tento výkonný nástroj nabízí kompletní vývojový cyklus v jednom prostředí pro psaní, úpravy, ladění a sestavování kódu, a to včetně různých funkcí jako překladače, nástroje pro doplňování kódu, správu zdrojových kódů a rozšíření. Visual Studio je dostupné pro systém Windows i MacOS. [27]

3.2 XCode

Xcode je oficiální IDE od Applu, vydáno v roce 2003 pro vývoj softwaru na Apple platformách. Xcode je komplexním nástrojem pro vytváření aplikací pro iPhone, iPad, Mac, Apple Watch a Apple TV, hlavně pro iOS. Toto IDE je jediný oficiální způsob tvorby aplikací pro App Store, vhodný pro začátečníky i zkušené vývojáře. Obsahuje vše potřebné pro psaní, kompilaci a ladění aplikací, s možností snadného odeslání do App Store. Xcode nabízí efektivní nástroje pro rychlý vývoj aplikací a má uživatelsky přívětivé prostředí, což zmenšuje obtíže začínajících vývojářů. [29]

3.3 Android Studio

Android Studio je oficiální IDE pro vývoj aplikací pro Android. Jedná se o výkonný nástroj s kompletními funkcemi od psaní kódu po testování a nasazení aplikací na platformě Android. [30]

Android Studio využívá systém sestavování založený na Gradle, emulátor Androidu, šablony kódu a integraci s GitHubem pro podporu vývoje aplikací pro operační systém Android. Android Studio využívá funkci „Apply Changes“ k odesílání změn kódu a zdrojů do běžící aplikace. Editor kódu nabízí vývojářům doplňování, lámání a analýzu kódu. Výsledné aplikace jsou kompilovány do formátu APK pro odeslání do obchodu Google Play. [31]

První stabilní verze Android Studia byla vydána v prosinci 2014, nahrazuje Eclipse Android Development Tools (ADT) jako hlavní IDE pro vývoj aplikací pro Android. Je dostupný pro desktopové platformy macOS, Windows a Linux a lze ho stáhnout přímo od společnosti Google. [31]

3.4 Android Emulator

Android SDK poskytuje virtuální emulátor mobilního zařízení, který umožňuje spouštění, ladění a testování aplikací Androidu přímo na počítači. Emulátor je podobný fyzickému zařízení s Androidem a obsahuje navigační ovládací prvky a přístupnost dotykové obrazovky. Lze nastavit různé konfigurace zařízení, včetně verzí systému Android a úrovní API. Emulátor umožňuje provádět všechny funkce jako skutečné zařízení a nabízí možnost dynamického binárního překladu strojového kódu zařízení na architekturu procesoru počítače. [32]

Výhody emulátoru Android

Virtuální zařízení umožňují rychlejší přenos souborů přetažením (drag and drop). Emulátor Androidu umožňuje práci s fyzickými senzory a nabízí širokou škálu funkcí, jako jsou hry, prohlížení internetu a úprava nastavení. Umožňuje vývojářům vybrat libovolnou verzi systému Android pro testování aplikací a zkoušení různých testovacích scénářů. [32]

Nevýhody emulátoru Android

Emulátor Androidu bývá pomalejší než skutečná fyzická zařízení a nedokáže zaznamenat vlastnosti jako rychlost baterie, polohu nebo hardwarové aktivity. Testování na emulátoru není tak přesné jako na skutečném zařízení a může chybět identifikace problémů se sítěmi nebo oznámeními. Emulátor může při nedostatku místa na disku selhat. [32]

4 React Native

Nyní se podíváme na konkrétní programovací jazyk pro vývoj mobilních aplikací, který je možno použít v operačních systémech s macOS či MS Windows nebo v mobilních operačních systémech iOS či Android. Tím je *React Native*.

React Native je JavaScriptový framework pro vytváření mobilních aplikací pro iOS a Android, který využívá React pro tvorbu uživatelských rozhraní. Sdílení kódu mezi platformami usnadňuje vývoj pro oba operační systémy současně. Aplikace se píší v JSX a JavaScriptu a díky „mostu“ s nativním vykreslovacím rozhraním API pro iOS a Android vypadají a chovají se jako skutečné mobilní aplikace. React Native umožňuje přístup k funkcím platformy, jako je fotoaparát nebo poloha uživatele, a má potenciál rozšířit se i na další platformy. [33]

Výhody React Native

React Native se odlišuje od jiných metod vývoje multiplatformních aplikací, jako je Cordova nebo Ionic, tím, že skutečně využívá nativních vykreslovacích rozhraní API hostitelských platforem. To znamená, že místo použití webových pohledů jako ostatní frameworky, React Native převádí značení na skutečné nativní prvky uživatelského rozhraní. Tento přístup zajišťuje lepší výkon a přístup k nativním prvkům uživatelského rozhraní. [33]

Díky oddělené práci od hlavního vlákna uživatelského rozhraní si aplikace zachovává vysoký výkon. Cyklus aktualizace v React Native funguje podobně jako v Reactu, což znamená, že při změně stavu nebo rekvizit se znovu vykreslí příslušné pohledy. Hlavním rozdílem mezi React Native a Reactem pro web je využití nativních knihoven uživatelského rozhraní místo HTML a CSS. [33]

React.js vs. React Native

React je open-source knihovna JavaScriptu, která slouží k vytváření rychlých a spolehlivých uživatelských rozhraní pro webové aplikace, s důrazem na opakovaně použitelné komponenty. Společnost Facebook představila React v roce 2011 jako reakci na problémy se škálovatelností na své platformě. Tento framework pomohl vyřešit problémy s výkonem a umožnil rychlejší aktualizace uživatelského rozhraní, což vedlo k jeho širokému využití v aplikacích Facebooku, včetně Instagramu. Vývojem Reactu se později přirozeně vyvinul React Native, mobilní framework, který umožňuje vytváření téměř nativních mobilních aplikací pomocí JavaScriptu. I když oba frameworky úzce souvisejí, slouží různým účelům:

React se používá pro vývoj webových aplikací, zatímco React Native slouží k vývoji mobilních aplikací a nepoužívá HTML. [34]

4.1 Node.js

Node.js vyvinul Ryan Dhal v roce 2009 a jedná se o open-source a multiplatformní běhové prostředí JavaScriptu. [35]

Node.js využívá mimo prohlížeče JavaScriptový engine V8, jádro prohlížeče Google Chrome, což mu dodává vysokou výkonnost. Aplikace v Node.js běží v jediném procesu, kde není potřeba vytvářet nová vlákna pro každý požadavek. [36]

Node.js poskytuje sadu asynchronních I/O příkazů, které umožňují provádět operace jako čtení ze sítě nebo přístup k databázi bez blokování kódu JavaScriptu. Tím umožňuje Node.js zpracovávat tisíce současných připojení s jedním serverem bez zbytečné zátěže spojené s řízením vláken. Tato schopnost přitahuje vývojáře frontendu, kteří jsou již obeznámeni s JavaScriptem, a umožňuje jim psát kód jak pro klienta, tak i pro server bez nutnosti učení nového jazyka. [36]

4.2 NPM

NPM (Node Package Manager) je největší registr softwaru na světě. Vývojáři open-source ze všech kontinentů používají npm ke sdílení a půjčování balíčků a mnoho organizací používá npm také ke správě soukromého vývoje. [37]

Systém npm se skládá ze tří různých součástí:

- Webové stránky,
- rozhraní příkazového řádku (CLI),
- registr. [37]

Webové stránky poskytují rozhraní pro vyhledávání balíčků, nastavování profilů a správu dalších aspektů práce s npm, včetně možnosti vytvoření organizace pro správu přístupu k veřejným nebo soukromým balíčkům. CLI rozhraní umožňuje vývojářům komunikovat s npm přímo z terminálu. Registr je rozsáhlá veřejná databáze softwaru JavaScript a obsahuje metainformace o balíčcích. [37]

Všechny balíčky npm jsou definovány v souborech s názvem *package.json*. Obsah souboru package.json musí být zapsán ve formátu JSON. V definičním souboru musí být přítomna alespoň dvě pole: název a verze. [38]

4.3 Expo CLI

Expo je platforma a framework pro vytváření univerzálních aplikací React, umožňující vývoj, iteraci, testování a nasazení aplikací pro Android, web a iOS pomocí stejné kódové základny v TypeScriptu nebo JavaScriptu. Expo CLI poskytuje vrstvu nad React Native, usnadňující vývojářům plnění úkolů a eliminuje nutnost propojovat externí knihovny pro iOS a Android, přičemž zároveň provádí složitou práci v backendu. Expo je uznáno jako nástroj pro řízený pracovní postup vytváření React nativních aplikací. Expo CLI však nenabízí úplnou kontrolu nad systémy iOS a Android zvlášť, jak tomu je u holého rozhraní React Native CLI. [39]

React Native CLI

Oproti Expu je React Native CLI. React Native CLI je tradiční přístup k vytváření aplikací React Native. Poskytuje maximální flexibilitu a kontrolu nad projektem a nastavením. [40]

Výhody React Native CLI:

- Plná podpora nativních modulů: React Native CLI umožňuje přístup a integraci nativních modulů, které jsou k dispozici v ekosystému React Native. [40]
- Možnosti přizpůsobení: Máte plnou kontrolu nad konfigurací sestavení, což vám umožní vyladit výkon a velikost aplikace. [40]

Nevýhody React Native CLI:

- Strmější křivka učení: React Native CLI vyžaduje více nastavení a konfigurace. [40]
- Delší nastavení pro vývoj: Nastavení projektu React Native CLI zahrnuje instalaci a konfiguraci nativních závislostí, což může zabrat mnohem více času ve srovnání se začátkem práce s Expo. [40]
- Ruční proces sestavování: React Native CLI vyžaduje ruční konfiguraci nástrojů pro sestavení, což může být časově náročnější a složitější. [40]
- Nativní fyzické zařízení: React Native CLI vyžaduje, abyste pro testování aplikace pro Android nebo iOS měli fyzické zařízení specifické pro danou platformu. [40]

5 Autentizace

Slovo autentizace pochází z řeckého slova authentikos a latinského authenticus, což znamená pravý, původní, hodnověrný. Podle vyhlášky Národního bezpečnostního úřadu č. 56/1999 Sb. je autentizace definována jako proces ověření identity subjektu s požadovanou mírou záruky (§ 2 písm. f). [41]

Důležité je zdůraznit, že identita subjektu může zahrnovat nejen jeho osobní identitu (př. jméno Jan Novotný), ale také skupinovou příslušnost (př. jeden z administrátorů), schopnost (př. schopnost řídit motorové vozidlo) nebo negaci těchto vlastností (př. „ne“, jsem hledaná osoba). [41]

5.1 Autentizace a její základní dělení

Proces autentizace můžeme rozdělit do dvou základních fází:

1. Registrace,
2. verifikace. [41]

Fáze registrace spočívá v uložení vzoru identifikačních znaků subjektu, takzvaného etalonu, do databáze autentizačního systému. Fáze verifikace spočívá v předložení identifikačních znaků autentizovaným subjektem, porovnání těchto předložených identifikačních znaků s etalonem uloženým v databázi a učinění rozhodnutí, zda tento důkaz identifikace přijmout, či nikoliv. [41]

Podle druhu použitých identifikačních znaků rozlišujeme následující základní druhy autentizace:

- Znalostní autentizace (něco znát),
- autentizace prostřednictvím autentizačního předmětu (něco mít),
- biometrická autentizace (něčím být),
- vícefaktorová autentizace. [41]

Znalostní autentizace: Základem je znalost, kterou má daný subjekt, jako jsou hesla, jednorázová hesla a kontrolní otázky. [41]

Autentizace prostřednictvím autentizačního předmětu: Subjekt vlastní jedinečný předmět, jako jsou klíče nebo magnetické karty. [41]

Biometrická autentizace: Využívá jedinečné vlastnosti lidského těla, například otisk prstu, hlas, obličej nebo dynamiku psaní. [41]

5.2 Biometrická autentizace

Biometrická autentizace využívá jedinečné charakteristiky lidského těla, které nosí každý člověk stále s sebou. Tyto vlastnosti nelze zapomenou, ztratit, půjčit. Vlastnosti lidského těla jsou však často stochastické, mohou se během života měnit. Charakteristiky lidského těla použité při biometrické autentizaci mohou mít fyziologický nebo behaviorální charakter. [41]

Fyziologické charakteristiky lidského těla souvisí s jeho parametry. Příkladem takových charakteristik jsou – otisky prstu (vzory na prstu), otisky dlaní, geometrie ruky, vzory cév na ruce, znaky v obličeji, vzory na sítnici a vzory v duhovce. [41]

Kvantitativní ukazatele autentizace

Autentizovaný subjekt může být buď oprávněný uživatel nebo neoprávněný narušitel. V ideálním případě jsou identifikační znaky předložené oprávněným uživatelem jako důkaz jeho identity přijaty (správné přijetí) a identifikační znaky předložené neoprávněným narušitelem jako důkaz jeho identity odmítnuty (správné odmítnutí). Avšak rozhodnutí autentizace může být chybné. Pokud jsou identifikační znaky předložené oprávněným uživatelem odmítnuty, dochází k chybnému odmítnutí. Pokud jsou identifikační znaky předložené neoprávněným narušitelem přijaty, dochází k chybnému přijetí. [41]

Tabulka 1, rozhodovací situace

Rozhodnutí			
Přijmout	**Odmítnout**		
Žádoucí stav	Chybné odmítnutí	**Oprávněný uživatel**	**Situace**
Chybné přijetí	Žádoucí stav	**Neoprávněný narušitel**	

Zdroj: Vlastní

Mezi nejvýznamnější patří ukazatel *False accept rate* a ukazatel *False reject rate*. False accept rate (FAR) představuje podmíněnou pravděpodobnost chybného přijetí autentizace neoprávněným narušitelem. False reject rate (FRR) představuje podmíněnou pravděpodobnost chybného odmítnutí autentizace oprávněného uživatele. [41]

Poznámka:

Části psaného textu byly seskupeny pomocí umělé inteligence. K tomuto úkonu byl použit program ChatGPT. [42]

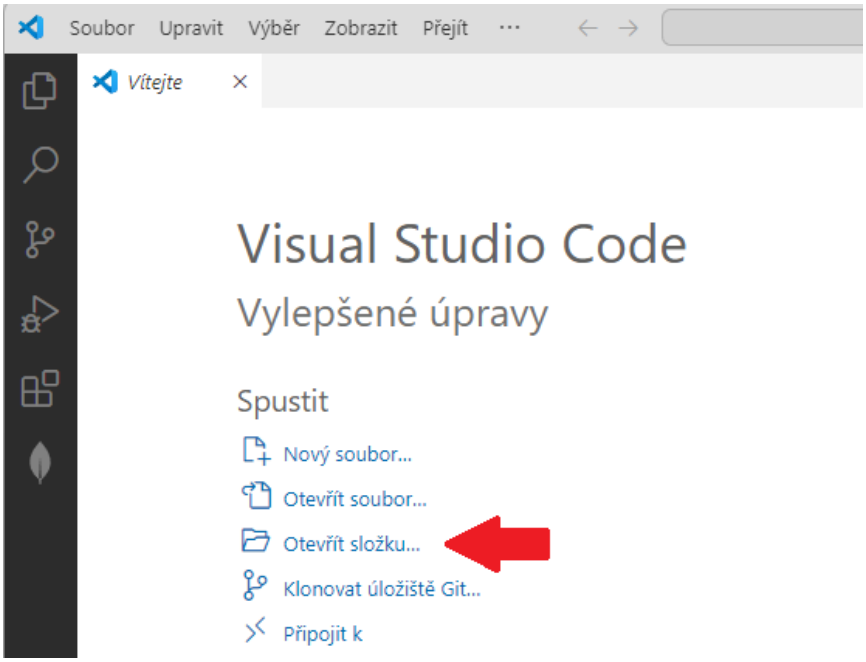
6 Příprava pro vývoj aplikace

Pro úspěšné naprogramování aplikace si potřebujeme nainstalovat vybrané soubory. Jedním z nich je *Node.js*. Ten si můžeme nainstalovat v prohlížeči od daného vývojáře. Přítomnost Node.js si můžeme ověřit díky příkazu *node -v*, který napíšeme do příkazové řádky Windows. Ta nám ukáže aktuální verzi, a tudíž i přítomnost Node.js v našem počítači.

Jako další si potřebujeme naistalovat *NPM* (Node Package Manager). Toho docílíme pomocí příkazu *npm install -g*. „G“ (-g) znamená globálně, a tudíž nemusíme pro další aplikace používající NPM toto rozšíření nadále stahovat. NPM je poté přístupné pro všechny aplikace.

6.1 Práce v IDE Visual Studio

Pro vývoj mobilní aplikace budeme potřebovat dva programy. Jedním je *IDE Visual Studio* a *Android Studio* využívající *Android Emulator*. V prostředí OS Windows si založíme složku, která nám bude sloužit pro projekt jako úložiště a vhodně ji pojmenujeme. Tuto nově vytvořenou složku otevřeme ve Visual Studiu. Na uvítací obrazovce Visual Studia toho můžeme docílit dvěma způsoby, buď v horní liště přejdeme do nabídky *Soubor -> Otevřít složku* a následně vybereme umístění složky, kterou jsme vytvořili v předešlém kroku nebo o něco kratší přístup k nově vytvořené složce, a to pomocí volby *Otevřít složku…* dostupné přímo na hlavní obrazovce viz obrázek s červenou šipkou. Poté po vyzvání zvolíme umístění dané složky, kterou jsme vytvořili v předešlých krocích.



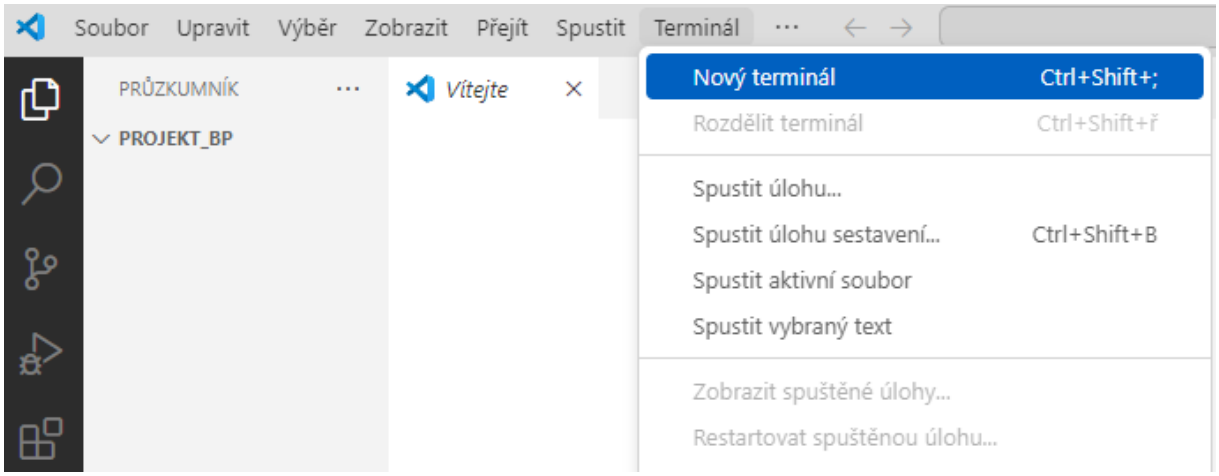
Obrázek 4, otevření složky v IDE VS

Zdroj: Vlastní

6.2 Terminál

Důležitou součástí při práci je terminál. Je to vlastně příkazový řádek obdobně jako ve Windows. Tento příkazový řádek je vestavěný přímo v aplikaci a má stejné funkce jako CMD ve Windows. Ve Visual Studiu je vestavěný, tudíž má své místo v aplikaci a nemusíme překrývat jednotlivá okna či zobrazovat více oken vedle sebe.

Terminál si spustíme v horní liště aplikace v záložce *Terminál*, poté zvolíme *Nový terminál* viz obrázek níže.



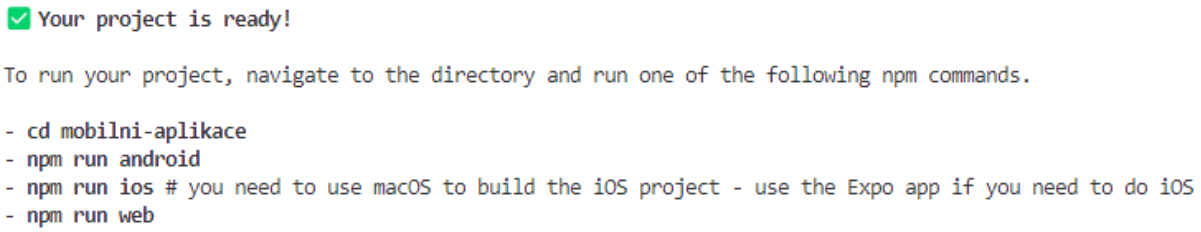
Obrázek 5, spuštění terminálu

Zdroj: Vlastní

Výhodou terminálu ve Visual Studiu je i fakt, že po jeho otevření se nám zobrazuje přímo cesta k námi vytvořené složce. V příkazovém řádku Windows bychom museli ručně dojít k této složce v závislosti na jejím umístění v našem počítači. Pro tento úkon je potřeba znát jednotlivé příkazy příkazového řádku, jako například *cd Složka1* (change directory; přesměrování do složky Složka1), *cd ..* (přesměrování o úroveň výše v adresářové struktuře), *dir* (zobrazení obsahu adresáře), *mkdir Složka2* (vytvoření složky s názvem Složka2) či kombinace typu *cd D:* (přechod na disk D:\). Z předešlého výčtu je zřejmé, že je i další způsob, jak vytvořit složku i za pomocí příkazového řádku rovnou ve Visual Studiu.

Je třeba mít na paměti, že samotný příkazový řádek můžeme najít pod odlišným názvem, například macOS oproti Windows. macOS používá *Terminál*, Windows používá *Příkazový řádek (CMD)*, ale princip funkce Terminálu a Příkazového řádku je stejný. Dále je třeba zmínit, že jednotlivé příkazy se liší skrze různými operačními systémy, například v OS macOS můžeme najít příkaz *dir* jako *ls (list)* nebo příkaz *ipconfig* pro zobrazení síťového rozhraní ve Windows jako *ifconfig* v macOS či Linuxu. Příkazy pro Linux a macOS jsou podobné, ne-li stejné, protože oba OS vycházejí z UNIXového operačního systému viz teoretická část.

Nyní nainstalujeme do zvolené složky aplikaci React Native s Expo rozšířením. Do příkazového řádku napíšeme následující příkaz – *npx create-expo-app mobilni-aplikace*. Tento příkaz nám založí React Native aplikaci s využitím Expo s názvem *mobilni-aplikace*. Příkaz nám nainstaluje Expo lokálně, to znamená, že Expo je dostupné pouze pro daný projekt. Po dokončené instalaci se pomocí terminálu ve VS Code přesuneme do nově vytvořené složky s projektem, viz obrázek níže s výstupem z terminálu.



Obrázek 6, informace o úspěšném nainstalování aplikace

Zdroj: Vlastní

6.3 Spuštění aplikace

Abychom mohli aplikaci spustit, přesuneme se do nově vytvořené složky mobilni-aplikace. Toho docílíme příkazem *cd mobilni-aplikace*. Z tohoto místa vždy budeme spouštět a manipulovat s aplikací.

Aplikaci spustíme příkazem *npx expo start*. Zobrazí se nám informace viz níže na obrázku.

Obrázek 7, výstup možností spuštění

Zdroj: Vlastní

6.4 Nastavení SW Android Emulator

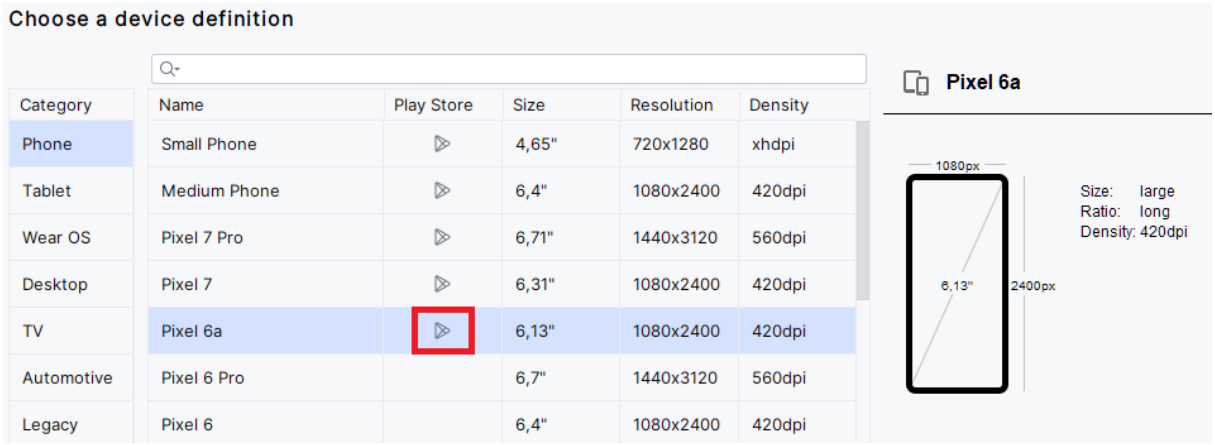
Aplikaci můžeme spustit na zařízení Android, jedná se o virtuální zařízení, ale v tento moment zařízení není dostupné. Virtuální mobilní zařízení si vytvoříme díky programu Android Studio.

Aplikace se dá spustit i na reálném zařízení za pomocí aplikace *Expo Go* dostupné v ochodu Google Play pro zařízení s OS Android. Po stažení aplikace si naskenujeme vygenerovaný QR kód a aplikace se nám vykreslí do zařízení. Pro tuto práci ale budeme využívat virtuální zařízení.

Můžeme si všimnout na obrázku č. zašedlé položky > *Press | open web*, aplikace se dá spustit i v běžném webovém prostředí, ale k tomu je zapotřebí doinstalovat potřebné závislosti a to hlavně závislosti *react-dom*, dostupné z dokumentace https://docs.expo.dev, kde můžeme najít vše potřebné týkající se vývoje za použití technologie Expo.

Nyní si potřebujeme vytvořit a nastavit virtuální mobilní zařízení. Po otevření aplikace Android Studio se nám zobrazí hlavní obrazovka a vybereme možnost z rozbalovacího menu *More Actions* položku *Virtual Device Manager*, viz *Příloha E*.

V levé horní části klikneme na tlačítko „+“ a zvolíme si zařízení z nabídky. Z obrázku níže je patrné, že si můžeme nastavit mnoho virtuálních zařízení, které obsahují OS Android, jsou to tablet či televize aj.

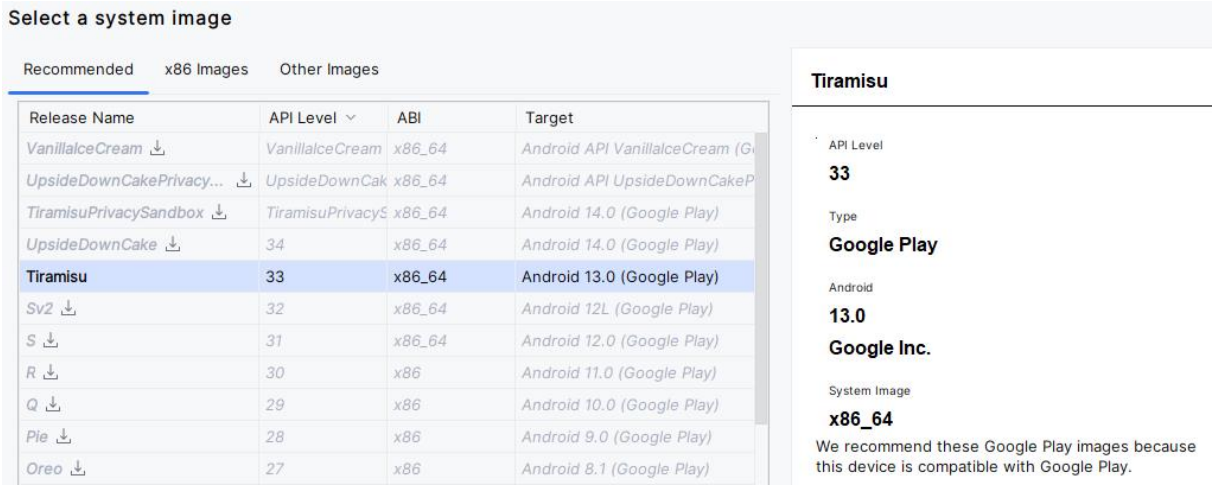


Obrázek 8, seznam virtuálních zařízení

Zdroj: Vlastní

Vybereme mobilní telefon *Google Pixel 6a* podporující aplikaci Google Play. Tato informace je uvedena v červeném rámečku na *Obrázku č. 8*. Po zvolení příslušného zařízení zvolíme „*Next*“ v pokračování nastavování.

V dalším kroku potřebujeme naistalovat OS Android. Vybraná verze OS je již zvolená, Android 13.0 Tiramisu, viz *Obrázek č. 9*. S tímto přednastaveným OS nemusíme stahovat další verze, které jsou na obrázku „zašedlé“.

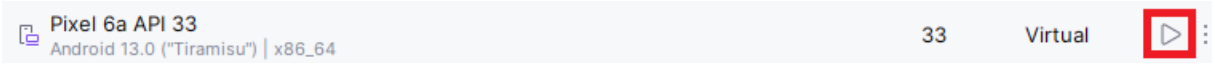


Obrázek 9, výběr operačního systému

Zdroj: Vlastní

Na další stránce si pojmenujeme zařízení, ostatní nastavení necháme tak, jak bylo nativně přednastavené a dokončíme proces tvorby virtuálního zařízení tlačítkem „*Finish*“.

Tímto se nám vytvořilo mobilní virtuální zařízení, které spustíme ikonkou pro spuštění zařízení, označeno červeným rámečkem, viz obrázek níže.



Obrázek 10, spuštění virtuálního zařízení

Zdroj: Vlastní

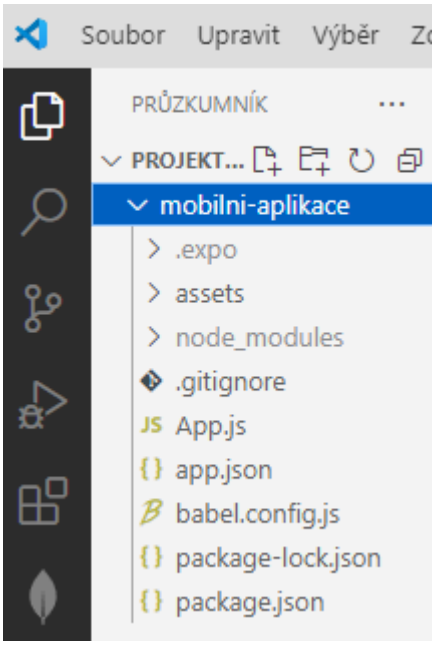
Emulátor nám spustí a zobrazí mobilní zařízení, virtuální mobilní zařízení funguje stejně jako fyzické zařízení.

Nyní můžeme spustit aplikaci na virtuálním zařízení pomocí stisknutí „a“ v terminálu VS Code dle předešlého výstupu, viz *Obrázek č. 7*.

V *Příloze F* lze vidět virtuální zařízení s uvítací obrazovkou, která je nativně dostupná při prvotním spuštění nikterak upraveného kódu aplikace.

7 Vývoj aplikace

Abychom mohli uzpůsobovat a měnit aplikaci, musíme přejít do VS Code, kde již máme nainstalovanou aplikaci. Na přiloženém obrázku níže je vidět struktura aplikace. Jako kořenový soubor aplikace je *App.js*. Dále jsou zde například data a konfigurační soubory k platformě Expo ve složce *.expo* nebo *.json* formát souboru *package.json,* ve kterém jsou nainstalované závislosti používající aplikace a její metadata.



Obrázek 11, adresářová struktura aplikace

Zdroj: Vlastní

V souboru App.js, což je základní/kořenový soubor aplikace, který se načítá při spuštění, si můžeme změnit výchozí uvítací text a tím pádem i zkontrolovat, jestli se změny propisují do emulátoru. Místo nativního textu *Open up App.js to start working on your app!* v komponentě <Text> můžeme změnit tento text na běžně používaný *Hello World!* a zjistit funkční propojení s emulátorem. Dále si můžeme na přiloženém *Obrázku č. 12* všimnout klasické výchozí struktury aplikace. V horní části jsou naimportovány položky k následné práci, pod nimi je export funkce s názvem *App* a dole na obrázku jsou definovány CSS styly. Tento princip struktury je stejný při tvorbě nových souborů.

App.js M ×

mobilni-aplikace > App.js > ...

```
1   import { StatusBar } from 'expo-status-bar';
2   import { StyleSheet, Text, View } from 'react-native';
3
4   export default function App() {
5     return (
6       <View style={styles.container}>
7         <Text>Hello World!</Text>
8         <StatusBar style="auto" />
9       </View>
10    );
11  }
12
13  const styles = StyleSheet.create({
14    container: {
15      flex: 1,
16      backgroundColor: '#fff',
17      alignItems: 'center',
18      justifyContent: 'center',
19    },
20  });
```

Obrázek 12, struktura souboru App.js

Zdroj: Vlastní

Tvorba souborů a složek

V levé části vývojového prostředí si založíme novou složku, a tu pojmenujeme *src* (ze slova source; zdroj). V této složce se budou nacházet zdrojové kódy programu. V softwarovém vývoji je vhodné dbát na úpravu struktury souborů a kódu, proto do složky src ještě vytvoříme další složku s názvem *screens*. V ní budou jednotlivé části programu. Tento proces slouží pro lepší přehlednost. V *Příloze G* v červeném čtverečku je znázorněna ikona pro vytvoření nové složky. Složky scr, a v ní složka screens, slouží pro jednotlivé soubory, které budou provádět určitou aplikační logiku. Nyní jsou v aplikaci dva, námi vytvořené soubory. Ty se vytvoří obdobně jako nová složka v IDE, jedná se o ikonu hned vedle nalevo od zvýraznění tvorby složky viz *Příloha G* (po najetí na ikony získáme popis dané funkcionality). V Reactu používáme pro soubory přípony *.jsx*.

V souboru *Task.jsx* poté bude struktura kódu, která bude tvořit jednotlivé úkoly, které si uživatel vytvoří. Jedná se také o samotnou stylizaci jednotlivých úkolů.

Postup rozdělení souborů a jednotlivých funkcionalit kódu pro projekt děláme i pro jednotlivé části aplikace. Zaměřujeme se na rozdělení aplikace do komponent, které vykonávají určitou funkci v naší aplikaci. Díky tomu můžeme snadno opětovně využít dané komponenty, máme lepší správu kódu a celkově je tento proces vhodný na údržbu aplikace. Můžeme si také

všimnout složky *assets*, ve které se nacházejí obrázky, které daná aplikace používá. Adresářová struktura je zvýrazněna v *Příloze G*.

Instalace rozšíření do IDE

Pro lepší práci si ještě do IDE nainstalujeme rozšíření s názvem *ES7+ React/Redux/React-Native snippets*. Možnost nainstalování rozšíření je v levé části IDE, předposlední ikona viz *Příloha G*.

Nyní, když si rozšíření naistalujeme, můžeme použít v našem prázdném souboru Task.jsx zkratku *rnfes*, která nám vytvoří React Native funkční komponentu, se kterou budeme dále pracovat. Poté si vytvoříme šablonu pro jednotlivé úlohy, které se do této šablony budou propisovat. Toho docílíme díky takzvaným *props*, neboli také *properties*; *vlastnosti*. Tento koncept nám umožňuje posílat různá data z jedné komponenty do druhé komponenty. Hodnoty z jedné komponenty budou dostupné uvnitř druhé komponenty, naší šablony pro úlohy, jako vlastnosti a tím pádem nemusíme tvořit kód pro samotný text úlohy a její strukturu pokaždé znovu.

Tento proces nám ulehčuje manipulaci s kódem a splňujeme hlavní koncept programování – neopakovatelnost kódu. Na přiloženém obrázku je ukázka práce s props a propojení se stylizací jednotlivých komponent. Popis jednotlivých komponent je dále v textu.

```
1   import React from "react";
2   import { View, Text, StyleSheet, TouchableOpacity } from "react-native";
3
4   const Task = (props) => {
5       return (
6           <View style={styles.item}>
7               <View style={styles.itemLeft}>
8                   <TouchableOpacity style={styles.square}></TouchableOpacity>
9                   <Text style={styles.itemText}>{props.text}</Text>
10              </View>
11          </View>
12      )
13  }
```

Obrázek 13, ukázka souboru Task.jsx

Zdroj: Vlastní

7.1 Vývoj aplikační logiky

V následujícím souboru, *TodoList.jsx*, se budeme zabývat samotnou logikou vytváření úloh. Jedná se o uložení, načtení a smazání úlohy. Zde již musíme naimportovat pro náš projekt více komponent zobrazené na obrázku níže.

```
1   import React, { useState, useEffect } from "react";
2   import { Text, View, KeyboardAvoidingView, StyleSheet, TextInput, TouchableOpacity, ScrollView }
3   from "react-native";
4   import AsyncStorage from '@react-native-async-storage/async-storage';
5   import Task from "./Task";
```

Obrázek 14, import komponent

Zdroj: Vlastní

Je zde několik nových komponent. Jako první je *useState*. Ta nám umožňuje uchovávat stav a dle potřeby ho aktualizovat. Tato komponenta vrací dva stavy – aktuální hodnotu a hodnotu pro aktualizaci. Další komponentou je *useEffect*, která poskytuje připojení k externímu systému (např. k lokálnímu úložišti *AsyncStorage*). Jako useState, i tato komponenta uchovává počáteční stav a hodnotu pro aktualizaci. Pro upřesnění, v React Native nazýváme funkce useState, useEffect popřípadě useContext jako *hooks*. Hooks jsou funkce, které komponentám umožňují získat funkcionalitu. Důležitou součástí projektu je asynchronní úložiště AsyncStorage. AsyncStorage je takzvané API, které lokálně na našem zařízení ukládá a načítá data. Toto úložiště se hodí pro uložení malého objemu dat jako například uživatelské nastavení či naše poznámky. Aby nám toto uložiště fungovalo, musíme ho i nainstalovat, nejen naimportovat.

Pro tento úkon nám slouží příkaz z NPM knihovny – *npm install @react-native-async-storage/async-storage.* Asynchronní úložiště se nám tedy nainstaluje a je připraveno na použití. *Poznámka:* asynchronní úložiště funguje na principu, že neblokuje hlavní vlákno procesu, a tudíž samotná aplikace může být v běhu.

Jako poslední si musíme naimportovat samotný soubor Task.jsx, ve kterém je struktura poznámek a skrze props posíláme do této komponenty text poznámky ze souboru TodoList.jsx.

7.1.1 Načtení poznámky z paměti

Do funkční komponenty s názvem *TasksScreen* budeme psát náš kód. Deklarujeme hooky useState, první pro jednotlivý text úkolu, druhý pro celý seznam úkolů, deklarace je následovná – *const [task, setTask] = useState()*;. Toto je standardní zápis, jak použít useState; do hranatých závorek se poté píše počáteční proměnná a funkce, která bude měnit hodnoty, v našem případě proměnná *task* a funkce *setTask*. Tomuto procesu se také říká destrukturalizace

pole. Pro celý seznam úkolů je postup stejný, až na zadání jiných názvů proměnné, funkce a hooku useState, který bude vracet pole, jedná se totiž o seznam, tj. pole. Výsledná deklarace je následovná – *const [taskItems, setTaskItems] = useState([])*;. Ukázka je níže na obrázku.

```
7   const TasksScreen = function () {
8       const [task, setTask] = useState();
9       const [taskItems, setTaskItems] = useState([]);
```

Obrázek 15, nastavení hooků useState

Zdroj: Vlastní

K načtení úkolů z asynchronní paměti budeme potřebovat useEffect. Prvním argumentem je funkce a druhým je pole závislostí. Založíme si asynchronní funkci s názvem loadTasks. Pro zachycení a zpracování chyb budeme používat bloky kódu *try* a *catch*. V prvním bloku provádíme požadované instrukce a pokud se zde vyskytne chyba, provede se druhý blok kódu.

Princip je takový, že chceme-li načíst uložené poznámky, musíme deklarovat konstantu *savedTasks*, která se rovná asynchronní paměti a která načítá poznámky pod námi zvoleným klíčem *tasks*. Slovem *await* ve funkci se program zastaví a čeká na dokončení, dokud nám funkce *getItem()* nevrátí naše hodnoty. Pokud se nám hodnota uložených poznámek nerovná nule, tak to znamená, že jsme načetli poznámky.

Když načteme poznámky, tak musíme aktualizovat proměnnou v useState pomocí druhého parametru useState, a to funkcí setTaskItems, která mění obsah celého pole/seznamu poznámek. Tato hodnota je načtena ve formátu *JSON*, takže si ji musíme pomocí funkce *JSON.parse()* převést. Pokud se však při načítání úkolů vyskytne chyba, pomocí bloku kódu catch() ji odhalíme. Funkci *loadTaks* poté zavoláme takto – *loadTaks()*.

UseEffect obsahuje dva argumenty, první je náš kód, druhým argumentem je pole závislostí. Náš useEfect se bude spouštět jen jednou, tudíž pole necháme prázdné. Pokud dojde chyba v bloku try, provede se blok catch. Jako parametr si zvolíme proměnnou *error*, do které se uloží případné chyby. Metodou *console.error()*, která přijímá jeden nebo více argumentů, vypisujeme chybovou hlášku spolu s konkrétní chybou do prostředí, ve kterém se aplikace spouští. Ukázka celého procesu načtení dat je na obrázku níže.

```
10      useEffect(() => {
11          // Načtení uložených úkolů při prvním načtení komponenty
12          const loadTasks = async () => {
13              try {
14                  const savedTasks = await SecureStore.getItemAsync('tasks');
15                  if (savedTasks !== null) {
16                      setTaskItems(JSON.parse(savedTasks));
17                  }
18              } catch (error) {
19                  console.error('Chyba při načítání úkolů:', error);
20              }
21          };
22          loadTasks();
23      }, []);
```

Obrázek 16, načtení uložených úkolů z paměti

Zdroj: Vlastní

7.1.2 Uložení poznámky do paměti

Dále musíme řešit uložení nového úkolu do AsyncStorage. K tomu budeme používat také bloky try a catch. Deklarujeme si tedy novou asynchronní funkci s názvem *handleAddTask*. V bloku try deklarujeme konstantu *newTasks*, která bude reprezentovat pole. Do tohoto pole budeme vkládat stávající úkoly a k nim nové úkoly, stávající úkoly rozdělíme pomocí *spread* operátoru (tři tečky), který nám provádí to, že vytvoří nové pole obsahující všechny prvky stávajícího pole taskItems. Na konec tohoto pole se přidá naše nová poznámka uložená v proměnné *task*.

Poté nový seznam úkolů uložíme do asynchronního úložiště. I zde použijeme klíčové slovo await a namísto funkce getItem() zde musíme použít funkci *setItem()*. Funkce setItem() v asynchronním úložišti bude obsahovat náš klíč – tasks a funkci pro převedení textu na JSON řetězec *JSON.stringify()* z naší nově vytvořené poznámky. Hook useState nadále změní proměnnou taskItems pomocí funkce setTaskItems na nový seznam úkolů. Funkci setTask nastavíme na prázdný řetězec a prázdný řetězec je předám proměnné task, čehož docílíme, že

můžeme zadávat nové úkoly. Blok pro detekci chyb je stejný jako u předešlého kódu pro načítání poznámek. Výsledný kód je přiložen níže.

```
25      const handleAddTask = async () => {
26          // Uložení nového úkolu do úložiště
27          try {
28              const newTasks = [...taskItems, task];
29              await SecureStore.setItemAsync('tasks', JSON.stringify(newTasks));
30              setTaskItems(newTasks);
31              setTask('');
32          } catch (error) {
33              console.error('Chyba při ukládání úkolu:', error);
34          }
35      };
```

Obrázek 17, uložení nového úkolu do paměti

Zdroj: Vlastní

7.1.3 Odstranění poznámky z paměti

Součástí aplikace je také odstranění dokončeného či chybně zadaného úkolu. Založíme si novou asynchronní funkci uloženou do konstanty s názvem *completeTask*. Na rozdíl od předešlých asynchronních funkcí, tato funkce bude přijímat jeden parametr a to *index*. Jedná se o index, kterým jsou očíslovány jednotlivé položky v poli, v našem případě poznámky/úkoly. I v tomto případě použijeme bloky kódu try a catch. Blok catch je stejný jako u předešlých ukázek. V bloku try si deklarujeme novou proměnnou pomocí klíčového slova *let*, do které uložíme kopii seznamu úkolů, tudíž do pole pomocí spread operátoru přidáme proměnnou taskItem z hooku useState, díky kterému měníme obsah ze seznamu úkolů.

Dalším krokem je odstranění položky z pole úkolů. K nově vytvořené proměnné přidáme metodu *splice()* pomocí tečkové syntaxe, která nám slouží k odstranění či nahrazení daných hodnot na základě námi zvolených parametrů, těmi jsou index a číslovka 1. Index určuje pozici v poli a číslice 1 znamená, kolik prvků má být odstraněno. Dále potřebujeme po odstranění úkolu aktualizovat paměť, takže musíme aktualizovat AsyncStorage, tj. identifikovat úkoly a převést je na textový řetězec, kdy pak tato data (úkoly) můžeme uložit do paměti. Do paměti posíláme úkoly, které se nacházejí v proměnné *itemsCopy*.

Poté pomocí funkce setTaskItem aktualizujeme naše upravené pole úkolů, aby se nám ve správné úpravě zobrazilo na obrazovku zařízení. Kopie seznamu úkolů se nastaví jako nový stav. Výsledný kód je přiložen níže na obrázku.

```
37      const completeTask = async (index) => {
38          // Odstranění dokončeného úkolu a aktualizace úložiště
39          try {
40              let itemsCopy = [...taskItems];
41              itemsCopy.splice(index, 1);
42              await SecureStore.setItemAsync('tasks', JSON.stringify(itemsCopy));
43              setTaskItems(itemsCopy);
44          } catch (error) {
45              console.error('Chyba při odstraňování úkolu:', error);
46          }
47      };
```

Obrázek 18, odstranění úkolu z paměti

Zdroj: Vlastní

Vrácení komponenty ze souboru TodoList.jsx

Nyní musíme z funkce TasksScreen vrátit požadovanou komponentu. To nám ve funkci umožní *return()*, jakožto standardní část funkcí, kdy požadujeme vrácení hodnot. Jednotlivé komponenty, což jsou také základní stavební prvky v React Native, budou reprezentovat naši strukturu pro zobrazování úkolů, s okolním prostředím a prostředím pro tvorbu poznámek. Toho docílíme pomocí komponent *View, Text, ScrollView, TouchableOpacity*, námi naimportovaném souboru Task.jsx (v kódu použití jako komponenta *Task*), *KeyboardAvoidingView* a *TextInput*. Hierarchický postup je následovný. Všechny komponenty budou obaleny do jedné komponenty a to View, obecně tato komponenta slouží jako kontejner pro další komponenty.

Dále vytvoříme další kontejner, a to pro jednotlivé úkoly. Do komponenty Text vkládáme textové řetězce, v našem případě byl zvolen text *„Co mám dnes udělat…“*. Komponenta ScrollView nám zajistí, že pokud na obrazovce zařízení bude více úkolů, tak pomocí této komponenty můžeme posouvat obsah. V komponentě View nadále naprogramujeme malý kousek JavaScript kódu. Tento kód musíme „obalit“ do složených závorek. Tímto způsobem můžeme do React Native aplikace vkládat i čistý JavaScript.

Bude se nám jednat o to, abychom do komponenty View zobrazily jednotlivé úkoly. Budeme potřebovat anonymní funkci, poté je potřeba „zmapovat“ pole uložených úkolů pomocí funkce *map()* a použijeme proměnnou taskItems, která se nachází v hooku useState pro ukládání úkolů. Pomocí tečkové notace tuto proměnnou připojíme k mapování (procházení

v poli). Do funkce map() napíšeme anonymní funkci, která bude hledat podle položky a indexu. Položka reprezentovaná parametrem *item* je aktuální prvek pole a index je číselné označení prvku v daném poli. Anonymní funkci použijeme proto, že ji můžeme použít přímo v kódu a může být tedy jeho součástí a zároveň tuto funkci použít jako parametry pro jinou funkci, v našem případě pro funkci map().

Anonymní funkce se také vyznačuje tím, že nemá název, je například uložená do proměnné či konstanty. Tato funkce nám bude vracet komponentu TouchableOpacity, která reaguje na interakci od uživatele. Jako klíč zde používáme index a pomocí vlastnosti *onPress* voláme funkci completeTask, která přijímá argument index a po kliknutí na daný úkol se nám tudíž komponenta smaže. Do této komponenty také vkládáme komponentu Task, která je naimportována spolu s dalšími komponentami, kdy tato komponenta odkazuje na soubor Task.jsx, kdy v tomto souboru udáváme strukturu výsledného úkolu. V komponentě Task jsou vlastnosti *key* a *text*, také známé jako props. Key/klíč nám reprezentuje proměnná index a do vlastnosti text dáváme proměnnou item. Ukázka vrácení komponenty je v *Příloze H*.

Tvorba nových poznámek

Sekce pro vkládání a vytváření poznámek je tvořena následovně. Použijeme komponentu KeyboarAvoidingView, do které budeme psát kód. Tato komponenta je ošetřena tak, že automatické vyskakování klávesnice na zařízení je uzpůsobeno tak, aby nám tato klávesnice nepřekrývala důležité prvky v aplikaci. V našem případě toto použití není nutné, ale je vhodné pracovat s touto komponentou. Jako vstupní pole použijeme komponentu TextInput s nativním textem *„Napsat úkol“*, jako *obsah/value* je proměnná *task* a vlastností *onChangeText* posíláme do funkce setTask parametr *text*. Pozn. funkce setTask je funkce v hooku useState() a provádí aktualizaci právě zadaného úkolu.

Tlačítkem v podobě TouchableOpacity poté po kliknutí uživatelem voláme funkci handleAddTask, která je přiřazena události onPress. Funkce handleAddTask se zavolá a výsledem této funkce je uložení právě napsaného úkolu do paměti v zařízení. Tato komponenta ještě obsahuje kontejner View a komponentu Text, ve které je znak „+“, který je intuitivní pro vytvoření nového úkolu. Komponenty v tomto souboru, TodoList.jsx, jsou také patřičně nastylované. Ukázka kódu je v *Příloze H*.

Stylizace

Každá komponenta obsahuje atribut *style*, do kterého vkládáme naše vytvořené styly jako objekty, které jsou tvořeny pomocí metody *StyleSheet.create()* uložené v konstantě pojmenované *styles*. Přidání stylů poté vypadá například následovně: *<KeyboardAvoidingView style={styles.writeTaskWrapper}>*. Do atributu style vkládáme do složených závorek název konstanty se styly styles a pomocí tečkové notace připojíme název objektu, ve kterém je psán kód v jazyce CSS určený pro stylování prvků. Na přiloženém obrázku je ukázka procesu stylizace.

```
81  const styles = StyleSheet.create({
82      writeTaskWrapper: {
83          position: "absolute",
84          bottom: "4%",
85          width: "100%",
86          flexDirection: "row",
87          justifyContent: "space-around",
88          alignItems: "center",
89      },
```

Obrázek 19, ukázka stylizace

Zdroj: Vlastní

Export celé komponenty

Posledním krokem je vyexportovat samotnou funkci TaskScreen, ve které je psaná celá funkční logika kódu. Na samotný konec programu napíšeme klíčová slova *export default TaskScreen;*. Díky tomuto bude celá komponenta psaná v souboru TodoList.jsx dostupná i v jiných souborech po jejím naimportování.

Seskupení komponent

Abychom nyní vše seskupili, budeme pracovat v kořenovém souboru aplikace App.js. Zde je potřeba naimportovat několik knihoven s komponenty a moduly potřebné pro další postup. Vytvoříme si zde biometrické ověřování a logiku pro navigaci mezi stránkami v aplikaci. Navigační logiku pro naší aplikaci výhradně nepotřebujeme, pracujeme jen s jednou obrazovkou – screen a na žádné další nepřecházíme. Je ale vhodné tuto navigaci do projektu zahrnout, např. pro další vývoj aplikace do budoucna a pro celistvost projektu, kdy tento prvek budeme v mobilním vývoji skoro vždy využívat, protože mnoho aplikací používá více než jednu obrazovku – screen.

Jako první si naimportujeme či nainstalujeme potřebné knihovny a moduly. Budeme zde používat samotnou knihovnu React, kterou používají všechny soubory psané v Reactu, dále

hooky useState a useEfect, komponenty View, StyleSheet, Modal, Text, StatusBar a soubor TodoList. Předešlé již známe, kromě Modal a StatusBar. Komponenta Modal nám bude sloužit k zobrazení důležitých informací, v našem případě k zobrazení biometrické autentizace. Modal nám zajistí, že interaktivní část obrazovky bude fungovat pouze v té části, kde jsme si nadefinovali Modal. StatusBar nám zobrazí horní pruh týkající se stavových hlášení v telefonu (připojení k síti, stav baterie, oznámení atd.), díky této komponentě budeme mít toto prostředí zobrazené. Naimportované komponenty a závislosti jsou přiloženy na obrázku níže.

```
1  import React, { useState, useEffect } from 'react';
2  import { View, StyleSheet, Modal, Text } from 'react-native';
3  import { StatusBar } from 'expo-status-bar';
4  import { NavigationContainer } from '@react-navigation/native';
5  import { createNativeStackNavigator } from '@react-navigation/native-stack';
6  import * as LocalAuthentication from 'expo-local-authentication';
7  import TodoList from './src/screens/TodoList';
```

Obrázek 20, import komponent a závislostí

Zdroj: Vlastní

Navigace

Pro nastavení navigace, která nám slouží pro přecházení mezi obrazovkami si musíme tyto závislosti a komponenty nainstalovat. I když máme jen jednostránkovou aplikaci, tak si tuto navigaci zakomponujeme do aplikace, aplikaci budeme mít připravenou pro další vývoj. Z knihovny NPM si nainstalujeme potřebné závislosti, první je *npm install @react-navigation/native*, která vytvoří navigační strom a řízení navigace.

Druhá závislost je *npm install @react-navigation/native-stack*, kdy tato funkcionalita nám bude řadit obrazovky do zásobníku. Třetí instalací, *npx expo install react-native-screens react-native-safe-area-context* zajistíme lepší kompatibilitu mezi Expo a React Native. Poslední instalací bude přidání autentizace pomocí příkazu *npm install expo-local-authentication.* Nainstalované závislosti najdeme také v souboru *package.json* v objektu *dependencies*. Ukázky importů a závislostí jsou na *Obrázku č. 20*.

Pro tvorbu navigace potřebujeme vytvořit konstantu *Stack*, ve které je funkce *createNativeStackNavigator()*, s konstantou Stack budeme dále v menu pracovat. Dále budeme psát kód do funkční komponenty App. Tu si můžeme deklarovat i tímto způsobem – *export default function App(){ kód }*. Tímto způsobem se tato funkce přímo exportuje oproti například funkci v souboru TodoList.jsx, kde jsme ji museli zavolat až na konci programu. Ukázka je na *Obrázku č. 20.*

Autentizace uživatele

Nyní vytvoříme autentizaci pro uživatele. V tomto případě se do aplikace dostaneme pomocí zabezpečení, které máme nastavené v reálném zařízení či ve virtuálním. Používáme-li biometrickou autentizaci, aplikace nás vyzve k zadání otisku prstů, používáme-li PIN kód, aplikace nás vyzve k zadání PIN kódu atd., kdy tyto druhy zabezpečení máme nastaveny v mobilním zařízení a které používáme.

Vytvoříme si hook useState, proměnná bude *authentication* a funkce *setAuthentication*, výchozí hodnota useState bude *true.* Dále potřebujeme hook useEffect, díky kterému docílíme jen jednoho spuštění ověření, pole tudíž bude prázdné. V useEffect je dále asynchronní funkce *authenticate*, která provádí autentizaci uživatele. Po provedení autentizace se hodnota ve funkci setAuthentication v hooku useState nastaví na *false*, čímž byla autentizace dokončena. Při chybných pokusech po nás zařízení bude chtít stále prověřit naši totožnost. Tímto máme vyřešený hook useEffect. Ukázka kódu je přiložena níže.

```
9   const Stack = createNativeStackNavigator();
10
11  export default function App() {
12    const [authentication, setAuthentication] = useState(true);
13    useEffect(() => {
14      async function authenticate() {
15        const result = await LocalAuthentication.authenticateAsync();
16        setAuthentication(false);
17      }
18      authenticate();
19    }, []);
```

Obrázek 21, autentizace uživatele

Zdroj: Vlastní

Vrácení komponenty ze souboru App.js

Z funkční komponenty App nyní požadujeme vrátit příslušné komponenty. Hlavní komponentou bude View, do které budeme vkládat další komponenty. Nachází se zde StatusBar, který díky vlastnosti *style* nastavené na *auto* nám přizpůsobí pozadí dle nativně zvoleného v zařízení a vlastnosti hidden nastavené na false, která nám zajistí, že StatusBar se bude zobrazovat. Nyní potřebujeme komponentu *NavigationContainer*, která nám bude obalovat celou navigační strukturu aplikace. Dále použijeme komponentu *Stack.Navigator*, díky které definujeme konkrétní navigační zásobník (udržujeme jím historii procházení mezi obrazovkami) a definujeme zde ostatní konfigurace pro obrazovky/screens. Nyní pomocí komponenty *Stack.Screen* definujeme jednotlivé screeny pomocí vlastností. Používáme zde

název, určení komponenty, která se má zobrazit a konfiguraci, která je nastavena pro nadpis. Můžeme si také všimnout, že zde používáme konstantu Stack, která obsahuje funkci *createNativeStackNavigator(),* viz předešlý text. Tento způsob práce je typický pro navigaci v React Native. Ukázka je přiložena níže.

```
21    return (
22      <View style={styles.container}>
23        <StatusBar style="dark" hidden={false} />
24        <NavigationContainer>
25          <Stack.Navigator initialRouteName="Home">
26            <Stack.Screen
27              name="ToDoList"
28              component={TodoList}
29              options={{ title: 'Úkolníček' }}
30            />
31          </Stack.Navigator>
32        </NavigationContainer>
```

Obrázek 22, vrácení komponenty ze souboru App.js

Zdroj: Vlastní

Nyní budeme pracovat s prostředím autentizace. V komponentě Modal zobrazíme požadovanou autentizaci, pokud má být tato komponenta viditelná, je hodnota *authetication* nastavena na *true* viz předešlý text ohledně deklarace useState. Nachází se zde také komponenta View a Text. View nám slouží jako černé pozadí, kdy při autentizaci by se mohlo stát, že by nás zařízení vyzvalo k autentizaci, ale tato sekce se může zobrazit ne na celé výšce zařízení, a tudíž by mohlo dojít k tomu, že by byly vidět některé poznámky a použití autentizace by trochu ztrácelo smyls. Na tomto pozadí je poté napsán text „*Autorizovaný přístup*“. Ukázka je na obrázku níže.

```
33        <Modal visible={authentication} transparent>
34          <View style={styles.modalBackground}>
35            <Text style={styles.textReport}>Autorizovaný přístup</Text>
36          </View>
37        </Modal>
38      </View>
```

Obrázek 23, prostředí pro autentizaci uživatele

Zdroj: Vlastní

Všechny potřebné komponenty jsou také patřičně nastylovány způsobem, který je popisován výše v textu.

Grafická vizualizace

Dostáváme se do závěrečné fáze vývoje. Vytvoříme si *splash screen*, což je obrazovka, která se nám bude načítat při spouštění aplikace. Tu vytvoříme v programu Canva. Dále vytvoříme logo pro aplikaci, které se bude zobrazovat na naší ploše tak, jak u mobilních telefonů známe. Pro tuto akci použijeme program IconKitchen.

Nově vytvořené obrázky vložíme do složky *assets*, kde se nachází obrázky, které aplikace využívá. Po vložení načítací obrazovky a ikony do složky se přesuneme do souboru *app.json.* Zde upravíme názvy obrázků (cesty k nim jsou stejné, jak bylo nastaveno nativně) u vlastností objektu *expo.* Jsou to *icon*, u vnořených objektů *splash* a *android* měníme vlastnosti *image* a *foregroundImage.* V tomto souboru si také můžeme pojmenovat aplikaci, kdy zadaný název se bude zobrazovat jako popisek aplikace na zařízení. V našem případě byl zvolen název *„Úkolníček“.* Níže je ukázka nastavení názvu aplikace a její ikony.

```
1  {
2    "expo": {
3      "name": "Úkolníček",
4      "slug": "mobilni-aplikace",
5      "version": "1.0.0",
6      "orientation": "portrait",
7      "icon": "./assets/icon.png",
```

Obrázek 24, konfigurace souboru app.json

Zdroj: Vlastní

Zde je ukázka načítací obrazovky splash screen a ikona pro aplikaci.

Obrázek 25, ukázka načítací obrazovky

Zdroj: Vlastní

Obrázek 26, ukázka ikony pro aplikaci

Zdroj: Vlastní

Aplikaci si samozřejmě procesem vývoje zobrazujeme na virtuálním či fyzickém zařízení, abychom kontrolovali dosažení daných změn, které chceme provést.

7.2 Export aplikace do spustitelného souboru

Jako poslední krok je samotné vyexportování aplikace do souboru *.apk*, který nám umožní nainstalování aplikace na reálné zařízení bez použití služeb od společnosti Google. Nyní si potřebujeme založit účet na stránce *https://expo.dev*. Zde máme prostředí, kam se nám ukládají soubory, které lze distribuovat do Google Play či na naše zařízení.

Postup tvorby instalačních souborů se liší jen zadaným kódem dle účelu distribuce. Účet jsme si založili a nyní přejdeme do příkazové řádky ve Visual Studiu. Příkazem *expo login* se dle našich údajů přihlásíme do našeho účtu. Nyní do příkazové řádky vložíme následující příkaz – *eas build -p android --profile preview.* Spuštěním příkazu se nám také vytvoří nový soubor *eas.json*, ve kterém jsou nastavení pro sestavení a distribuci aplikace. EAS je zkratka pro *Expo Application Services.* Nyní se nám tvoří výsledná aplikace. Tento proces chvíli potrvá.

Po úspěšném sestavení aplikace se přesuneme do účtu Expo, kam se nám instalační soubor uloží. Je vhodné si tuto stránku spustit na mobilním zařízení, protože právě na mobilní zařízení si budeme aplikaci stahovat. Po rozkliknutí nově vytvořené aplikace se nám zobrazí možnost *Install*. Soubor .apk si tedy nainstalujeme do zařízení a po stažení jej spustíme. Výsledná aplikace s ukázkou poznámek je zobrazena v *Příloze I*.

ZÁVĚR

V této bakalářské práci bylo cílem naprogramovat a zabezpečit mobilní aplikaci. Tato aplikace obsahuje poznámkový blok, kdy poznámky se dají tvořit, mazat a načítat. Jako programovací jazyk byl zvolen React Native.

V teoretické části je popis témat spjatých se zadaným tématem; úvod do operačních systémů, úvod do mobilních operačních systémů. U těchto témat je zmíněna historie, vývoj a funkčnost. Zmíněny jsou zde dva největší zástupci od každého druhu OS, a to MS Windows jakožto OS pro stolní počítače a Google Android jakožto OS pro mobilní zařízení. Ve světě je ale velmi populární také operační systém macOS spolu s OS na mobilní zařízení iPhone, oba systémy od firmy Apple.

Dále je v práci zmíněno vývojové prostředí, pro platformu Windows je nejznámější a nejpoužívanější IDE Microsoft Visual Studio. OS s macOS mohou také toto IDE použít, ale je zde uvedeno i IDE XCode, protože toto vývojové prostředí je úzce spjato právě s macOS. Kdyby se v této práci nepoužíval na vývoj aplikace React Native, museli bychom pro tvorbu aplikace v macOS pro mobilní zařízení s iOS používat například jazyk Swift a vyvíjet v IDE XCode.

Poté je v textu práce vysvětlen i samotný React Native a potřebné závislosti pro vyvíjení aplikací. Je zde zmíněno, co to je autentizace, její základní principy a dělení.

Praktická část je zaměřena na vývoj mobilní aplikace. Nachází se zde popis procesu tvorby aplikace, od nastavení IDE a Android emulátoru, až po samotný proces tvorby logiky aplikace zakončený exportem do instalačního souboru.

POUŽITÁ LITERATURA

[5] KLATOVSKÝ, Karel a Josef PECINOVSKÝ. Windows 11 snadno a rychle. Praha: Grada Publishing, 2022. ISBN 978-80-271-2316-2.

[41] HUB, Miloslav. Bezpečnost a ochrana informací v prostředí internetu. Pardubice: Univerzita Pardubice, 2013. ISBN 978-80-7395-701-8

[1] BIGELOW, Stephen. *Operating system (OS)* [online]. 1 [cit. 2024-03-02]. Dostupné z: https://www.techtarget.com/whatis/definition/operating-system-OS

[2] KLIMEŠ, Cyril. *OPERAČNÍ SYSTÉMY 1* [online]. Ostrava, 2013 [cit. 2024-03-02]. Dostupné z: https://publi.cz/download/publication/577?pc=1. Distanční opora. Ostravská univerzita, Přírodovědecká fakulta.

[3] *Types of Operating Systems* [online]. 1 [cit. 2024-03-02]. Dostupné z: https://www.geeksforgeeks.org/types-of-operating-systems/

[4] SAILELLAH, Hassan Rizky Putra. *The Definition of Windows Operating System: History, Functions, and Features* [online]. 2023 [cit. 2024-04-23]. Dostupné z: https://it.telkomuniversity.ac.id/en/windows-operating-system/

[6] Windows [online]. 2024 [cit. 2024-04-23]. Dostupné z: https://www.computerhope.com/jargon/w/windows.htm

[7] What is macOS? [online]. [cit. 2024-04-23]. Dostupné z: https://www.javatpoint.com/what-is-macos

[8] RAMANATHAN, Tara. MacOS [online]. [cit. 2024-04-23]. Dostupné z: https://www.britannica.com/technology/macOS

[9] STRICKLAND, Jonathan. How Mac OS X Works: The Purpose of Operating Systems [online]. [cit. 2024-04-23]. Dostupné z: https://computer.howstuffworks.com/macs/mac-os-x1.htm

[10] APPLE. Apple Developer. Online. Dostupné z: https://developer.apple.com/library/archive/documentation/MacOSX/Conceptual/OSX_Technology_Overview/SystemTechnology/SystemTechnology.html#//apple_ref/doc/uid/TP40001067-CH207-BCICAIFJ. [cit. 2024-04-28].

[11] XNU. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2022 [cit. 2024-04-23]. Dostupné z: https://cs.wikipedia.org/wiki/XNU

[12] HASLAM, Karen. MacOS 14 Sonoma superguide: Everything you need to know [online]. 2024 [cit. 2024-04-23]. Dostupné z: https://www.macworld.com/article/1670742/macos-14-release-date-features-compatibility-beta.html

[13] GOAD, Michael. Mobile operating system [online]. [cit. 2024-04-23]. Dostupné z: https://www.techtarget.com/searchmobilecomputing/definition/mobile-operating-system

[14] SCHMIDT, Cory. What is Android? Here is a complete guide for beginners [online]. 2016 [cit. 2024-04-23]. Dostupné z: https://www.nextpit.com/what-is-android

[15] CALLAHAM, John. The history of Android: The evolution of the biggest mobile OS in the world [online]. 2023 [cit. 2024-04-23]. Dostupné z: https://www.androidauthority.com/history-android-os-name-789433/

[16] Platform architecture [online]. 2023 [cit. 2024-04-23]. Dostupné z: https://developer.android.com/guide/platform

[17] BROWN, C. Scott. What is Android? Here's everything you need to know [online]. 2022 [cit. 2024-04-23]. Dostupné z: https://www.androidauthority.com/what-is-android-328076/

[18] Historie verzí Androidu. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2024 [cit. 2024-04-23]. Dostupné z: https://cs.wikipedia.org/wiki/Historie_verz%C3%AD_Androidu

[19] VOLLE, Adam. IOS [online]. 2024 [cit. 2024-04-23]. Dostupné z: https://www.britannica.com/topic/iOS

[20] POSEY, Brien. Apple iOS [online]. 2023 [cit. 2024-04-23]. Dostupné z: https://www.techtarget.com/searchmobilecomputing/definition/iOS

[21] MOES, Tibor. What is iOS? Everything You Need to Know [online]. 2023 [cit. 2024-04-23]. Dostupné z: https://softwarelab.org/blog/what-is-ios/

[22] NATIONS, Daniel. What Is iOS? [online]. 2021 [cit. 2024-04-23]. Dostupné z: https://www.lifewire.com/what-is-ios-1994355

[23] ROBERTS, Sienna. What is iOS? A Comprehensive Introduction to Apple's OS [online]. 2024 [cit. 2024-04-23]. Dostupné z: https://www.theknowledgeacademy.com/blog/what-is-ios/

[24] CASSERLY, Martyn. IOS versions: Every version of iOS from the oldest to the newest [online]. 2024 [cit. 2024-04-23]. Dostupné z: https://www.macworld.com/article/1659017/ios-versions-list.html

[25] HANNA, Katie Terrell. Mobile app [online]. 2023 [cit. 2024-04-23]. Dostupné z: https://www.techtarget.com/whatis/definition/mobile-app

[26] What is an IDE? – Integrated Development Environment [online]. 2024 [cit. 2024-04-23]. Dostupné z: https://www.geeksforgeeks.org/what-is-ide/

[27] What is Visual Studio? [online]. 2023 [cit. 2024-04-23]. Dostupné z: https://learn.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2022

[28] Introduction to Visual Studio [online]. 2023 [cit. 2024-04-23]. Dostupné z: https://www.geeksforgeeks.org/introduction-to-visual-studio/

[29] HINDI, Daniel. How to Code Xcode: A Complete Tutorial For Beginners [online]. [cit. 2024-04-23]. Dostupné z: https://buildfire.com/xcode-tutorial/#:~:text=Xcode%20is%20Apple%E2%80%99s%20official%20IDE%20%28integrated%20development%20environment%29.,that%20consolidate%20the%20different%20aspects%20of%20building%20software.

[30] What is Android Studio? [online]. 2024 [cit. 2024-04-23]. Dostupné z: https://www.geeksforgeeks.org/overview-of-android-studio/

[31] Android Studio [online]. 2023 [cit. 2024-04-23]. Dostupné z: https://www.techtarget.com/searchmobilecomputing/definition/Android-Studio

[32] What is an Android Emulator? [online]. 2023 [cit. 2024-04-23]. Dostupné z: https://www.geeksforgeeks.org/what-is-an-android-emulator/

[33] Chapter 1. What Is React Native? [online]. [cit. 2024-04-23]. Dostupné z: https://www.oreilly.com/library/view/learning-react-native/9781491929049/ch01.html

[34] PATERSKA, Patrycja. React Native's popularity is still growing. Learn the pros and cons, and when to use React Native for your mobile app development. [online]. 2024 [cit. 2024-04-23]. Dostupné z: https://www.elpassion.com/blog/what-is-react-native-and-when-to-use-it

[35] MUTHUKUMARANA, Nimesha. Node.js — What You Need To Know and Why You Should Use It [online]. 2022 [cit. 2024-04-23]. Dostupné z: https://medium.com/@nimesha.muthukumarana/node-js-what-you-need-to-know-and-why-you-should-use-it-fe8e94d9fe4e

[36] DAHL, Ryan. OPENJS FOUNDATION. Node.js [online]. [cit. 2024-04-23]. Dostupné z: https://nodejs.org/en/learn/getting-started/introduction-to-nodejs

[37] SCHLUETER, Isaac Z. NPM, INC. NPM [online]. [cit. 2024-04-23]. Dostupné z: https://docs.npmjs.com/about-npm

[38] REFSNES DATA AS. W3Schools [online]. [cit. 2024-04-23]. Dostupné z: https://www.w3schools.com/whatis/whatis_npm.asp

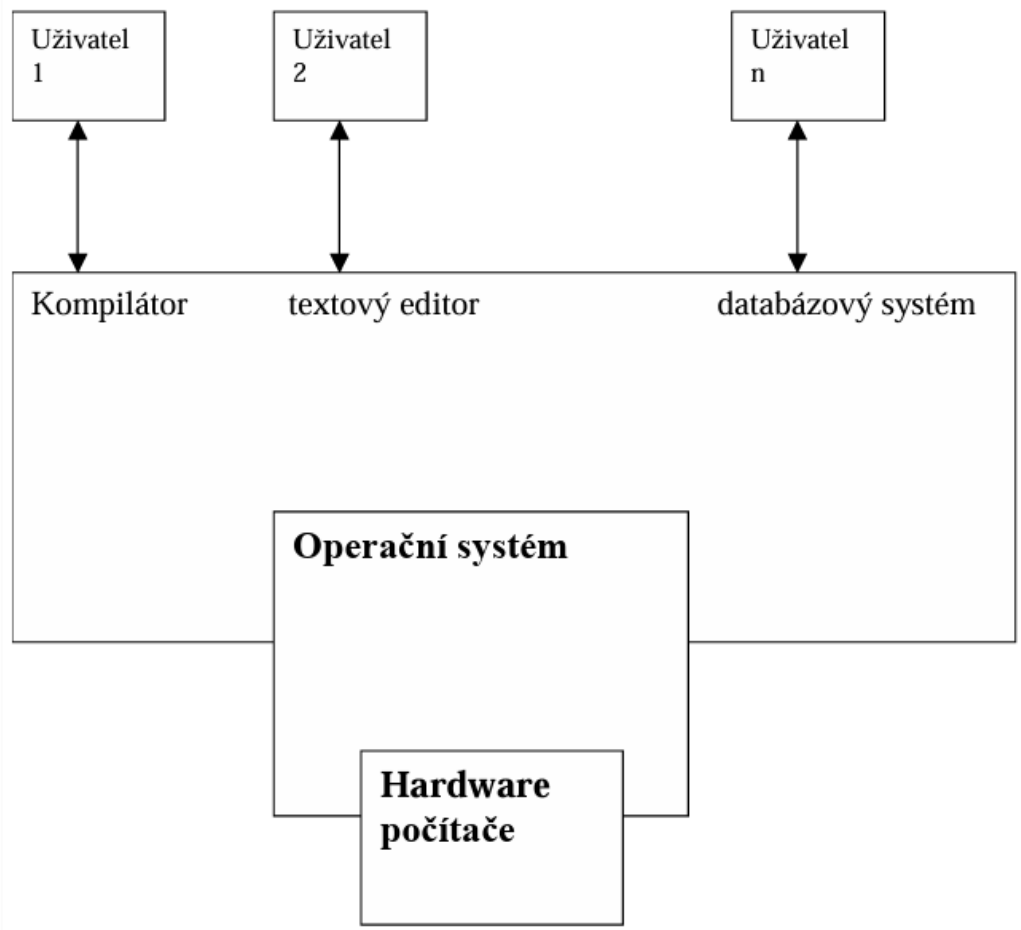
[39] PAL, Tanushree. React Native CLI vs Expo CLI [online]. [cit. 2024-04-23]. Dostupné z: https://www.4waytechnologies.com/blog/react-native-cli-vs-expo-cli

[40] SHARMA, Ajay. React Native CLI or Expo [online]. 2023 [cit. 2024-04-23]. Dostupné z: https://www.linkedin.com/pulse/react-native-cli-expo-ajay-sharma

[42] OPENAI. ChatGPT. Online. Dostupné z: https://chat.openai.com/. [cit. 2024-04-29].

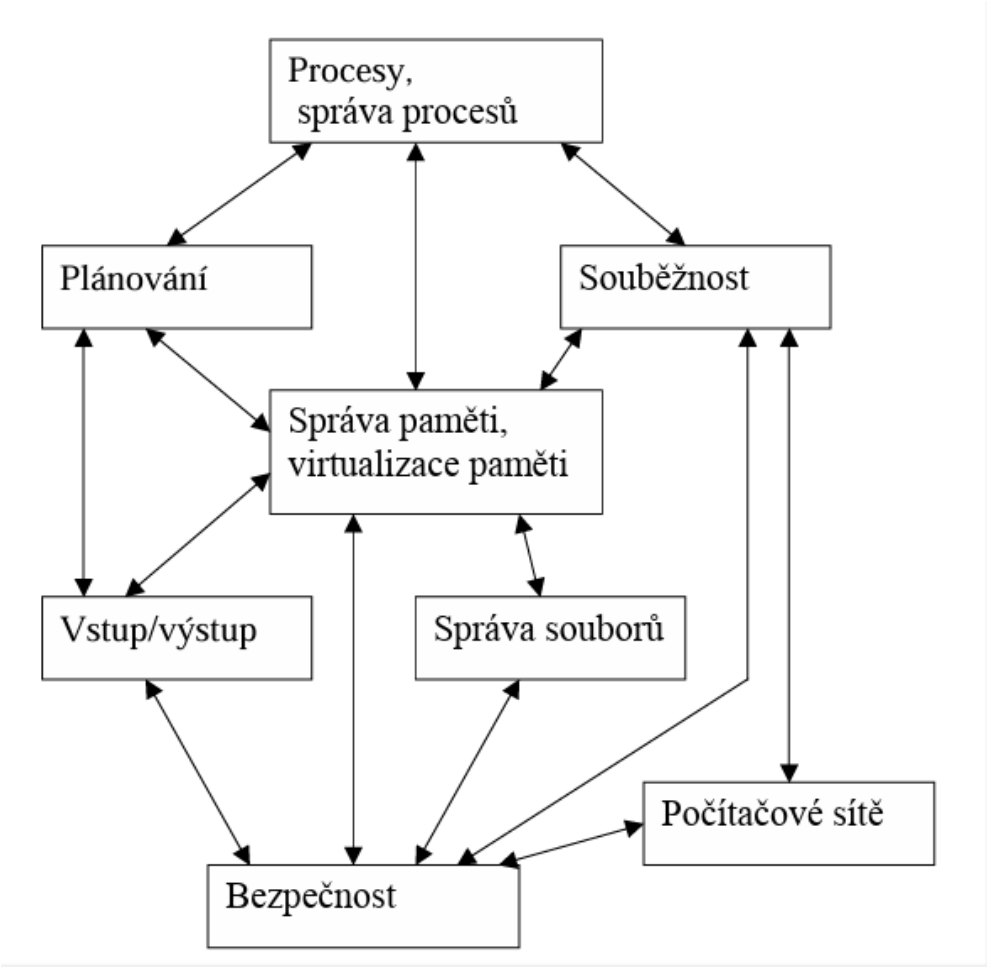
SEZNAM PŘÍLOH

Příloha A – Oddělení jednotlivých komponent od operačního systému



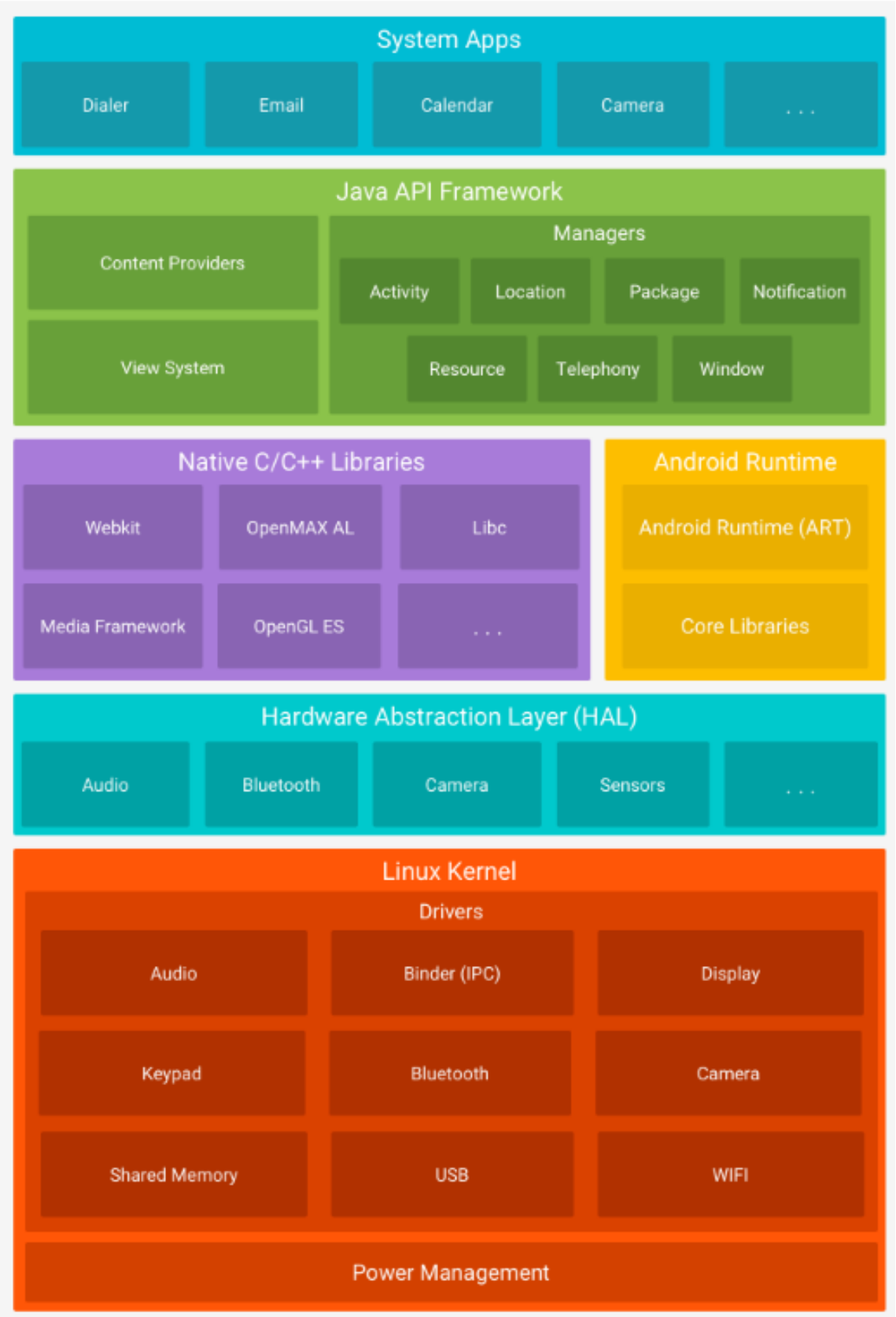
Zdroj: převzato a upraveno dle [2]

Příloha B – Oblasti služeb operačních systémů



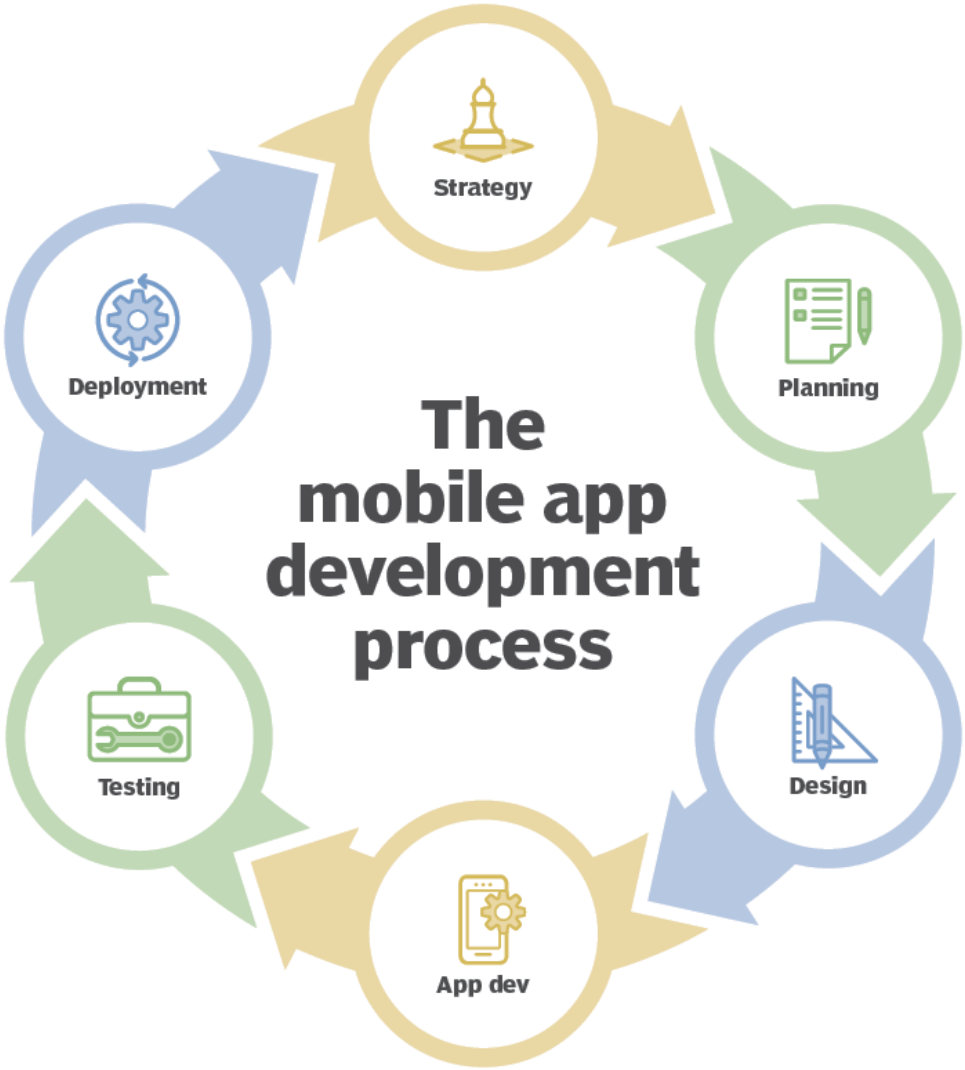
Zdroj: převzato a upraveno dle [2]

Příloha C – Architektura platformy Android



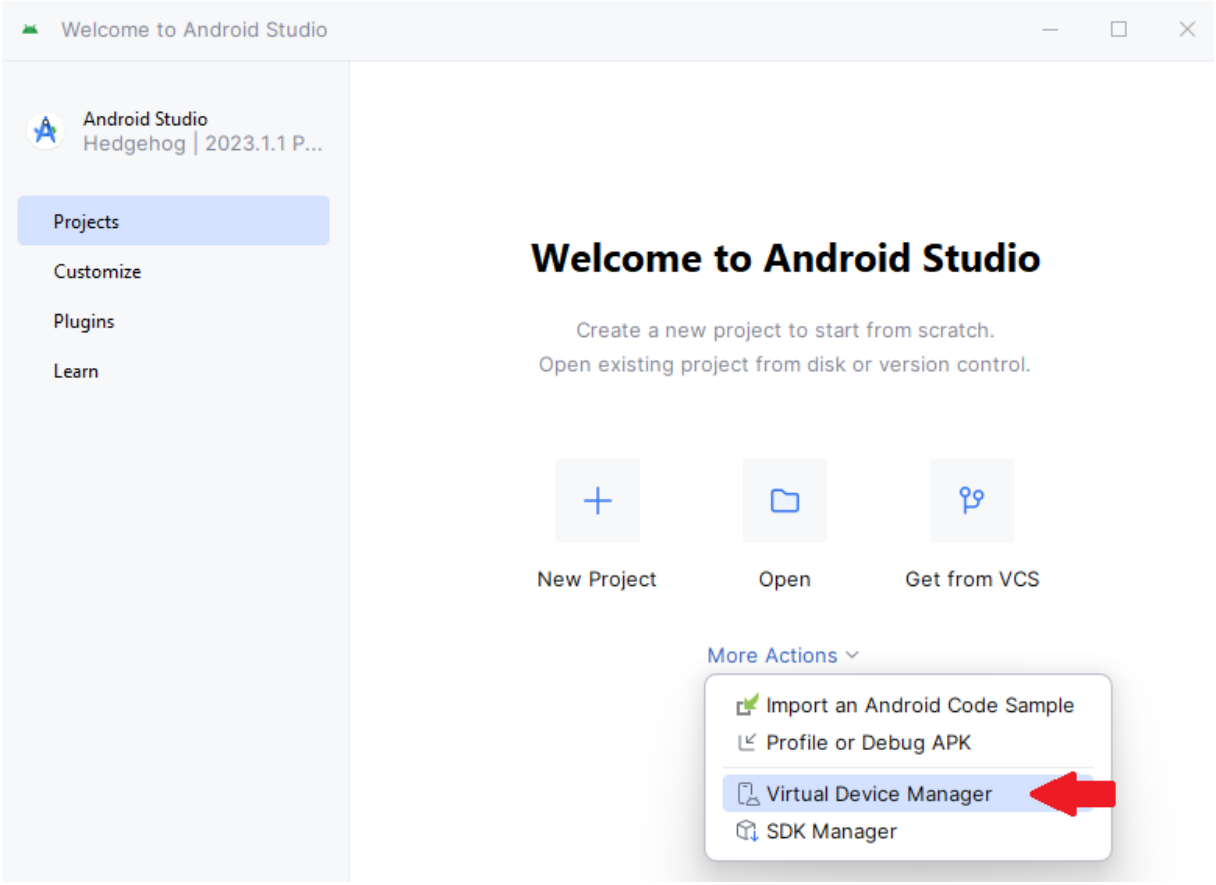
Zdroj: převzato z [16]

Příloha D – Proces vývoje mobilní aplikace



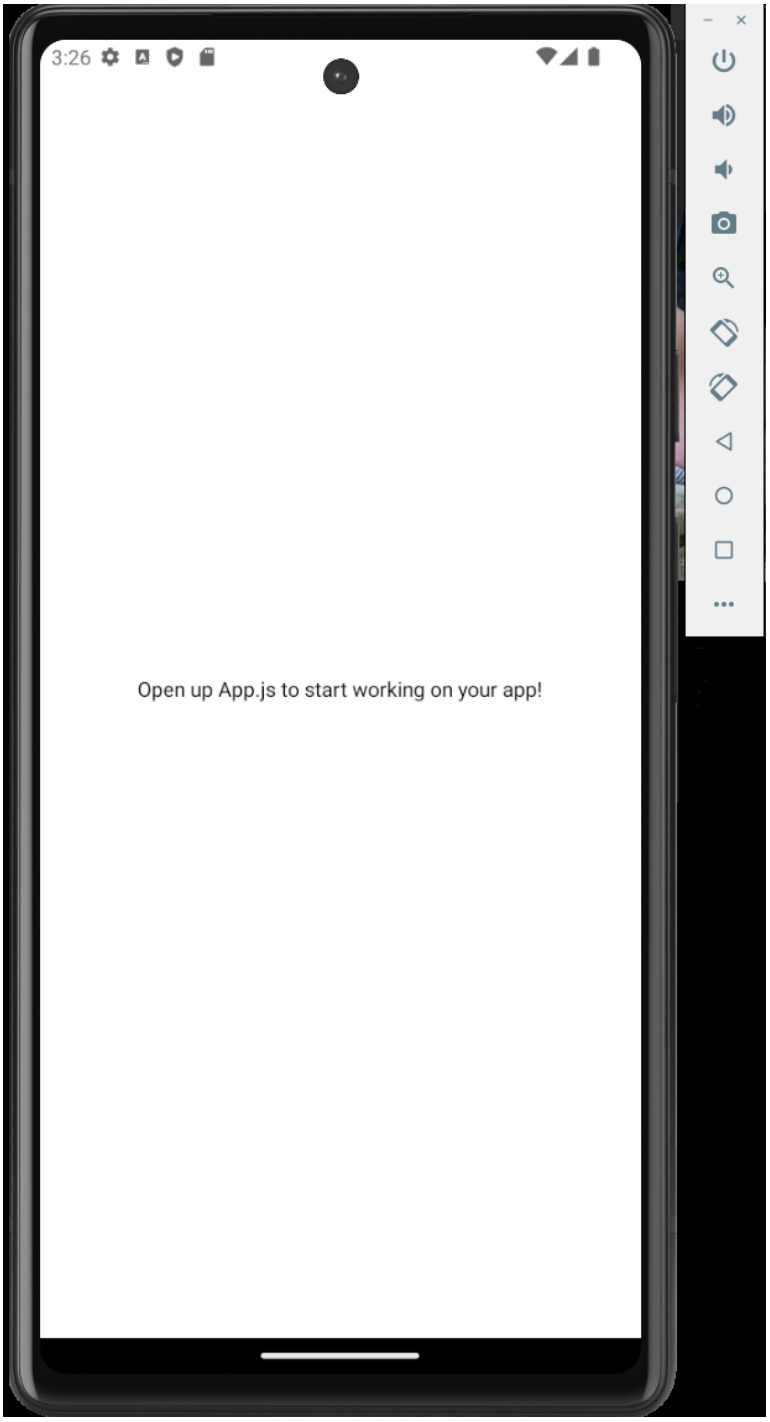
Zdroj: převzato z [25]

Příloha E – Proces tvorby virtuálního zařízení



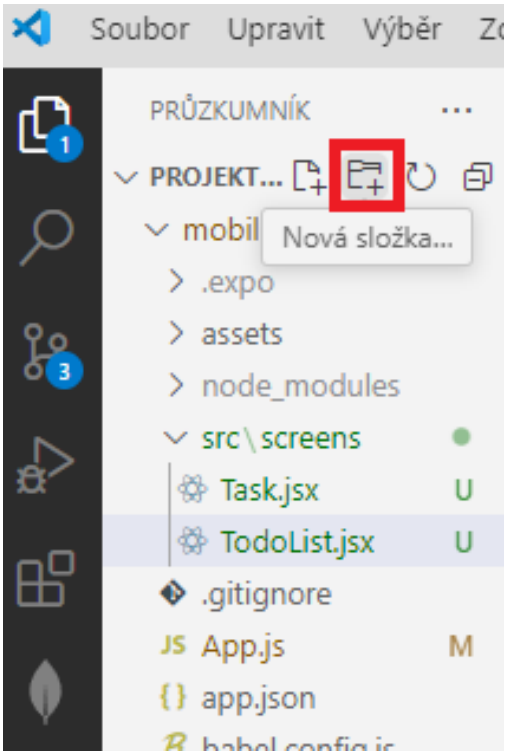
Zdroj: Vlastní

Příloha F – Virtuální zařízení



Zdroj: Vlastní

Příloha G – Struktura projektu



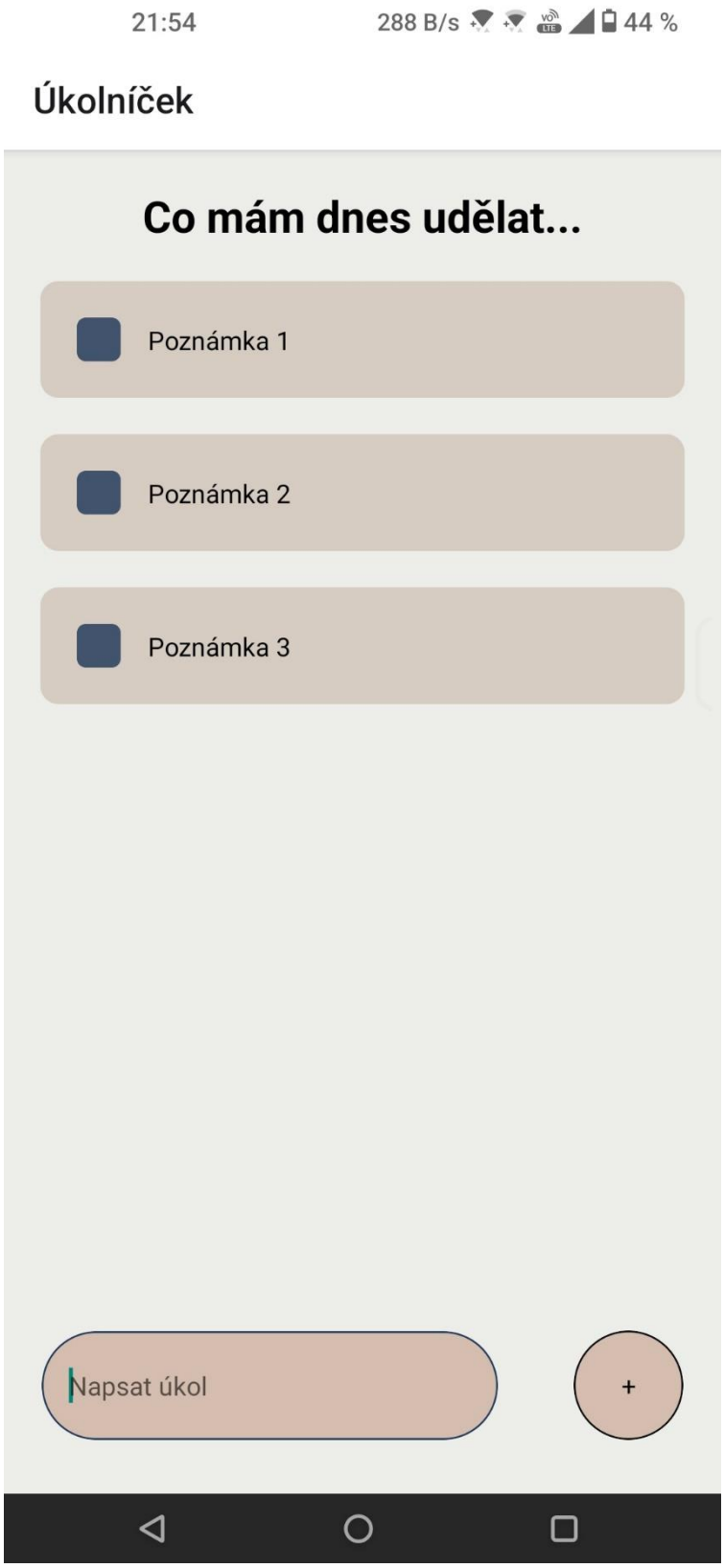
Zdroj: Vlastní

Příloha H – Ukázka kódu

```
49      return (
50          <View style={styles.Container}>
51              <View style={styles.List}>
52                  <Text style={styles.Text}>Co mám dnes udělat...</Text>
53                  <ScrollView>
54                      <View >
55                          {
56                              taskItems.map((item, index) => {
57                                  return (
58                                      <TouchableOpacity key={index}
59                                          onPress={() => completeTask(index)}>
60                                          <Task key={index} text={item} />
61                                      </TouchableOpacity>
62                                  )
63                              })
64                          }
65                      </View>
66                  </ScrollView>
67              </View>
68              <KeyboardAvoidingView style={styles.writeTaskWrapper}>
69                  <TextInput style={styles.input} placeholder={"Napsat úkol"}
70                      value={task} onChangeText={text => setTask(text)} />
71                  <TouchableOpacity onPress={handleAddTask}>
72                      <View style={styles.addWrapper}>
73                          <Text style={styles.addText}>+</Text>
74                      </View>
75                  </TouchableOpacity>
76              </KeyboardAvoidingView>
77          </View>
78      );
79  };
```

Zdroj: Vlastní

Příloha I – Ukázka aplikace na reálném zařízení



Zdroj: Vlastní