Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Mobilní management Dockeru

Diplomová práce

Autor: Michal Macinka
Studijní obor: Aplikovaná informatika

Vedoucí práce: doc. Mgr. Tomáš Kozel, Ph.D.

Hradec Králové srpen 2018

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 14.8.2018

Poděkování:

Děkuji vedoucímu diplomové práce doc. Mgr. Tomáši Kozlovi, Ph.D. za metodické vedení práce.

Anotace

Cílem této diplomové práce je seznámení s moderním způsobem vývoje, testování a distribuce aplikací pomocí platformy Docker se zaměřením na možnosti její správy. Diplomová práce je rozdělena do tří částí. První část práce se věnuje seznámení s platformou Docker, porovnání kontejnerizace a virtualizace, popisem technologie a využití při vývoji a testování softwaru. Druhá část práce se zabývá popisem multiplatformního vývoje mobilních aplikací, srovnáním dostupných možností a nástrojů pro vývoj a výběr vhodné technologie pro implementaci ukázkové aplikace. Poslední část je zaměřena na návrh a vlastní implementaci multiplatformní mobilní aplikace pro vzdálenou správu platformy Docker pro operační systémy Android a iOS.

Annotation

Title: Docker Mobile Management

The aim of this master thesis is to get acquainted with the modern way of development, testing and distributing of applications using the Docker platform and focusing on the possibilities of its administration. This thesis is divided into three parts. The first part is dedicated to get acquainted with the Docker platform, comparison of containerization and virtualization, description of technology and use in software development and testing. The second part describes the cross-platform development of mobile applications, comparing available options and tools for development and selection of suitable technology for implementation of sample application. The last part is focused on the design and implementation of its own cross-platform mobile applications for remote management of the Docker platform for operating systems Android and iOS.

Obsah

1 Úvod ... *1*

2 Cíl práce ... *3*

3 Kontejnerizační systémy ... *4*

3.1 Virtualizace ... **4**

3.2 Docker ... **8**

4 Mobilní platformy ... *13*

4.1 Android ... **13**

4.2 iOS ... **18**

5 Multiplatformní vývoj ... *21*

5.1 Přístupy k vývoji mobilních aplikací ... **21**

5.2 Výhody a nevýhody ... **29**

5.3 Závěr ... **30**

6 Návrh aplikace ... *32*

6.1 Mobilní klient ... **32**

6.2 Middleware ... **36**

6.3 Docker cluster ... **38**

7 Realizace aplikace ... *39*

8 Závěr ... *44*

9 Seznam použité literatury ... *45*

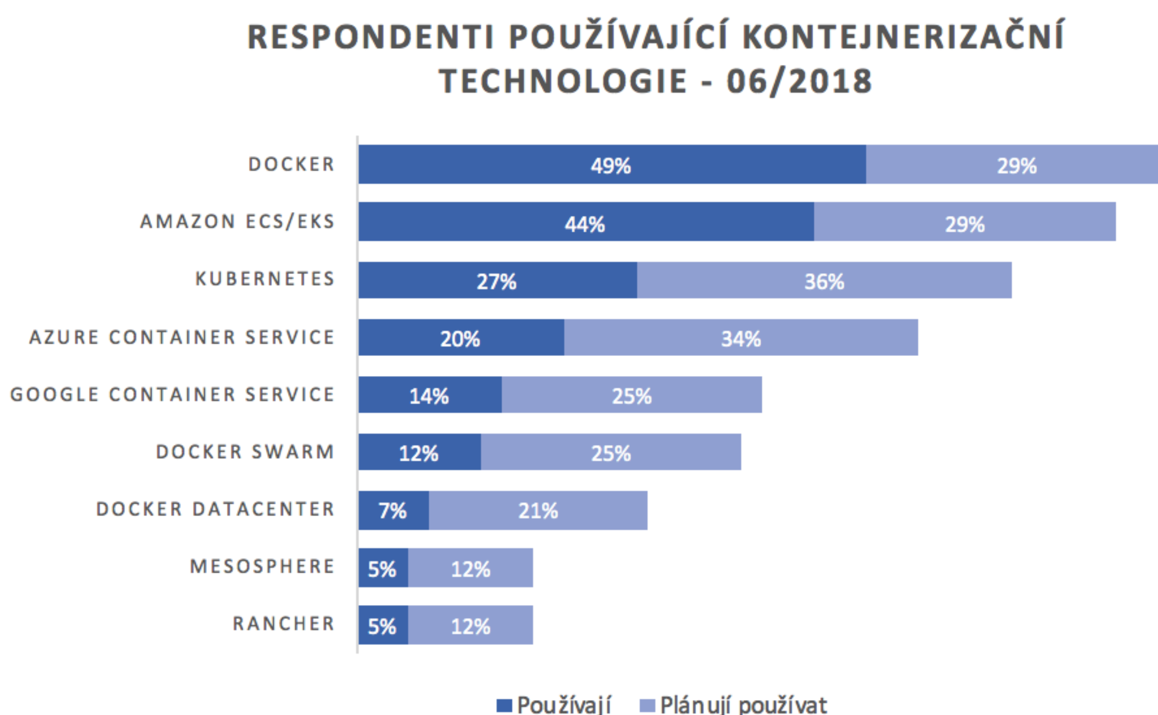
10 Seznam obrázků ... *51*

11 Seznam tabulek ... *52*

1 Úvod

V dnešní době není vždy snadné vyvíjet, testovat nebo přenášet aplikace mezi různými počítači, a to především kvůli rozmanitosti prostředí jednotlivých uživatelů. Důvodů může být spousta, ať už se jedná o závislosti nainstalovaných knihoven, dodatečných programů nebo například různých operačních systémů. Aby toto bylo možné, je potřeba docílit stejného nastavení na všech cílových zařízeních. Bohužel, to nebývá vždy snadné, a právě z tohoto důvodu byl vytvořen nástroj Docker.

Kontejnerová virtualizace, především Docker, je v současnosti jednou z nejpopulárnějších technologií v cloud computingu, což je patrné z Obrázek 1.



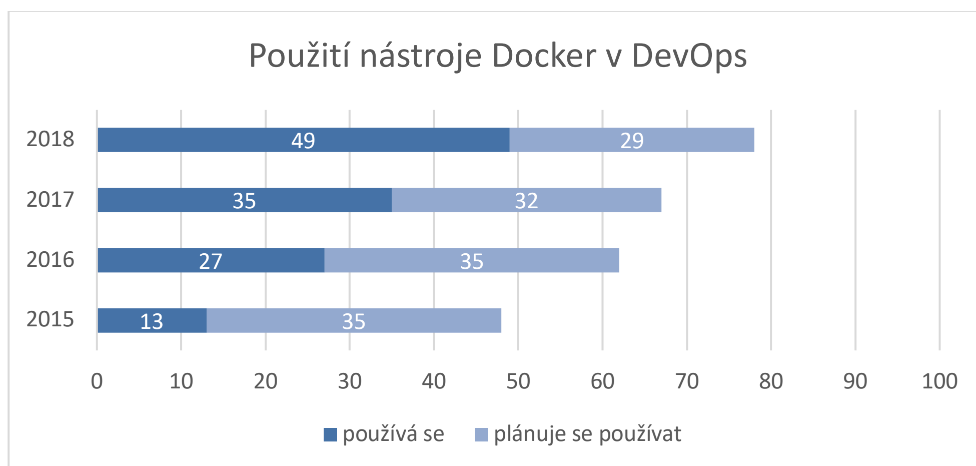
Obrázek 1 Použití kontejnerizační technologií. Zdroj: [1]

Tato technologie mění způsob, jak jsou dnes aplikace vyvíjeny a provozovány. Je obtížné jmenovat některou z velkých společností, která nevyužívá kontejnerizační technologie. Myšlenka kontejnerů není nová, operační systémy založené na Linuxu disponovaly touto virtualizací procesů od počátku roku 2000 [2]. Nicméně, až vstoupení Dockeru na scénu způsobilo využití těchto technologií jako hlavních nástrojů v přístupu při vývoji softwaru. Tento přístup, který zdůrazňuje spolupráci, komunikaci a integraci mezi vývojáři, testery

a odborníky na informační technologie se zkráceně nazývá DevOps (složený tvar anglických výrazů Development a Operations) [3]. Podle [4] bylo využití Dockeru:

- během prvního roku 13 % [5],
- mezi roky 2015 a 2016 se počet respondentů navýšil o 14 % [6],
- rok 2016 přinesl 11 % nárůst [7],
- v současnosti využívá Docker 49 % respondentů [1].

Na Obrázek 2 jsou shrnuty odpovědi respondentů, kteří využívají nebo plánují využívat tuto technologii do současné doby.



Obrázek 2 Využití nástroje Docker od roku 2015 do současnosti. Zdroj: [autor]

2 Cíl práce

Hlavním cílem této diplomové práce je vytvoření multiplatformní mobilní aplikace pro operační systémy Android a iOS. Tato mobilní aplikace bude sloužit ke vzdálené správě produkčního Docker prostředí Institutu moderních informačních technologií na Fakultě informatiky a managementu.

Součástí práce bude seznámení s nástrojem Docker, popisem architektury, výhod a nevýhod této technologie a porovnáním virtualizace na úrovni operačního systému se serverovou virtualizací. Další částí je popis možných přístupů k vývoji mobilních aplikací, seznámení s aktuálními a populárními technologiemi a výběr nejvhodnější technologie pro implementaci mobilní aplikace.

Praktická část práce se věnuje problematice implementovaného middlewaru pro komunikaci se serverovým prostředím a samotnou multiplatformní aplikací.

3 Kontejnerizační systémy

Serverová virtualizace, která zahrnuje běh více virtuálních serverů na jednom fyzickém počítači, je všudypřítomná a je to jeden z klíčových základních prvků moderní architektury cloud computingu. Virtualizace umožňuje sdílet zdroje fyzického serveru, jako je procesor, paměť RAM, přístup k síti a úložiště, mezi několik virtuálních serverů. Virtualizace pomáhá podstatně snižovat náklady na podporu komplexního výpočetního prostředí. Mimo to pomáhá k urychlení nasazení nových strojů, protože virtuální počítače mohou být spuštěny nebo škálovány za zlomek času potřebného k objednání, obdržení a konfiguraci fyzických serverů.

3.1 Virtualizace

3.1.1 Virtuální stroje

Na rozdíl od fyzických strojů, virtuální stroje (VM = Virtual Machine) ve skutečnosti neexistují – je to softwarový artefakt, který napodobuje fyzický server a skládá se jen z množiny souborů [8]. Mezi soubory VM je hlavní konfigurační soubor, který určuje kolik paměti a úložiště je alokováno daným virtuálním strojem. Konfigurační soubor také pojmenovává virtuální síťové karty přiřazené virtuálnímu stroji, stejně jako vstupní a výstupní periferie, ke kterým má přístup. Konfigurační soubory zobrazují úložiště VM jako soubor virtuálních disků, které jsou ve skutečnosti soubory v základním souborovém systému.

Když administrátor potřebuje duplikovat fyzický server, je vyžadováno hodně práce pro získání nového serveru, instalace operačního systému a jeho aplikací a kopírování dat. Vzhledem k tomu, že virtuální stroj je jen řada souborů, je možné vytvořit duplicitní server v řádu několika minut provedením potřebných změn v konfiguračním souboru. Případně je možné vytvářet nové virtuální stroje pomocí šablon. Šablona obsahuje výchozí nastavení pro hardware a software. Nástroje pro automatizaci mohou jednoduše tyto šablony využívat a přizpůsobovat je při nasazení nových serverů.

Hypervisor

Hypervisor je klíčový software pro hardwarovou virtualizaci, který umožňuje na hostitelském serveru vytvářet a spravovat virtuální stroje [8]. Jeho funkcí je virtualizovat

všechny zdroje hostitele jako procesory, operační paměť, úložiště, sítě a rozdělovat je mezi virtuální stroje spuštěné nad hypervisorem. Je zprostředkovatelem mezi hostitelským systémem a virtualizovaným hostovským systémem. Obsahuje správce virtuálních strojů (virtual machine manager = VMM), který je zodpovědný za správu VM spuštěných na serveru.

3.1.2 Kontejnery

Kontejnerizace je typ serverové virtualizace na úrovni operačního systému. Používá software nazývaný virtualizační kontejner, který běží na hostitelském operačním systému a poskytuje prostředí pro aplikace [8]. Kontejnery zaujímají odlišný přístup oproti běžným virtualizacím, které většinou používají hypervisory. Cílem virtualizace kontejneru není emulovat fyzický server pomocí hypervisoru. Všechny kontejnerizované aplikace sdílejí společné jádro operačního systému. Tento přístup snižuje využití zdrojů, protože není nutné spouštět vlastní operační systém pro každou aplikaci zvlášť. Kontejnery izolují aplikace mezi sebou na sdíleném operačním systému. Kontejnerizovaná aplikace běží na operačním systému hostitelského počítače (Linux nebo Windows).

Kontejnerizace není novou technologií. S vývojem systému Unix V7 byl v roce 1979 představen systém chroot, který umožňoval změnit kořenovou složku pro běžící proces a jeho potomky do nového umístění v souborovém systému. Tento pokrok znamenal počátek izolování procesů mezi sebou. Chroot byl přidán do distribuce BSD v roce 1982 [9]. Od této doby se kontejnerizační technologie vyvinuly až do podoby nástrojů, které jsou používány velkým množstvím uživatelů jako například Docker nebo Kubernetes.

Běžným použitím této technologie je modernizace stávajících aplikací. Zabalení existujících aplikací do kontejnerů okamžitě zlepšuje zabezpečení, snižuje produkční náklady a umožňuje migraci napříč cloudovým prostředím. Díky této transformaci je možné tyto balíčky distribuovat na libovolnou infrastrukturu a eliminovat tím běžný problém „na mém stroji to fungovalo“ [10].

Výhody a nevýhody

Použití kontejnerizace je v dnešní moderní době velmi rozšířené díky výhodám, které poskytuje. Mezi tyto výhody patří následující [8] [11]:

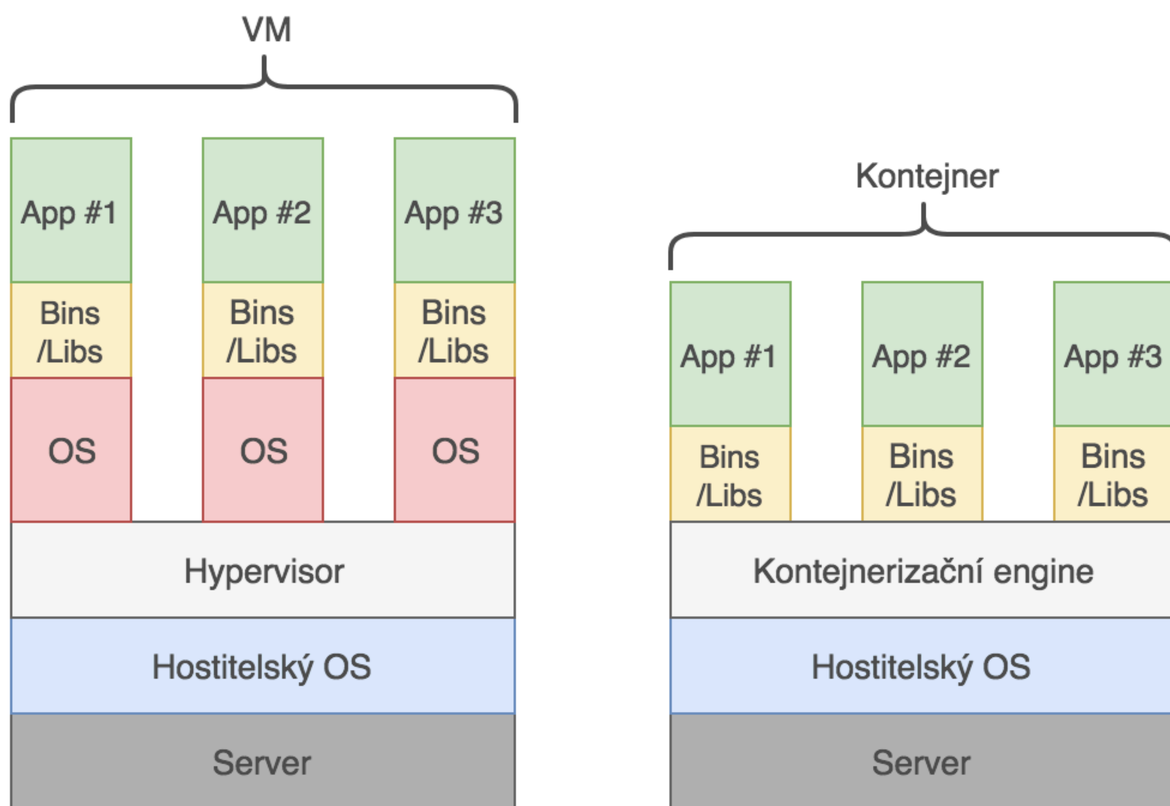
- velikost kontejnerů je v řádu desítek megabytů,
- běh kontejnerů je méně náročný na zdroje,
- jednoduchá škálovatelnost,
- vytváření nových kontejnerů trvá několik vteřin,
- snazší přiřazování zdrojů a spouštění aplikací na různých prostředích,
- použití kontejnerů může zkrátit čas potřebný pro vývoj, testování a nasazení aplikací,
- testování a hledání chyb je mnohem jednodušší, protože neexistuje rozdíl mezi během aplikace na lokálním, testovacím a produkčním prostředí,
- kontejnery jsou velmi efektivním řešením vzhledem k ceně, mohou snížit operační (méně serverů, méně obslužného personálu) a vývojové náklady (vývoj na konzistentním běhovém prostředí),
- kontejnerizace je ideální řešení pro vývoj mikroslužeb, DevOps a průběžný vývoj.

Vedle výhod je s kontejnerizací spojeno i několik nevýhod [12]:

- zabezpečení a izolovatelnost – z důvodu sdílení jádra, ostatních komponent systému a root přístupu jsou od sebe kontejnery navzájem méně izolované než virtuální stroje,
- menší flexibilita v operačních systémech – v případě požadavků na jiný operační systém je nutné spustit nový server,
- síťování – obtížná konfigurace pro různé případy použití, např. limitování přístupu ke kontejneru a současné zajištění síťové komunikace.

3.1.3 Porovnání technologií

I přes to, že virtuální stroje a kontejnery mají některé vlastnosti společné, existují mezi nimi značné rozdíly. Na Obrázek 3 je zobrazena struktura virtuálního stroje a kontejneru.



Obrázek 3 Struktura virtuálního stroje a kontejneru Zdroj: [12]

Virtuální stroje

Virtuální stroje na rozdíl od kontejnerů obsahují kompletní operační systém se všemi knihovnami a aplikacemi. Tento typ virtualizace bývá náročný na zdroje. V závislosti na operačním systému VM může zabírat mnoho úložného prostoru. Každý VM má vlastní jádro, které nesdílí s hostitelským počítačem, tudíž všechny VM jsou izolované na velmi nízké úrovni. VM, které běží na stejném hostitelském serveru, mohou používat různé operační systémy.

Kontejnery

Kontejnery sdílejí jádro hostitelského operačního systému. To mimo jiné znamená, že všechny kontejnery na stejném serveru používají stejný operační systém. Průměrná velikost kontejnerů je v rozmezí několika desítek megabytů oproti virtuálním strojům, které mohou zabírat několik gigabytů.

3.2 Docker

Docker je jednou z nejznámějších kontejnerových platforem, která je stále více nasazována pro vývoj a provoz přenosných distribuovaných aplikací. Administrátor nebo vývojář může využít Docker pro vytváření, odesílání a spouštění aplikací na různých platformách včetně cloudu, a to jak na fyzických, tak i virtuálních strojích.

Klíčovým důvodem pro široký úspěch kontejnerizace a Dockeru je zejména to, že kontejnerizace umožňuje efektivní nasazení aplikací v cloudu. Docker extrémně zjednodušil kontejnerizaci aplikací.

Docker kontejnery balí software v kompletní souborový systém, který obsahuje vše, co daná aplikace potřebuje k běhu. To znamená, že aplikace vždy poběží stejným způsobem nehledě na prostředí. Docker kontejnery nabízí následující klíčové výhody [13]:

- nenáročný
  - obrazy jsou vytvořeny z vrstvených souborových systémů, takže více obrazů sdílí společné soubory, čímž se sníží využití disku,
  - kontejnery sdílejí stejné jádro operačního systému, čímž efektivněji využívají operační paměť,
- otevřené standardy
  - model otevřených standardů znamená, že kontejnery běží na všech linuxových distribucích,
- bezpečnost
  - aplikace jsou díky kontejnerům od sebe navzájem izolovány, stejně jako od samotné infrastruktury.

Docker je využíván hlavně pro aplikace navržené a nakonfigurované pro běh v kontejneru, tzv. kontejnerizované aplikace [8]. Tradiční aplikace běží přímo na hostitelském operačním systému. Používají souborový systém, tabulku procesů, síťová rozhraní, porty a další. Problémem těchto aplikací je nemožnost spustit souběžně více verzí na stejném serveru z důvodu způsobení konfliktů. Docker poskytuje nástroj a platformu pro správu životního cyklu kontejnerů. Využití tohoto nástroje může být následující:

- vývoj aplikací nebo jejích jednotlivých komponent pomocí kontejnerů,

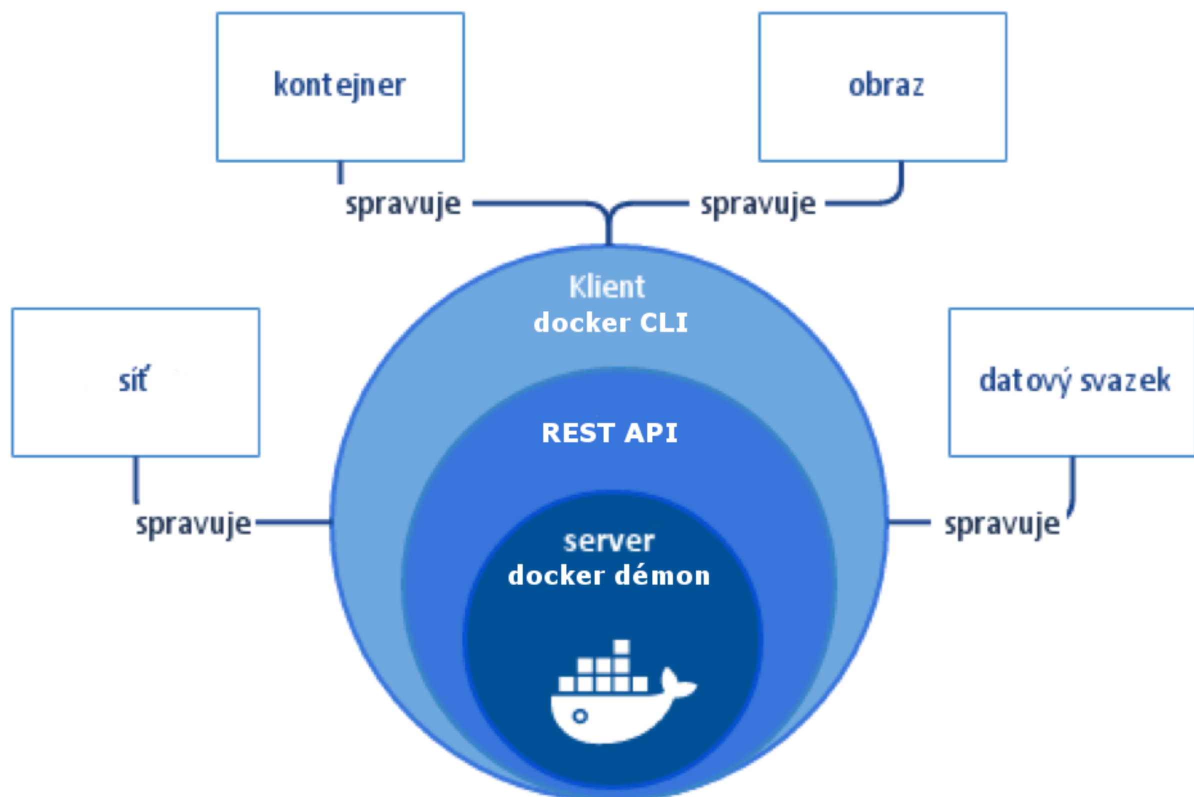
- testování aplikací,
- zjednodušené nasazování aplikací – ať už se jedná o lokální vývojové prostředí, datové centrum nebo cloud.

Docker Engine

Docker Engine je klient-server aplikace, která se skládá z těchto základních částí [14]:

- server, který představuje dlouho běžící program, tzv. démon,
- REST API, které specifikuje rozhraní, které programy mohou používat pro komunikaci s démonem,
- rozhraní příkazové řádky (CLI = komand line interface).

CLI používá Docker REST API k ovládání a interakci s démonem skrze skripty nebo příkazy CLI. Mnoho Docker aplikací používá základní API a CLI. Démon vytváří a spravuje Docker objekty, jako jsou obrazy, kontejnery, sítě a svazky.



Obrázek 4 Architektura Docker Engine. Zdroj: [14]

Technologii Docker je možné nativně provozovat na následujících platformách [14]:

- Linux
  - libovolná distribuce s verzí jádra 3.10 a novější,
  - pro většinu distribucí jsou dostupné dodatečné instrukce pro instalaci,
- Microsoft Windows
  - Windows Server 2016,
  - Windows 10,
- Cloud
  - Amazon EC2,
  - Google Compute Engine,
  - Microsoft Azure,
  - Rackspace.

Společnost Docker nabízí produkty, pomocí kterých je možné vytvářet a spouštět kontejnery i na operačních systémech Windows (Docker for Windows) a macOS (Docker for Mac).

3.2.1 Obrazy

Obraz (angl. Image) je šablona pouze pro čtení s instrukcemi pro vytvoření kontejneru [11]. Často bývá obraz založen na jiném obrazu s několika dalšími úpravami. Například je možné sestavit obraz, který je založený na Ubuntu obrazu, ale nainstaluje webový server a aplikaci, stejně jako údaje o konfiguraci, která je potřebná pro aplikaci. Je možné vytvořit vlastní obraz nebo použít některý z obrazů vytvořených ostatními autory ve veřejném repositáři. Pro vytvoření vlastního obrazu je potřeba vytvořit soubor Dockerfile [15], který obsahuje definované kroky nutné k vytvoření a spuštění obrazu. Instrukce v tomto souboru představuje jednu vrstvu v obrazu. V případě, že je potřeba udělat více kroků, například nainstalovat více balíčků, je vhodné příkaz řetězit pod sebe pomocí zpětného lomítka. Pokud se změní Dockerfile a obraz se opětovně sestavuje, sestavují se znovu pouze ty vrstvy, které byly změněny. Tato část dělá obrazy nenáročné, malé a rychlé v porovnání s ostatními virtualizačními technologiemi.

3.2.2 Kontejnery

Kontejner je spuštěná instance obrazu. Kontejnery je možné vytvářet, spouštět, zastavovat, přesouvat nebo mazat pomocí Docker API nebo CLI nástrojů. Kontejner je možné připojit k jedné nebo více sítím, připojit k němu úložiště nebo vytvářet nové obrazy na základě aktuálního stavu kontejneru [16]. Ve výchozím nastavení je kontejner relativně dobře izolován od ostatních kontejnerů a jeho hostitelského počítače. Tato izolovatelnost může být dle požadavků jednoduše spravována. Kontejner je definován jeho obrazem a konfigurací poskytnutou při vytvoření a spuštění. Po odstranění kontejneru zmizí všechny změny stavu, které nejsou uloženy v trvalém úložišti.

3.2.3 Služby

Služby umožňují škálovat kontejnery napříč několika démony, kteří pracují dohromady jako roj (angl. Swarm) [17]. Démon každého členu roje komunikuje s ostatními pomocí Docker API. Služby umožňují definovat požadovaný stav jako například počet replik dané služby, který musí být kdykoli k dispozici. Zákazníkovi se Docker služba jeví jako jediná aplikace.

3.2.4 Swarm

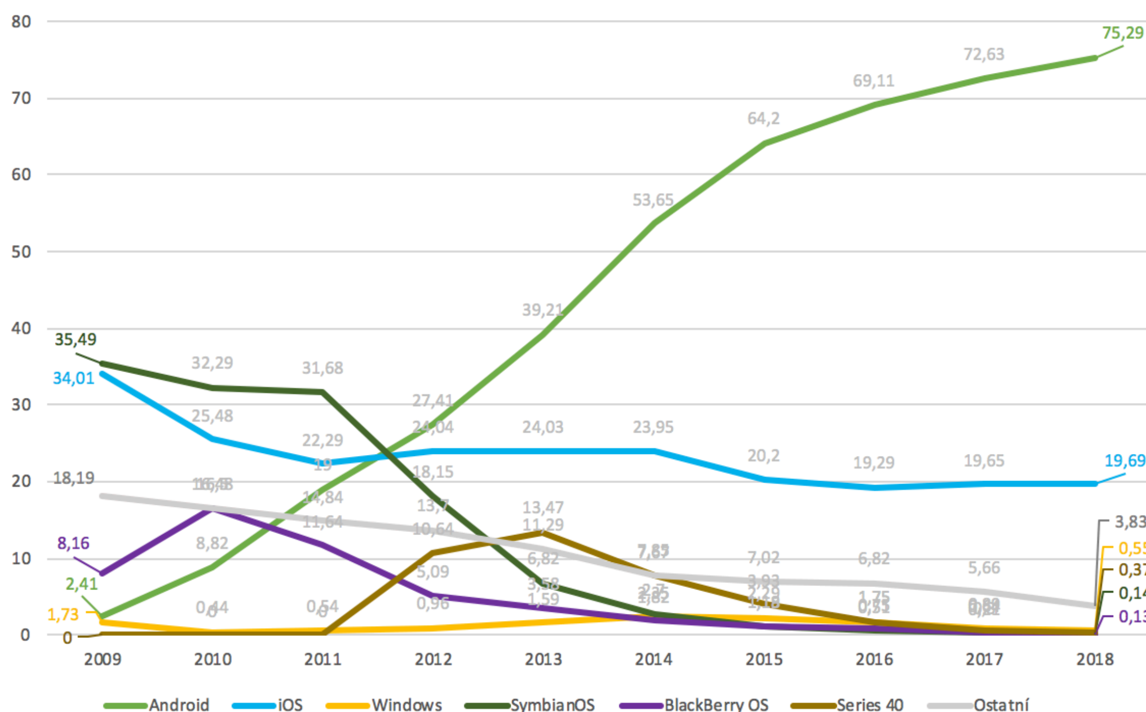
Swarm je nástroj pro správu a orchestraci clusteru [18]. Cluster se skládá z několika Docker hostitelů, kteří běží v módu roje (swarm mode). Tito hostitelé mohou mít roli manažera (manager), pracovníka (worker) nebo vykonávat obě role. Při vytvoření služby se definuje její optimální stav (počet replik, nastavení sítě a úložiště, vystavené porty a další). Docker pracuje tak, aby bylo dosaženo požadovaného stavu. To znamená, že v případě nedostupnosti uzlu, Docker přeplánuje úlohy jinému dostupnému uzlu [17]. Uzel je instance Docker Enginu, která je součástí roje. Jeden nebo více uzlů je možné provozovat na jednom počítači nebo cloudové službě, ale nasazení roje do produkčního prostředí běžně zahrnuje Docker uzly distribuované přes více fyzických a cloudových počítačů. Úloha je běžící kontejner, který je součástí služeb roje a je spravován manažerem roje, na rozdíl od samostatného kontejneru. Podle [19] je jednou z klíčových výhod služeb roje oproti samostatným kontejnerům je možnost upravení konfigurace včetně sítí a datových svazků, které jsou ke službě připojeny bez potřeby manuálně restartovat

službu. Docker aktualizuje konfiguraci, zastaví kontejnery se zastaralou konfigurací a vytvoří nové úlohy, které odpovídají požadované konfiguraci.

Když Docker běží v módu roje, je možné i nadále spouštět samostatné kontejnery na libovolném hostiteli, který je součástí roje. Klíčový rozdíl mezi samostatnými kontejnery a službami roje je ten, že pouze manažer roje může spravovat roj, zatímco samostatné kontejnery mohou být spuštěny na jakémkoli démonu [19].

4 Mobilní platformy

S vývojem mobilních zařízení úzce souvisí i vývoj mobilních operačních systémů. Během několika let vzniklo mnoho operačních systémů, jako například Symbian, Windows CE, Windows Mobile, Windows Phone, BlackBerry OS, iOS nebo Android. Podle [20] je na Obrázek 5 zobrazen podíl mobilních operačních systémů od roku 2009 do současnosti.



Obrázek 5 Podíl mobilních operačních systémů od roku 2009 do 2018. Zdroj: [20]

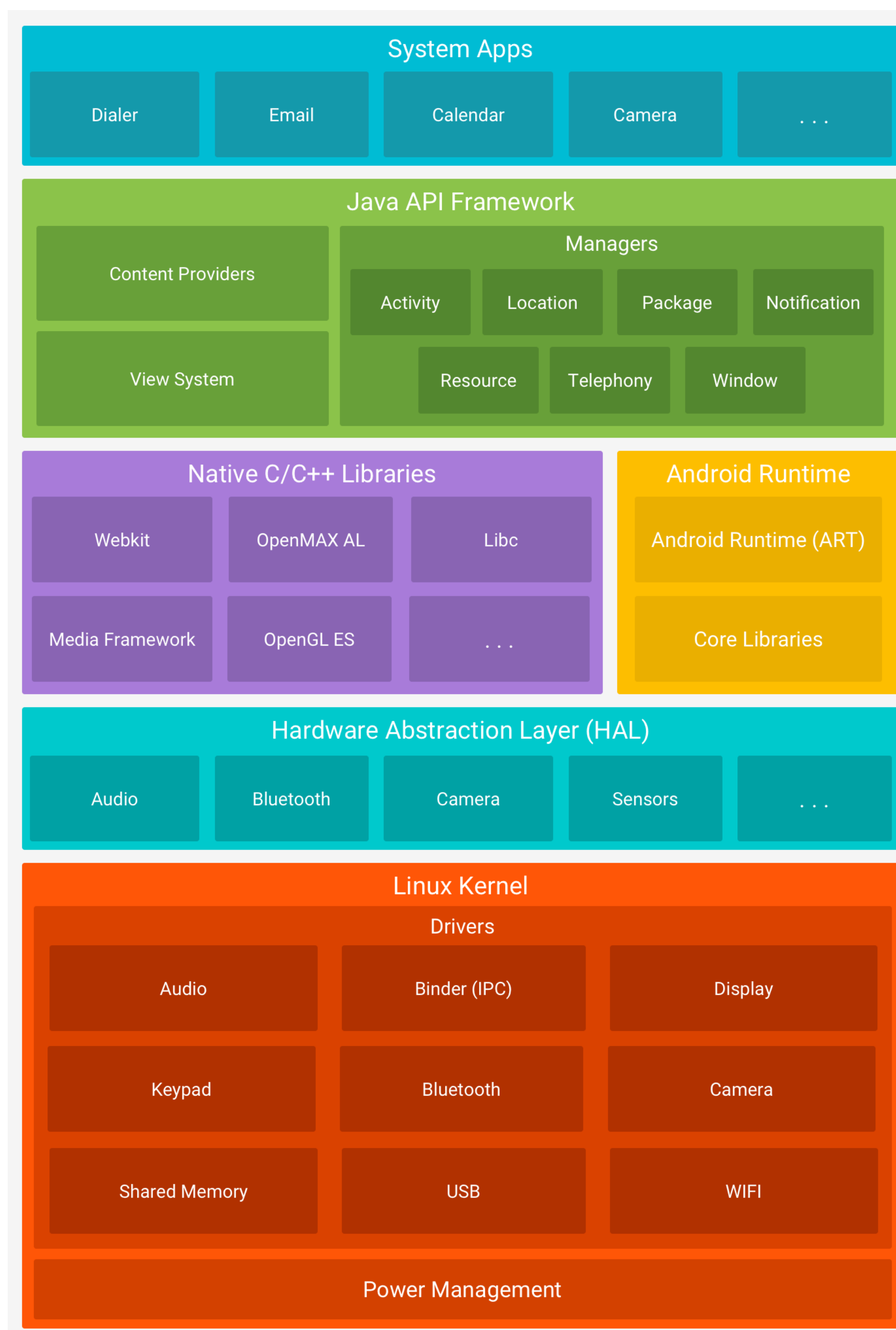
Z výsledků grafu je očividné, že během let se hlavními platformami mobilního trhu staly operační systémy Android (75,29 %) a iOS (19,69 %). S takovou procentuální uživatelskou základnou má význam vytvářet mobilní aplikace především pro tyto dvě platformy.

4.1 Android

Android je open-source operační systém založený na linuxovém jádru a je jedním z nejrozšířenějších mobilních operačních systémů dnešní doby [20]. Aktuální produkční verze operačního systému je podle oficiálního Android Git repositáře 8.1.0, která pro veřejnost vyšla 5. prosince 2017 [21]. O vývoj operačního systému se stará společnost Google a další firmy sdružené v Open Handset Alliance.

4.1.1 Architektura operačního systému

Android je sada softwarových komponent založených na platformě Linux vytvořený pro širokou škálu zařízení jako jsou mobilní telefony a tablety, televize, automobily, ale také nositelná zařízení v podobě fitness náramků a chytrých hodinek [22]. Obrázek 6 zobrazuje hlavní komponenty platformy Android.



Obrázek 6 Hlavní komponenty platformy Android. Zdroj: [23]

Linux Kernel

Základem platformy Android je linuxové jádro. Jádro tvoří abstraktní vrstvu mezi softwarem ve vyšších vrstvách a používaným hardwarem. Použití linuxového jádra umožňuje Androidu využívat klíčových funkcí zabezpečení a výrobcům vyvíjet hardwarové ovladače pro dobře známé jádro [24].

Hardware Abstraction Layer (HAL)

Hardwarová abstraktní vrstva (HAL) poskytuje standardní rozhraní, které vystavuje hardwarové možnosti zařízení do vyšší úrovně rozhraní Java API [25]. Vrstva HAL se skládá z několika knihovních modulů, z nichž každý implementuje rozhraní pro konkrétní typ hardwarové komponenty jako je například Bluetooth, fotoaparát, senzory atp. Když rozhraní API vyvolá přístup k hardwaru zařízení, systém Android načte knihovní modul pro danou hardwarovou komponentu.

Android Runtime

Android Runtime (ART) je aplikační běhové prostředí pro aplikace, které je nástupcem běhového prostředí Dalvik [26]. ART provádí překlad aplikačního bytekódu do nativních instrukcí, které jsou později provedeny běhovým prostředím zařízení. Oproti předchozímu Dalviku je jednou z hlavních výhod zavedení tzv. předběžné kompilace (ahead-of-time), zkompilování celé aplikace do nativního strojového kódu při její instalaci.

Native Libraries

Mnoho základních komponent a služeb systému Android, jako například ART nebo HAL, je sestaveno z nativního kódu, který vyžaduje nativní knihovny napsané v jazycích C a C++ [27]. Platforma Android poskytuje rozhraní Java API pro vystavení některých funkcí těchto knihoven aplikacím. Například je možné přistupovat ke grafickému rozhraní OpenGL ES prostřednictvím Java API rozhraní systému Android pro manipulaci s 2D a 3D grafikou v aplikacích.

Java API Framework

Celá sada funkcí operačního systému Android je dostupná prostřednictvím rozhraní API napsaného v jazyku Java. Tato API rozhraní jsou stavebními bloky potřebnými pro

vytváření Android aplikací [23]. Aplikační rozhraní poskytuje vývojáři přístup k různým funkcionalitám, které je možné použít při vývoji aplikací jako například spouštění služeb na pozadí, používání hardwarových komponent zařízení a další.

System Apps

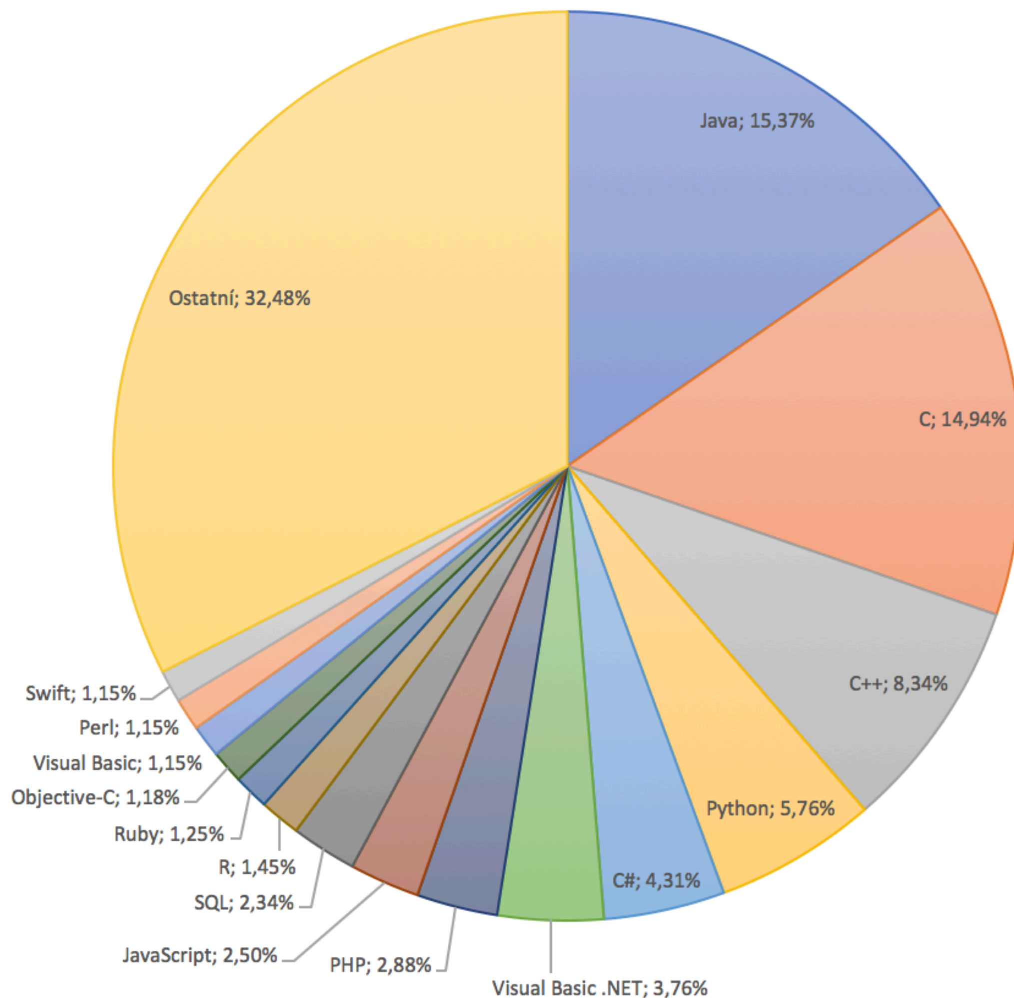
Systémové aplikace jsou předinstalované vývojáři operačního systému nebo výrobcem zařízení. Je to základní sada aplikací pro přístup k volání a SMS zprávám, elektronické poště, prohlížení internetu, poslech hudby a mnoho dalších funkcionalit. Tyto aplikace zůstávají v zařízení i po obnovení do továrního nastavení. Vedle těchto aplikací je možné nainstalovat uživatelské aplikace, kterými jsou jakékoliv aplikace nainstalovány uživatelem z různých zdrojů jako je obchod Google Play nebo například vlastní vytvořená aplikace.

4.1.2 Vývoj pro platformu Android

Pro vytváření nativních Android aplikací je možné využít programovací jazyk Java nebo relativně mladý jazyk Kotlin, který byl 17. května 2017 oznámen jako oficiálně podporovaný programovací jazyk pro vývoj aplikací [28]. Vývoj pro tento operační systém je možný na všech platformách.

Java

Java je vysokoúrovňový objektově orientovaný programovací jazyk běžící nad Java Virtual Machine (JVM), který byl vyvinut a představen firmou Sun Microsystems 23. května 1995 [29]. Tento jazyk je celosvětově používaný a několik let je jedním z nejpoužívanějších programovacích jazyků [30]. Java má širokou škálu využití – od mobilních aplikací, přes desktopové a serverové aplikace až po využití na webu a mnoho dalších. Na Obrázek 7 je podle [30] zobrazena oblíbenost programovacích jazyků.



Obrázek 7 Graf oblíbenosti programovacích jazyků v 06/2018. Zdroj: [autor] [30]

Kotlin

Kotlin je staticky typovaný programovací jazyk pro JVM, Android a JavaScript. Je vyvíjen společností JetBrains jako open-source projekt od roku 2010 [31]. Kotlin klade důraz především na následující [32]:

- interoperabilita – je možné nadále využívat knihovny napsané v jazyku Java a je samozřejmostí v rámci jednoho projektu používat kombinaci jazyků jako například Java a Kotlin nebo JavaScript a Kotlin
- bezpečnost – Kotlin je tzv. „null safe" jazyk, což znamená, že pracuje s možnými null situacemi v době kompilace, aby bylo zabráněno výjimkám za běhu kódu

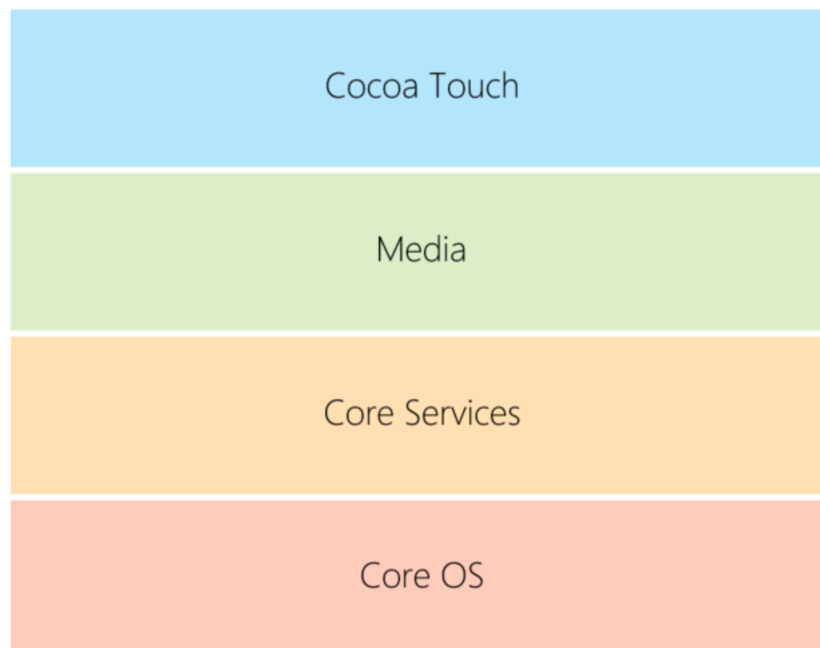
- jednoduchost – zápis zdrojového kódu je mnohem stručnější než například u jazyku Java, díky využití mnoha konceptů z funkcionálního programování pro zjednodušení zápisu kódu
- rozšíření – Kotlin umožňuje vytvářet rozšíření stávajících tříd pro přidání nové funkcionality i bez přístupu k daným zdrojovým kódům

4.2 iOS

iOS je mobilní operační systém vytvořený a vyvíjený společností Apple výhradně pro její zařízení. Operační systém byl oficiálně představen spolu s první generací zařízení iPhone 29. června 2007 [33].

4.2.1 Architektura operačního systému

Apple popisuje sadu rozhraní a technologií, které jsou aktuálně implementovány v rámci operačního systému iOS jako řada vrstev. Každá z těchto vrstev je tvořena řadou různých rozhraní, které mohou být použity a začleněny do vytvářených aplikací. Architektura iOS může být rozdělena do čtyř odlišných vrstev [34]:



Obrázek 8 Vrstvy iOS architektury. Zdroj: [35]

Core OS

Většina funkcí poskytovaných vyššími vrstvami je postavena na vrstvě Core OS a jejích nízkoúrovňových funkcích. Vrstva Core OS poskytuje mnoho rozhraní, které mohou aplikace používat přímo jako například rozhraní Core ML pro integraci strojového učení, Image I/O pro práci s obrazovými formáty a další. Tato vrstva také zapouzdřuje prostředí jádra a nízkoúrovňové unixové rozhraní ke kterým aplikace nemají přístup ze zjevných bezpečnostních důvodů. Nicméně díky knihovně libSystem, která implementuje standardní knihovny jazyka C a mnoho API rozhraní například pro práci s POSIX vlákny atp. [36].

Core Services

Tato vrstva poskytuje abstrakci nad službami poskytovanými ve vrstvě Core OS a umožňuje přístup k základním službám a funkcionalitám operačního systému iOS. Jedním z nejvíce vítaných dodatků k vrstvě Core Services je automatické počítání referencí (ARC = Automatic Reference Counting), kterou například využívá programovací jazyk Swift pro sledování a správu paměti aplikace. [37].

Media

Vrstva Media obsahuje velké množství rozhraní pro práci s multimédii, které je možné například použít zařízeních pro nahrávání a přehrávání zvuku nebo videa, pokročilou podporu animací nebo pro práci s 2D a 3D grafikou [34].

Cocoa Touch

Cocoa Touch je nejvyšší vrstvou iOS architektury. Obsahuje některá klíčová rozhraní, na které se nativní iOS aplikace spoléhají, přičemž nejvýznamnějším z nich je rozhraní UIKit. Tato vrstva definuje základní aplikační infrastrukturu a poskytuje řadu důležitých technologií jako je multitasking a dotykový vstup. iOS aplikace se silně spoléhají na rozhraní UIKit a nemohou fungovat bez spojení s UIKit a Foundation rozhraními.
Rozhraní UIKit je přizpůsobeno platformě iOS a je ekvivalentem k AppKit platformy OS X. UIKit poskytuje infrastrukturu pro grafické, událostmi řízené aplikace [38]. Dále se stará o ostatní základní aspekty, které jsou specifické pro platformu iOS jako jsou push notifikace, multitasking a přístupnost.

Vrstva Cocoa Touch poskytuje vývojářům velké množství vysokoúrovňových funkcí jako je Auto Layout, tisk, rozpoznávání gest a podpora dokumentů. Kromě rozhraní UIKitu obsahuje mimo jiné i rozhraní pro práci s mapou (MapKit), událostmi v kalendáři a upomínkami (EventKit) nebo specializované rozhraní pro kompozici emailů a SMS zpráv (MessageUI) [39].

4.2.2 Vývoj pro platformu iOS

Vývoj iOS aplikací má oproti platformě Android specifické požadavky na hardwarové prostředky. Jedinými zařízeními podporovanými společností Apple jsou zařízení s operačním systémem macOS. Použitelnými programovacími jazyky jsou Objective-C nebo jeho nástupce Swift.

Objective-C

Objective-C byl prvním z jazyků pro vytváření aplikací pro platformy macOS a iOS. Je to rozšíření jazyku C o objektově orientované schopnosti a dynamické běhové prostředí. Od jazyku C dědí syntaxi, primitivní typy, řídící struktury a přidává definici tříd a metod. Aktuální verze tohoto jazyku je 2.0, která byla představena v roce 2007 [40].

Swift

Swift je open-source programovacím jazykem vyvinutý společností Apple především na vývoj aplikací pro jejich platformy, který je nástupcem Objective-C. Tento jazyk přebírá bezpečné programovací vzory a přidává moderní funkce pro snadnější a flexibilnější programování [41] [42]. Mezi přednosti Swiftu patří [43]:

- otevřenost – vzhledem k jeho otevřenosti si Swift získal velkou podpůrnou komunitu uživatelů a mnoho nástrojů třetích stran,
- bezpečnost – jeho syntaxe nabádá ke psaní čistého a konzistentního kódu, která může být v některých případě velmi striktní,
- rychlost – Swift byl vytvořen s ohledem výkonnost – 2,6x rychlejší než Objective-C a 8,4x rychlejší než Python,
- žádanost – ke 29. červnu 2018 je podle [30] 15. nejpopulárnějším jazykem a na serveru GitHub je používán téměř 11 tisíci repositáři [44].

5 Multiplatformní vývoj

Vývoj mobilních systémů zaznamenal velký pokrok od jednoduchých komunikačních zařízení po vysoce výkonná výpočetní zařízení. V dnešní době jsou mobilní telefony schopny vykonávat složité a kritické úlohy, které vyžadují vyšší výpočetní schopnosti, vysokou dostupnost a mnoho dalších. S vývojem chytrých telefonů jsou tato zařízení řízena výkonnými operačními systémy, které umožňují uživateli přidávat a odebírat aplikace, které vykonávají stejné úlohy jako osobní počítače. V závislosti na zařízení, existují různé operační systémy, které mají vlastní standardy, programovací jazyky, vývojové nástroje a místa, ze kterých mohou uživatelé nakupovat a stahovat aplikace. Tato různorodost je výzvou pro softwarové vývojáře – každá z platforem má obrovskou základnu uživatelů představující potencionální zákazníky nových aplikací. Podniková strategie může vyžadovat, aby vývojáři cílili na širší skupinu uživatelů a vývoj aplikací pouze pro jednu platformu by mohl znamenat ztrátu velkého počtu potencionálních klientů. Vývoj pro každou platformu zvlášť vyžaduje, aby se specifické části vývojového cyklu softwaru prováděly několikrát, což se může být nadbytečné a nákladné [45].

Efektivním přístupem, jak lze tento problém vyřešit, může být zavedení multiplatformních vývojových nástrojů, které nabízí řešení podle principu „napiš jednou, spusť kdekoliv". Multiplatformní vývoj mobilních aplikací tedy znamená vytváření mobilních aplikací, které jsou kompatibilní s více mobilními operačními systémy. Tento přístup je založen především na znovupoužitelnosti jednou napsaného aplikačního kódu, což ušetří práci a čas vývojářům aplikace, kteří by v opačném případě museli jednou vytvořený kód napsat znovu pro ostatní platformy. Aplikace business logiky pro ostatní platformy nemusí být složitá, ale znamená vyšší časové a cenové náklady na aplikaci, které rostou s každou cílovou platformou.

5.1 Přístupy k vývoji mobilních aplikací

5.1.1 Native-like aplikace

React Native

React Native je JavaScriptový framework pro vytváření nativních aplikací pro Android a iOS. Je založen na JavaScriptové knihovně React od společnosti Facebook, která je určena

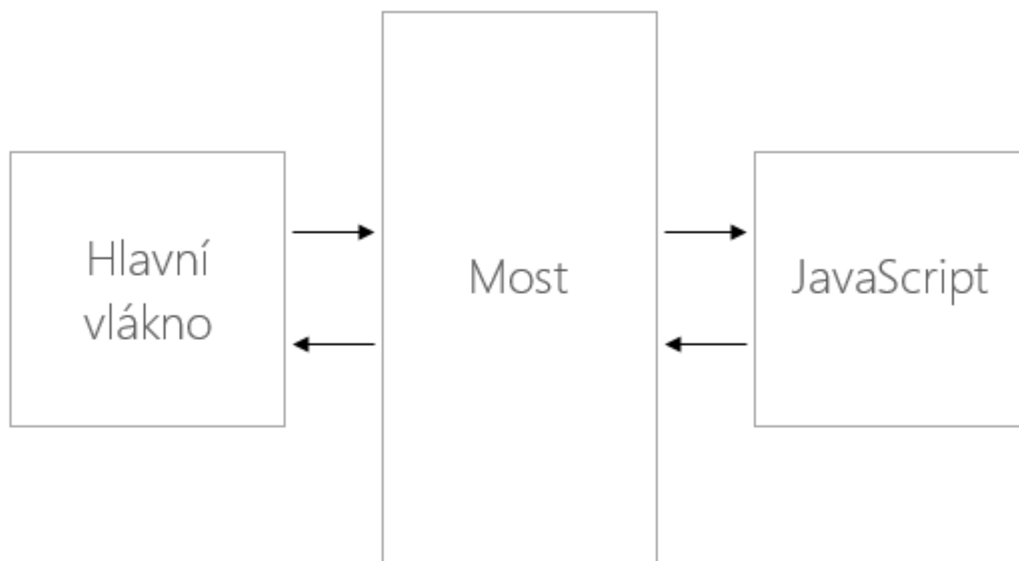
pro vytváření uživatelských rozhraní, ale místo cílení na webové prohlížeče se zaměřuje na mobilní platformy [46]. Pro webové React vývojáře je snadné vytvářet mobilní aplikace, které vypadají a působí „nativním“ dojmem, protože pracují v již známém prostředí JavaScriptové knihovny. Stejně jako React pro web, využívá i React Native pro psaní mobilních aplikací kombinaci jazyků JavaScript a jeho rozšíření JSX. JSX je krok preprocesoru, který přidává syntaxi značkovacího jazyku XML do JavaScriptu. Stejně jako XML, i JSX tagy mají název, atributy a potomky. Pro psaní aplikačního kódu je vhodné využívat JSX pro přehlednější a elegantnější syntaxi kódu, ale samozřejmě není jeho používání podmínkou. Následující část kódu je ukázka použití JSX a kódu po jeho kompilaci.

```
<Button color="yellow" shadowSize={2}>
     Release the ducks!
</Button>

React.createElement(
     Button,
     { color: 'yellow', shadowSize: 2 },
     'Release the ducks!'
)
```

V každé React Native aplikaci jsou dvě důležitá vlákna [47]. První z nich je hlavní vlákno, které běží i ve standardních nativních aplikacích a zabývá se zobrazováním prvků uživatelského rozhraní a zpracovává uživatelské vstupy. Druhé vlákno je specifické pro React Native. Jeho úkolem je vykonávat JavaScriptový kód v odděleném enginu. JavaScript pracuje s business logikou aplikace a také definuje strukturu a funkcionality uživatelského rozhraní. Tato dvě vlákna mezi sebou nikdy nekomunikují napřímo a nikdy se navzájem neblokují. Interakce mezi těmito vlákny probíhá pomocí tzv. mostu (angl. Bridge), který je v jádru React Native frameworku. Most má tři důležité vlastnosti [48]:

- asynchronnost – umožňuje asynchronní komunikaci mezi vlákny,
- dávkové zpracování – přenos zpráv z jednoho vlákna do druhého řeší optimálně,
- serializovatelnost – obě vlákna nikdy nesdílí nebo nepracují se stejnými daty, namísto toho si vyměňují serializované zprávy.



Obrázek 9 Komunikace mezi vlákny. Zdroj: [48]

Pomocí mostu React Native vyvolává nativní API pro vykreslování – v Objective-C pro iOS a v Javě pro OS Android. Aplikace tedy bude využívat skutečné UI komponenty, bude vypadat a působit jako jakákoliv nativní aplikace. React Native také nabízí JavaScriptové rozhraní pro práci s API jednotlivých platforem, takže aplikace mohou přistupovat k fotoaparátu, lokaci uživatele, senzorům daného zařízení apod [49]. Aktuálními podporovanými platformami pro vývoj React Native aplikací jsou pouze Android a iOS.

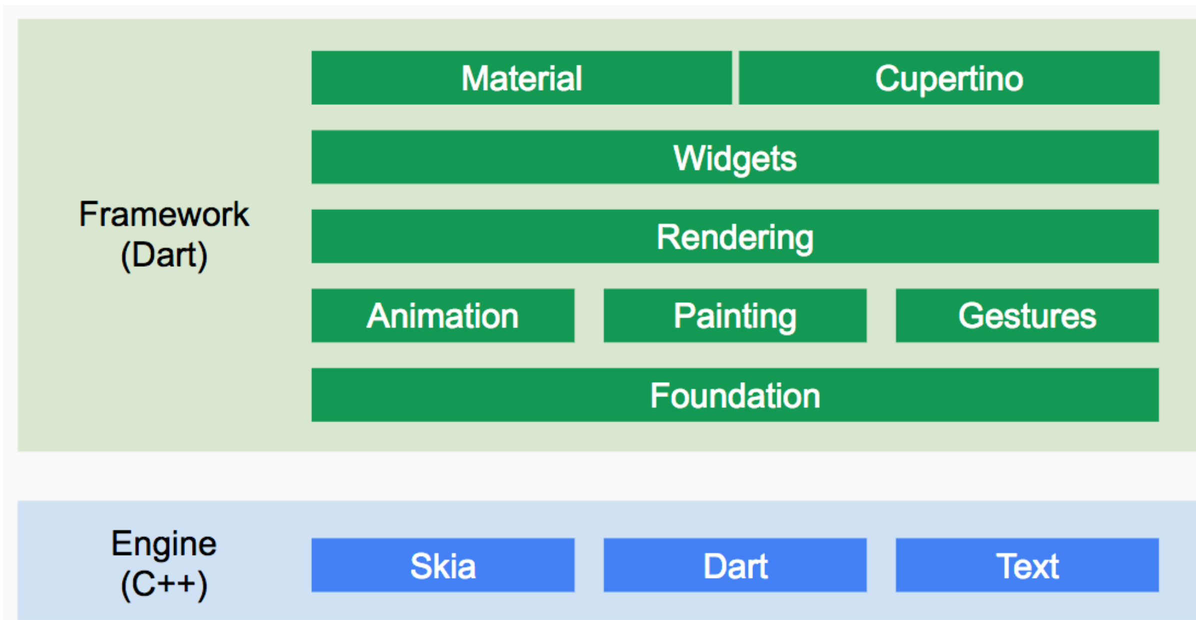
Velkou výhodou této technologie oproti ostatním metodám multiplatformního vývoje je vykreslování za pomocí standardních vykreslovacích API dané platformy. Většina ostatních technologií, jako Cordova, Ionic a další, používá kombinaci JavaScriptu, HTML a CSS stylů, které typicky vykreslují pomocí komponenty WebView, která slouží k zobrazování obsahu na webových stránkách [50]. Ačkoliv tento přístup může fungovat, přichází s ním i nevýhody, zejména pokud jde o výkon. Dále obvykle nemají přístup k nativním prvkům uživatelského rozhraní hostitelské platformy, a i když se tyto technologie snaží napodobit nativní prvky, výsledek není vždy stejný a mnohdy skýtá obrovské množství úsilí a času. Oproti tomu React Native překládá strukturu a vytvořené prvky do skutečných nativních komponent podle toho na které se nachází platformě. Další užitečnou vlastností React Native je kombinování funkcionalit napsaných v JavaScriptu s částmi kódu napsanými ve standardních jazycích jednotlivých platforem.

Flutter

Flutter je SDK od společnosti Google pro vytváření nativních mobilních aplikací pro platformy Android a iOS [51]. Podobně jako React Native (RN), Flutter využívá vlastní vysoce výkonný vykreslovací engine pro zobrazování widgetů (obdobné jako komponenty u RN). Vývojovým jazykem je Dart – moderní, rychlý a objektově orientovaný jazyk [52]. Pro vývojáře, kteří mají předchozí zkušenosti s vývojem Android aplikací, je použití Dartu jednodušší, právě díky objektově orientovanému přístupu.
Flutter SDK nabízí následující [53]:

- silně optimalizovaný 2D vykreslovací engine,
- moderní framework inspirovaným Reactem,
- bohatou sadu widgetů pro Android a iOS,
- rozhraní pro jednotkové a integrační testy,
- nástroje pro spouštění testů na platformách Windows, Linux a Mac,
- nástroje příkazové řádky pro vytváření, sestavování, testování a kompilování,
- interoperabilitu s výchozími programovacími jazyky daných platforem.

Obrázek 10 představuje architekturu Flutter SDK. Flutter je sestaven pomocí programovacích jazyků C, C++, Dart a Skia (2D vykreslovací engine). Framework se skládá z knihoven pro vytváření stylů specifických pro platformy Android (Material Design) a iOS (Cupertino), široké škály předvytvořených widgetů, rozhraní pro vykreslování, animace a mnoho dalších [53].



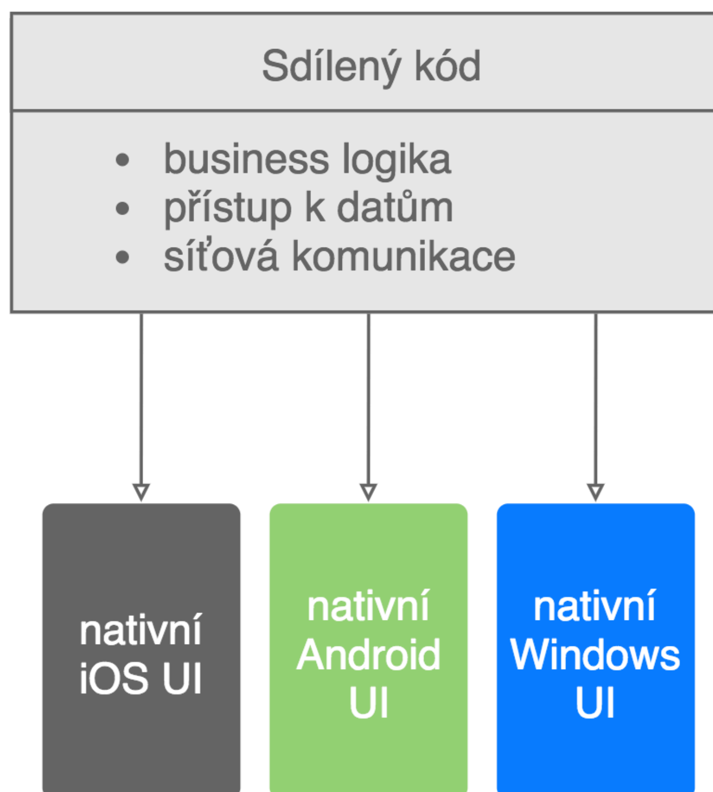
Obrázek 10 Architektura Flutter SDK. Zdroj: [53]

Xamarin

Xamarin je nástroj pro multiplatformní vývoj aplikací na mobilní operační systémy Android, iOS a Windows vyvíjený společností Microsoft [54]. Tato platforma byla vytvořena vývojáři projektu Mono [55], jako open-source projekt založený na .NET frameworku a společnost Xamarin vznikla 16. května 2011 [56]. Xamarin byl původně vytvořen jako komerční projekt do roku 2016 [57], kdy byla firma odkoupena společností Microsoft.

Xamarin pro vytváření aplikací pro všechny mobilní platformy používá programovací jazyk C#. Aplikace jsou nativně kompilované, stejně jako při použití technologií React Native nebo Flutter, což umožňuje vytvářet vysoce výkonné aplikace, které vypadají a působí stejným dojmem jako ty nativní. Tato technologie je založená na .NET frameworku, která umožňuje jazyku C# využít různé užitečné .NET vlastnosti jako lambda výrazy, integrovaný dotazovací jazyk (LINQ) nebo asynchronní programování.

Společná část aplikačního kódu jako business logika, přístup k databázi a síťová komunikace je sdílena všem cílovým platformám [58]. Xamarin umožňuje vytvořit vrstvu s uživatelským rozhraním specifickým pro každý operační systém. Díky tomu, že multiplatformní aplikace vytvořená pomocí Xamarinu vypadá zcela jako nativní na kterémkoliv zařízení, poskytuje lepší uživatelskou zkušenost ve srovnání s generickými hybridními aplikacemi.



Obrázek 11 Sdílení společné části kódu se všemi cílenými platformami. Zdroj: [58]

5.1.2 Progresivní webové aplikace

Progresivní webové aplikace (angl. Progressive Web Apps = PWA) jsou vyvinuté pomocí řady specifických technologií a standardních vzorů, které jim umožňují využívat funkce webových i nativních aplikací [59]. Jejich hlavními výhodami je jejich dostupnost (je mnohem jednoduší a rychlejší navštívit webovou stránku než nainstalovat aplikaci) a možnost sdílení webové aplikace prostřednictvím odkazu. Díky tomu, že jsou aplikace hostovány na webu, je mnohem jednodušší publikovat nové změny než například u nativních aplikací, a to z důvodu zdlouhavého schvalovacího procesu obchodů jednotlivých platforem.

Vytvoření PWA zahrnuje více technologií. Reprezentují novou filozofii pro vytváření webových aplikací zahrnující některé specifické vzory, rozhraní a další funkce. Z prvního pohledu nelze webovou aplikaci posoudit a označit ji za PWA. Aplikace může být považována za PWA až v případě, kdy splní určité požadavky nebo implementuje sadu daných funkcí jako je například offline funkcionalita, instalovatelnost, jednoduchá synchronizovatelnost, možnost odesílat push notifikace a další.

Mimo to existují nástroje, které měří komplexnost aplikace v procentech (např. populární aplikace Lighthouse od společnosti Google [60]). Implementováním různých technologických funkcionalit je možné aplikaci udělat více progresivnější a tím, získat větší skóre v měřící aplikaci Lighthouse. Každá PWA by se měla snažit dodržovat následující klíčové principy [61]:

- obsah lze vyhledat prostřednictvím vyhledávačů,
- aplikaci je možné přidat na domovskou obrazovku zařízení,
- možnost sdílet aplikaci prostřednictvím URL,
- nezávislost na síťovém připojení,
- progresivnost – na základní úrovni je použitelná pro starší prohlížeče, kompletní funkcionalita na nejnovějších prohlížečích,
- umožňuje odesílat notifikace v případě, že je dostupný nový obsah,
- responzivita,
- připojení mezi uživatelem a aplikací je zabezpečené.

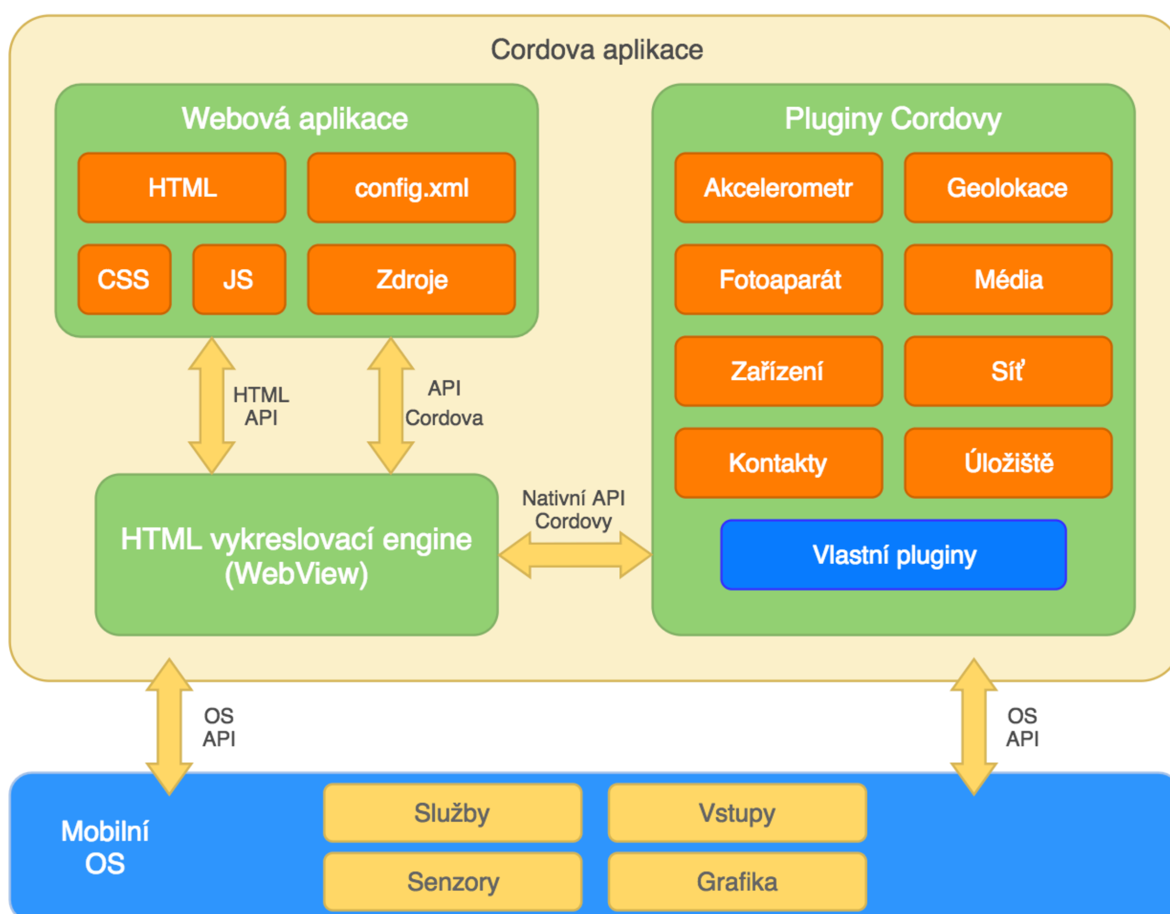
Tyto klíčové principy jsou současně i výhodami progresivních webových aplikací.

5.1.3 Hybridní aplikace

Posledním typem přístupu multiplatformních mobilních aplikací jsou hybridní aplikace. Tyto aplikace jsou jako ostatní aplikace, které je možné nalézt v telefonu. Lze je vyhledat v obchodech daných platforem, nainstalovat je, využívat je ke kontaktu s ostatními uživateli přes sociální sítě, pořizovat fotografie, sledovat svůj zdravotní stav a mnoho dalších [62]. Stejně jako webové stránky na internetu, hybridní aplikace jsou vytvářeny pomocí technologií HTML, CSS a JavaScriptu. Hlavním rozdílem je, že hybridní aplikace jsou hostovány uvnitř nativní aplikace, která využívá WebView mobilní platformy. Toto aplikaci umožňuje přistupovat k funkcionalitám zařízení, jako je například akcelerometr, fotoaparát, kontakty a jiné. K těmto funkcím je často z mobilního prohlížečů omezen přístup. Podobně jako u PWA, je i u hybridních aplikací obtížné posoudit, jakým způsobem byla aplikace vytvořena a jestli se opravdu jedná o nativní nebo hybridní aplikaci. Správně napsaná hybridní aplikace by neměla vypadat nebo se chovat jinak než její nativní ekvivalent. Mezi velmi známé technologie hybridních aplikací patří například Apache Cordova a Ionic Framework.

Apache Cordova

Apache Cordova je open-source framework pro vývoj mobilních aplikací [63]. Umožňuje použít standardní webové technologie jako HTML5, CSS3 a JavaScript pro multiplatformní vývoj aplikací. Aplikace jsou spouštěny uvnitř aplikačních obalů (wrapperů) cílených na každou platformu a spoléhají na standardní API pro přístup ke všem možnostem zařízení, jako jsou senzory, data, stav sítě a další. Na Obrázek 12 je zobrazena architektura Cordova aplikace.



Obrázek 12 Architektura Cordova aplikace. Zdroj [63]

Webview s podporou technologie Cordova může poskytovat aplikaci s kompletním uživatelským rozhraním. Na některých platformách může být také součástí větší hybridní aplikace, která kombinuje WebView s nativními komponentami aplikace. Samotná aplikace je implementována jako webová stránka. Ve výchozím nastavení je pro její spuštění nezbytný soubor index.html, který odkazuje na CSS, JavaScript, obrázky,

mediální soubory a další soubory. Aplikace je spouštěna ve WebView uvnitř aplikačního obalu v nativní aplikaci, která je distribuována do jednotlivých obchodů.
Pluginy jsou nedílnou součástí ekosystému Cordovy. Poskytují rozhraní pro Cordovu a nativní komponenty, které komunikují navzájem a vážou se na standardní API zařízení. Toto umožňuje vyvolat nativní kód z JavaScriptu. Projekt Apache Cordova udržuje sadu pluginů [64], které se poskytují přístup k možnostem zařízením, jako je například baterie, fotoaparát, kontakty a mnoho dalších. Kromě základních pluginů existuje mnoho pluginů třetích stran, které poskytují další funkcionality. Tyto pluginy nemusí být dostupné pro všechny platformy.

Ionic

Ionic je framework pro vývoj mobilních aplikací pomocí HTML 5 zaměřený na vytváření hybridních mobilních aplikací [65]. Na Ionic je možné nahlížet jako na frontendový UI framework, který obsluhuje všechny interakce uživatelského rozhraní mobilní aplikace. Funkcionalitu je možné připodobnit k Bootstrap frameworku pro webový vývoj s rozdílem zaměření na podporu široké škály běžných nativních komponent, animací a designu specifického pro jednotlivé platformy. To znamená, že Ionic nabízí velké množství komponent stylovaných stejně jako nativní komponenty, které se nachází v nativních SDK operačních systémů Android a iOS. Vzhledem k tomu, že je Ionic HTML5 framework, potřebuje nativní aplikační obal, jako je například zmíněný Apache Cordova, aby aplikace mohla fungovat jako nativní. Oproti ostatním hybridním technologiím je hlavním cílem Ionicu vývoj nativních nebo hybridních mobilních aplikací na rozdíl od vývoje mobilních webových stránek [66]. Vývojové technologie při použití Ionicu jsou standardní jako při vývoji webových aplikací. Je možné využít technologie HTML, CSS a JavaScript.
Vedle Ionic frameworku byl vytvořen Ionic Native [67]. Ionic Native je velká sada pluginů spravovaných komunitou. Pluginy jsou napsané v jazyku TypeScript a ve spojení s Apache Cordova zjednodušují přidávání libovolné nativní funkcionality do mobilní aplikace.

5.2 Výhody a nevýhody

Multiplatformní nástroje a jejich využití k vývoji aplikací má mnoho pro a proti. Využití těchto technologií může záviset na typu aplikace, schopnostech jednotlivých vývojářů,

rozpočtu a dalších faktorech. Mezi hlavní výhody multiplatformních technologií je možné zařadit následující [68] [69]:

- velká znovupoužitelnost kódu – při správném návrhu a použití vhodné technologie je odlišný zpravidla pouze kód pro uživatelské rozhraní jednotlivých platforem,
- rychlejší a jednodušší vývoj – odpadá nutná znalost vývojových technologií specifických pro každou platformu díky jednotné technologii a společné části kódu,
- konzistentnost všech platforem – sjednocení verzování, přidaných funkcionalit, oprav chyb a další,
- menší velikost vývojového týmu – díky sjednocené vývojové technologii nezáleží, jestli daný vývojář vytváří Android nebo iOS aplikace, tzn. není potřeba více vývojových týmů pro každou platformu zvlášť,
- snížení nákladů potřebných pro vývoj aplikace – jeden vývojový tým obsluhuje všechny platformy,
- větší dosah aplikace na mobilním trhu podle cílených platforem.

I přes spoustu výhod, které tyto technologie v dnešní době nabízí existují i nevýhody, které není možné přehlížet:

- výkon není stejný jako v nativních aplikacích,
- limitace platforem – každá platforma má vlastní specifické vlastnosti a funkce, které nemusí a mnohdy nejsou dostupné u ostatních platforem,
- podpora nových funkcionalit s novými aktualizacemi operačních systémů.

5.3 Závěr

Pro implementovanou multiplatformní aplikaci jsou důležité následující aspekty:

- podpora zvolených platforem,
- výkonnost,
- uživatelská přívětivost – aplikace by měla vypadat a působit jako nativní,
- technologie s velkou uživatelskou základnou pro případnou podporu s možnými problémy při vývoji a následnému udržování aplikace,
- předchozí zkušenosti s některou z technologií.

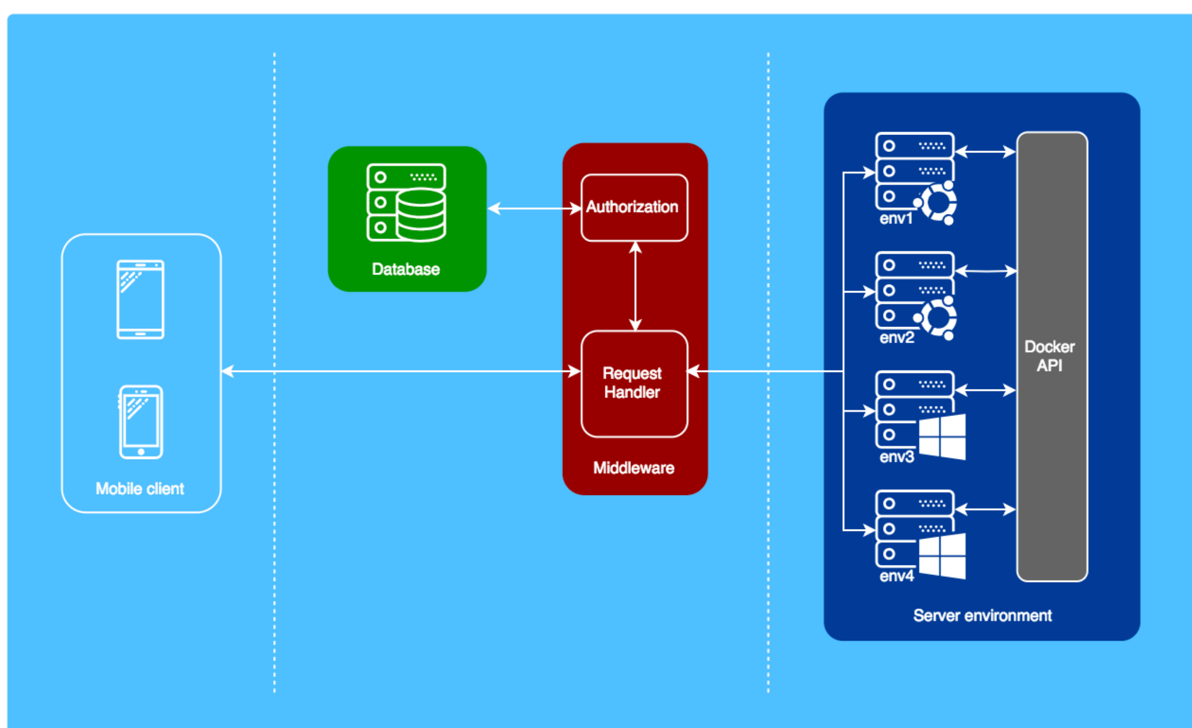
Zatímco všechny vybrané technologie podporují vývoj mobilních aplikací na platformy Android a iOS, ne všechny se chovají nebo působí ve výsledku dojmem nativní aplikace. Po výkonnostní stránce jsou vhodnější technologie, které jsou kompilovány jako nativní aplikace než technologie, které pouze nativních technologií využívají.

Dalším důležitým aspektem bylo zohlednění implementačního jazyka a použité technologie. Vyvíjené řešení bylo vyzkoušeno v testovacím prostředí firmy MoroSystems. Tato firma pro dodávané a interní projekty ve valné většině používá programovací jazyky Java, C++ a JavaScript. Mobilní divize firmy se věnuje pouze vývoji nativních aplikací pomocí výchozích programovacích jazyků daných platforem. Vzhledem k předchozím zkušenostem autora s technologiemi využívající JavaScript byla vybrána technologie React Native, která je vhodná pro budoucí rozvoj a udržování aplikace firmou MoroSystems

6 Návrh aplikace

Tato práce se věnuje především vytvoření mobilního klienta, který bude umožňovat vzdálenou správu serverového prostředí technologie Docker. Vývojáři Dockeru vytvořili nástroj Universal Control Plane, který nabízí podnikové řešení pro správu clusteru, které zahrnuje i správu uživatelů a jejich rolí. Tento nástroj je dostupný pouze pro Enterprise verzi [70], proto bylo navrženo vlastní řešení v podobě třívrstvé architektury s následujícími vrstvami:

- mobilní klient,
- middleware,
- Docker cluster.



Obrázek 13 Architektura komunikace mezi jednotlivými vrstvami na oddělení IMIT. Zdroj: [autor]

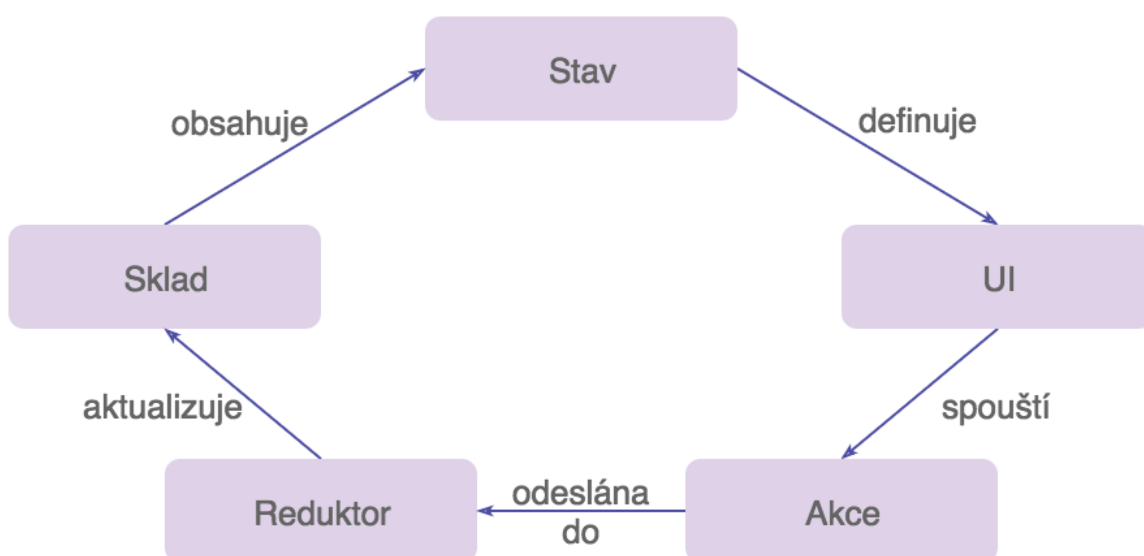
6.1 Mobilní klient

Mobilní klient je představován multiplatformní mobilní aplikací pro operační systémy Android a iOS. Celá aplikace je implementována pomocí programovacího jazyku JavaScript, frameworku React Native a knihovny Redux.

Redux

Redux je JavaScriptová knihovna, která umožňuje jednodušší správu stavu aplikace. Vychází z architektury Flux od společnosti Facebook [71]. Redux se skládá ze tří hlavních částí [72]:

- akce (actions),
- reduktory (reducers),
- sklad (store).

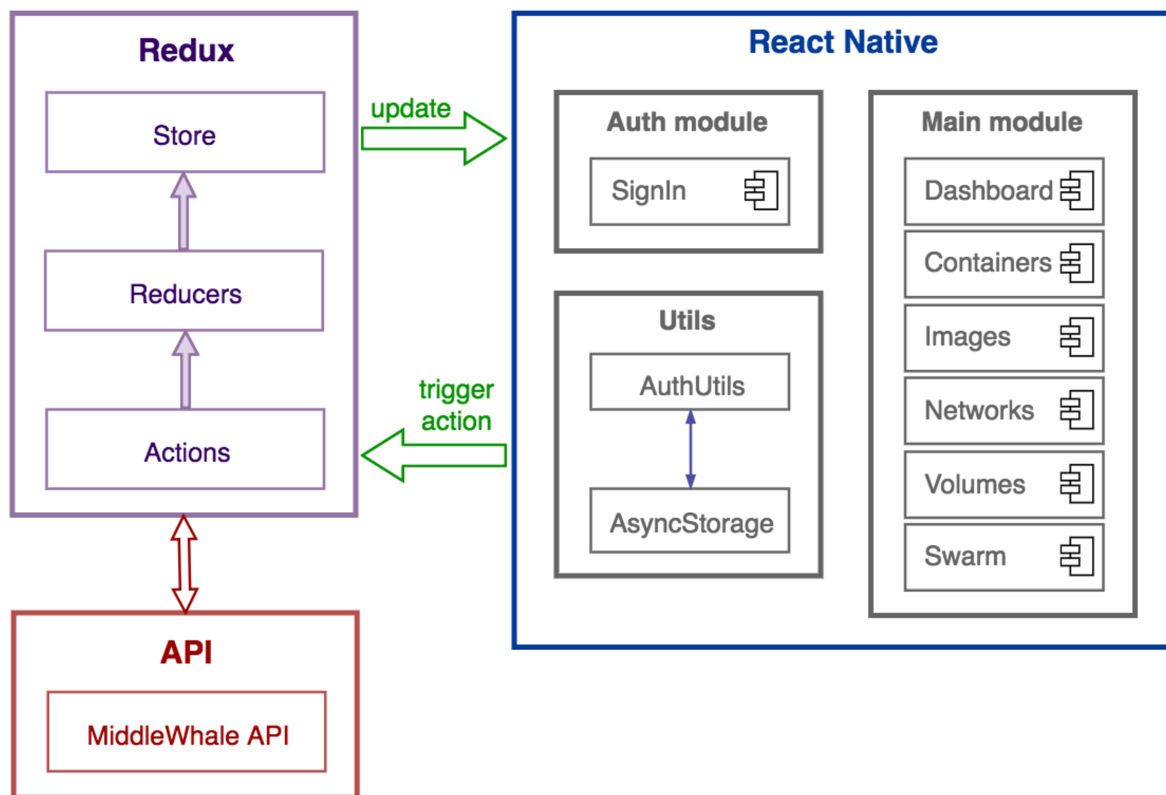


Obrázek 14 Architektura knihovny Redux. Zdroj: [73]

Sklad je kontejner, ve které se udržuje stav celé aplikace. Drží v sobě neměnnou referenci reprezentující stav celé aplikace a může být aktualizována pouze prostřednictvím akcí. Akce obsahují informaci, která je zasílána do skladu. Reprezentují to, jak chceme, aby se stav aplikace změnil. Reduktory jsou funkce, které přijímají aktuální stav, akci, kterou mají provést a jejich výsledkem by měl být nový stav [72].

Na Obrázek 15 je zobrazena architektura mobilní aplikace, která je rozdělena do tří částí. První část architektury je výše zmíněný Redux, který zajišťuje veškerou správu stavu mobilní aplikace. Tato část očekává spuštění jedné z definovaných akcí, kterou iniciuje uživatel v mobilní aplikaci interakcí s uživatelským rozhraním. Podle dané akce budou provedeny následné kroky implementované v definovaných reduktorech, např. komunikace s druhou částí architektury – MiddleWhale API. Druhá část přijímá

požadavky na její REST API zaslané Reduxem a zasílá odpovědi na základě zpracovaných výsledků. Redux dané odpovědi od MiddleWhale API zpracovává, ukládá je do svého skladu a vysílá aktualizaci o změně stavu všem komponentám uživatelského rozhraní, které jsou aktuálně zobrazeny.



Obrázek 15 Architektura mobilní aplikace. Zdroj: [autor]

Poslední a hlavní částí této architektury je struktura samotné aplikace, která je rozdělena do jednotlivých modulů pro větší přehlednost při implementaci. Každý z modulů obsahuje následující strukturu souborů:

- ***/components*** – společné React komponenty daného modulu,
- ***/scenes*** – React komponenty, které představují jednotlivé obrazovky aplikace (kontejnery propojené s Reduxem),
- ***/utils*** – adresář pro utilizační třídy daného modulu,
- ***/reducers*** x ***reducer.js*** – adresář nebo soubor s definicí jednotlivých reduktorů, akcí a stavu daného modulu (dle upraveného vzoru Ducks [74]),
- ***constants.js*** – soubor s definovanými konstantami pro daný modul,

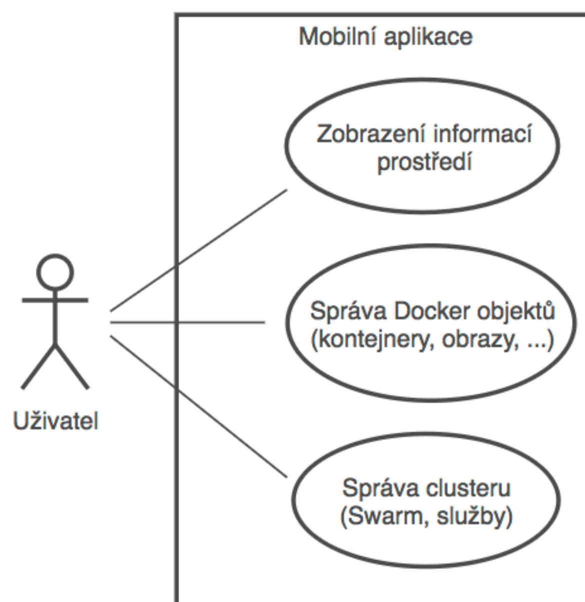
- ***index.js*** – definice veřejného rozhraní pro zjednodušení importu specifikovaných souborů.

Aplikace je rozdělena do dvou modulů – ***auth*** a ***main*** modul. Auth modul obsahuje komponentu SignIn, která zodpovídá za přihlášení uživatele do aplikace. Komponenta úzce souvisí s třídou AuthUtils.js, která se stará o manipulaci s uživatelem, např. zpracování přijatého tokenu při přihlášení, odhlášení uživatele nebo ověření, zda je v aplikaci přihlášen nějaký uživatel. Main modul je jádrem aplikace. Každá z jeho několika komponent obsahuje UI prvky pro komunikaci skrze middleware s definovanými Docker nody.

Požadavky na mobilní aplikaci

Pro mobilní aplikaci jsou stanoveny následující požadavky:

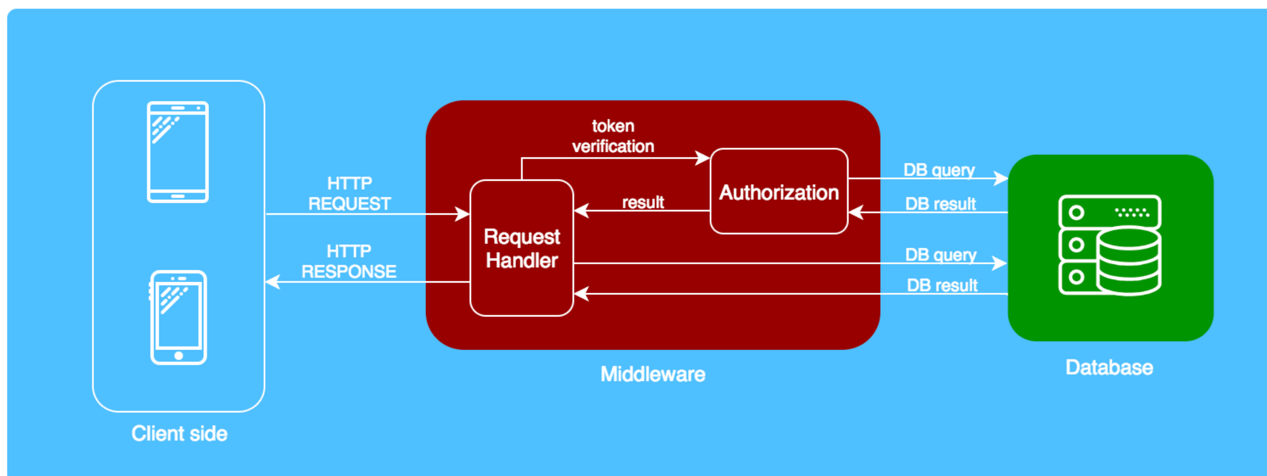
- funkční
    - zobrazení informací o jednotlivých prostředích,
    - správa Docker objektů,
    - správa clusteru,
- nefunkční
    - jednoduché a uživatelsky přívětivé UI specifické pro danou platformu.



Obrázek 16 Případy použití mobilní aplikace. Zdroj: [autor]

6.2 Middleware

MiddleWhale [75] je navržen jako middleware mezi klientskou a serverovou částí a obstarává ověření jednotlivých uživatelů a jejich požadavků posílaných na konkrétní REST API Docker clusteru. Celý middleware je postaven na platformě Node.js a pro správu uživatelů využívá NoSQL databázi MongoDB. Na Obrázek 17 je zobrazen ověřovací proces uživatele komunikujícího s middlewarem.



Obrázek 17 Proces ověření uživatele. Zdroj: [autor]

Pro zabezpečení systému bylo implementováno ověřování pomocí tokenů. Toto ověřování se nazývá Bearer autentizace a je to HTTP autentizační schéma, které zahrnuje bezpečnostní tokeny nazývané bearer token (bearer = nositel/posel). Tento druh ověřování může být v překladu rozuměn jako „dej přístup nositeli tohoto tokenu“. Bearer token je zakódovaný řetězec, který je obvykle vygenerován serverem jako odpověď na základě úspěšného přihlášení uživatele. Klient musí tento token posílat v autorizační hlavičce požadavku v případě, že chce přistupovat k zabezpečeným zdrojům:

```
Authorization: 'Bearer <token>'
```

Toto autentizační schéma by mělo být, podobně jako schéma typu Basic, používáno pouze přes zabezpečený HTTPS (SSL) protokol.

Ověřování uživatele na middlewaru probíhá ve dvou případech – přihlášení uživatele a odeslání požadavku uživatele na Docker Engine API.

6.2.1 Přihlášení uživatele

Proces ověření uživatele při přihlašování na middleware probíhá na koncovém bodě **„/user/signin**“ pomocí POST požadavku, jehož součástí jsou data v JSON formátu:

```
{
  "username": "duckling",
  "password": "1am.potato"
}
```

Server pomocí knihovny bcrypt [76] provede stejným způsobem, jako při registraci nového uživatele, zašifrování zadaného hesla hashovací funkcí a odešle dotaz na databázi, kde je ověřeno, zda oba řetězce souhlasí. V případě shody jsou uživateli vygenerovány dva tokeny – krátkodobý a dlouhodobý. Krátkodobý token je představován JWT tokenem (formát tokenu „*JWTpart1.JWTpart2.JWTpart3*"). Pro každého uživatele je unikátní podle emailu a tajného klíče a doba jeho platnosti je nastavena na 5 minut. Dlouhodobý token je unikátní a skládá se z aktuálního data, času, uživatelského jména a zašifrovaného hesla. Celý tento řetězec je dále zašifrován stejnou metodou jako uživatelské heslo. Oba tyto tokeny jsou odeslány jako jeden ve formátu:

„*JWTpart1.JWTpart2.**uniqueToken**.JWTpart3*".

6.2.2 Odeslání požadavku na API

Druhým případem je odesílání HTTP požadavku na koncový bod s Docker nodem „**/api/:env/***" (:env – libovolné Docker prostředí, nakonfigurované na middlewaru). Každý požadavek musí v hlavičce obsahovat Bearer token, který při přihlášení uživatel obdržel. Z důvodu většího zabezpečení je nutné, aby klientská strana znala formáty přijímaného a odesílaného tokenu. Na rozdíl od tokenu, který uživatel obdrží, je nutné prohodit 2. a 3. část tokenu:

„*JWTpart1.**uniqueToken**.JWTpart2.JWTpart3*".

6.2.3 Konfigurace Docker nodů

Konfigurace a správa jednotlivých Docker nodů probíhá pomocí textového konfiguračního souboru, který stačí jednoduše vložit do složky „**endpoints**" na middlewaru. Při spouštění middlewaru se provádí inicializace všech koncových bodů podle těchto konfiguračních souborů. Ukázka konfigurace jednoho ze čtyř koncových bodů systému je ve formátu:

```
env1;Docker L-Node 1;http://imitgw.uhk.cz;59830;docker-l-vm01;Linux
```

Struktura konfigurace je následující:

- unikátní název – název, pro který bude vytvořena cesta na API middlewaru („/api/env1/“),
- název koncového bodu,
- URL koncového bodu,
- port Docker Engine API daného bodu,
- server hostname,
- operační systém.

6.3 Docker cluster

Konfigurace celého Docker clusteru na IMIT zobrazuje Tabulka 1:

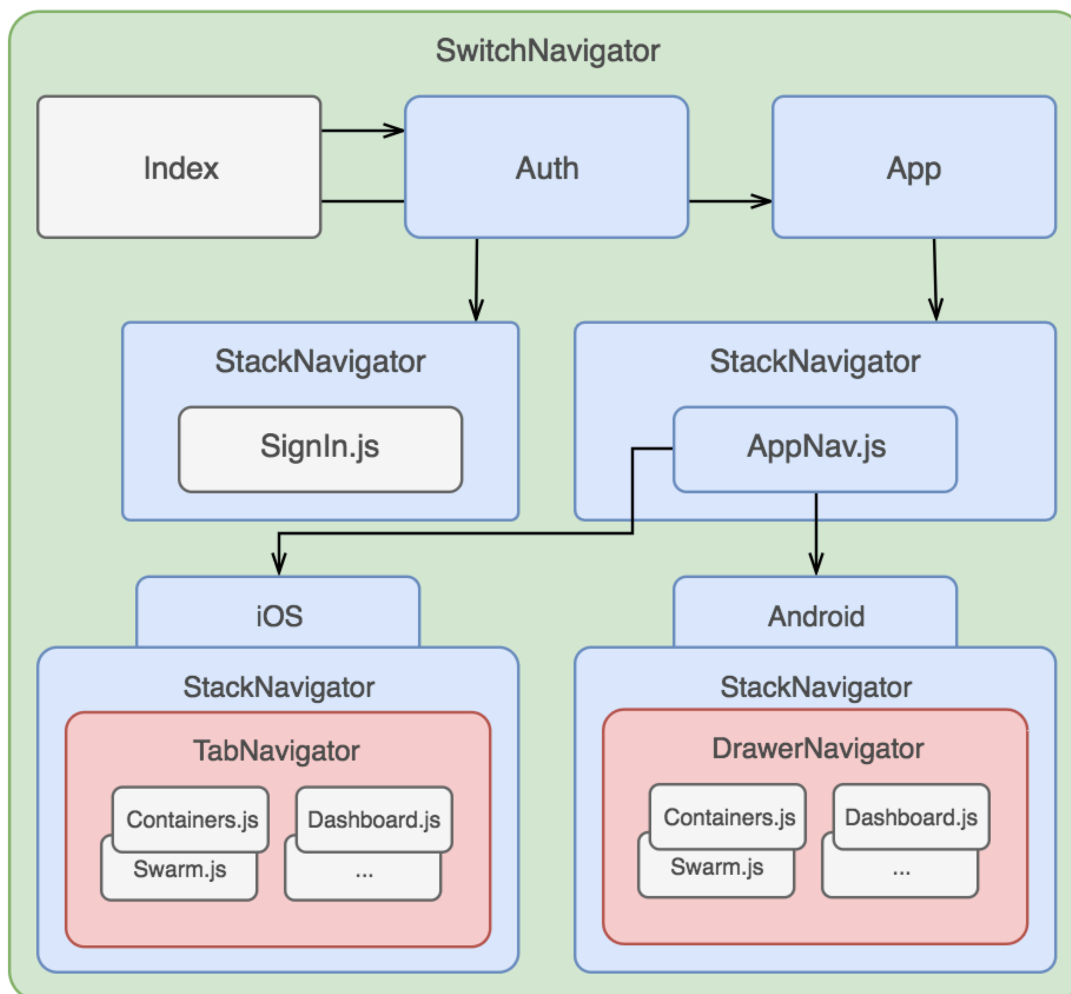
Hostname	Oper. systém	vCPU	RAM	API port
docker-l-vm01	Ubuntu 16.04.3 LTS	4x	4 GB	59830
docker-l-vm02	Ubuntu 16.04.3 LTS	4x	4 GB	59831
docker-ws-vm01	Windows Server 2016 64bit	4x	4 GB	59832
docker-ws-vm02	Windows Server 2016 64bit	4x	4 GB	59833

Tabulka 1 Konfigurace jednotlivých Docker serverů v clusteru. Zdroj: [autor]

Komunikace s jednotlivými Docker instancemi je zprostředkována skrze bránu na adrese http://imitgw.uhk.cz. Pro všechny Docker Engine API byl zvolen výchozí port 8593, který je skrze univerzitní firewall přesměrován na výše zmíněné API porty.

7 Realizace aplikace

Klíčovým prvkem každé mobilní aplikace je navigace skrze celou aplikací. Pro multiplatformní aplikaci je možné použít jednu navigační strukturu pro všechny cílené platformy nebo specifikovat dle jednotlivých platforem. Pro navigační strukturu této aplikace byla použita knihovna React Navigation [77], která je doporučeným řešením pro obsluhu přechod mezi jednotlivými obrazovky. Cílem této knihovny je využití prvků pro návrh navigační struktury běžné pro uživatelské prostředí nativních aplikací operačních systémů Android a iOS. Celá struktura zobrazena na Obrázek 18 se skládá z několika prvků, které jsou zapouzdřeny do sebe. Toto zapouzdření je důležité pro odstínění viditelnosti cest ostatními prvky a pro doplnění chybějících funkcionalit jednotlivých navigačních prvků.



Obrázek 18 Navigační struktura aplikace. Zdroj: [autor]

Nejvyšším prvkem je SwitchNavigator, který má funkci přepínače mezi přihlášenou (App) a nepřihlášenou částí aplikace (Index, Auth). Inicializace SwitchNavigatoru spustí Index.js jako výchozí obrazovku. Funkcí této obrazovky je přesměrovat uživatele do správné části aplikace podle toho, jestli je přihlášený do aplikace nebo ne. Tato část kódu zjišťuje ihned po vytvoření aktivity, jestli v persistentním úložišti existuje záznam o uživateli a na základě výsledku přesměrovává do vybrané části.

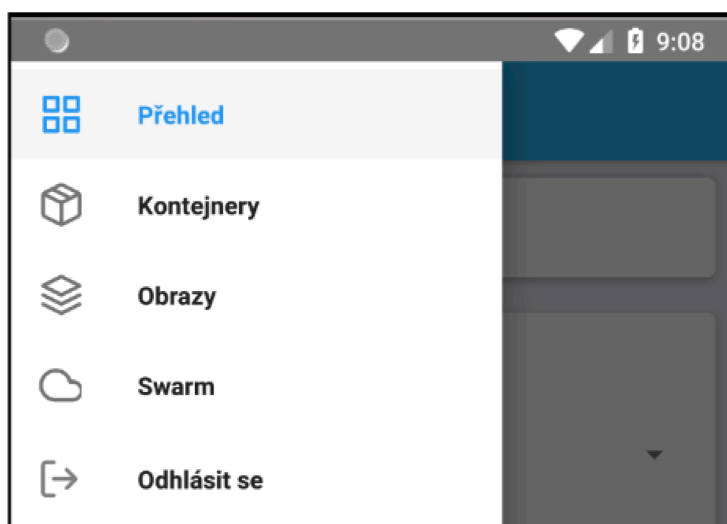
```
async auth() {
        const { navigation } = this.props;
        await getStorageItem(ASYNC_ST_USER_KEY)
            .then((user) => {
                if (user !== null) {
                    navigation.navigate('App');
                } else {
                    navigation.navigate('Auth');
                }
            });
    }
```

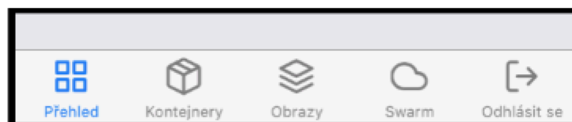
Část AppNav v sobě zapouzdřuje navigační strukturu k hlavním obrazovkám navigace a detailním obrazovkám jako například zobrazení detailů kontejneru, vytvoření nového kontejneru apod. Kód je společný pro obě platformy a je zde využívána funkce React Native pro rozlišení přípon souborů podle specifické platformy. Tato funkce je vhodná pro případy, kdy je kód pro danou platformu moc komplexní. První řádek zobrazeného kódu automaticky vybere patřiční soubor MainScreens.android.js nebo MainScreens.ios.js a nahraje ho.

```
import MainScreens from './MainScreens';
import DetailNav from './DetailNav';

export default createStackNavigator({
    MainScreens: {
        screen: ({ screenProps, navigation }) => <MainScreens screenProps={{
screenProps, navigation }} />,
    },
    DetailNav: {
        screen: ({ screenProps, navigation }) => <DetailNav screenProps={{
screenProps, navigation }} />,
    }
}, {
    headerMode: 'none',
});
```

Rozdělení navigační struktury zvlášť pro OS Android a iOS je z důvodů rozdílných UI prvků. Android aplikace používají především navigační prvek Navigation Drawer, který představuje boční menu se seznamem položek zobrazujících jednotlivé obrazovky hlavní navigace. Oproti tomu iOS aplikace využívají tlačítkovou navigační lištu, která je ve většině případů umístěna ve spodní části obrazovky.





Obrázek 19 Navigace platforem Android a iOS. Zdroj: [autor]

Vzhledem k těmto rozdílům a cíli udržet aplikaci uživatelsky přívětivou a co nejvíce nativní, bylo nutné část MainScreens rozdělit zvlášť pro každou platformu. Část pro platformu Android je řešena pomocí DrawerNavigatoru a každá z obrazovek je obalena do StackNavigatoru. StackNavigator představuje zásobník s jednotlivými obrazovkami a umožňuje zobrazit navigační lištu, které je možné nastavit název a dvě libovolné akce.

```
Dashboard: {
        screen: createStackNavigator({
            Dashboard: {
                screen: Dashboard,
                navigationOptions: ({ navigation }) => createAndNavOptions(
                    s.MENU_LABEL_DASHBOARD,
                    setNavAction(navigation, false),
                ),
            },
        }),
```

```
                navigationOptions: {
                    drawerLabel: s.MENU_LABEL_DASHBOARD,
                    drawerIcon: ({ tintColor }) => <Feather name={i.DASHBOARD}
        size={25} color={tintColor} />,
                },
            }, ...
```

iOS část využívá na rozdíl od Androidu BottomTabNavigator pomocí kterého je možné přizpůsobit navigace ve stejném stylu jako běžné nativní aplikace pro iOS. Stejně jako v Android části jsou i zde obrazovky obalené StackNavigatorem.

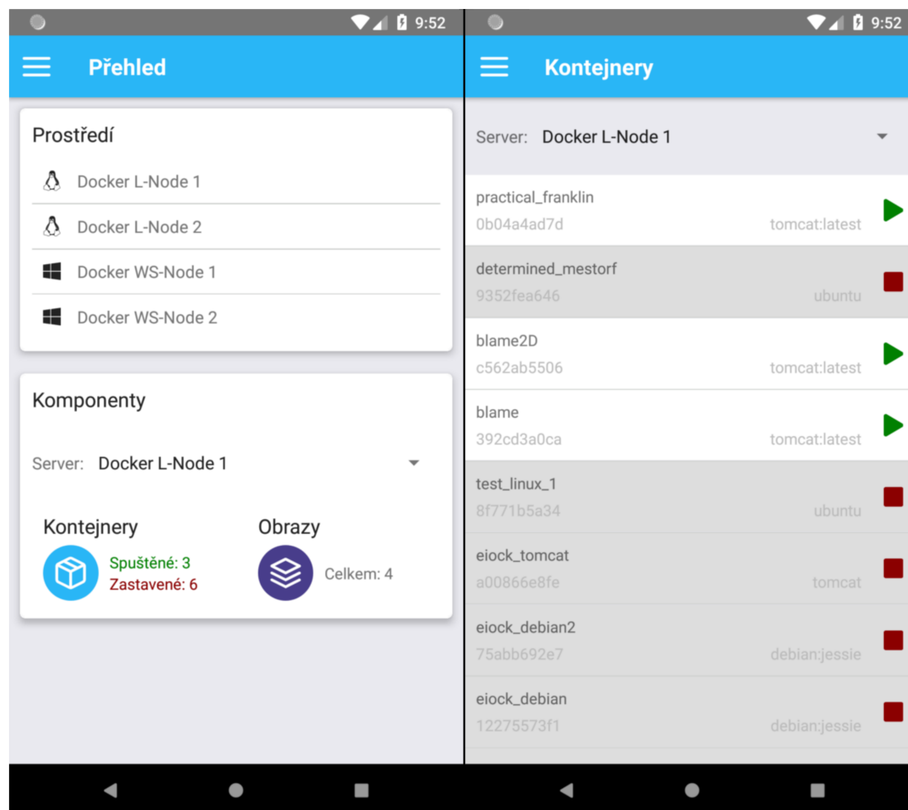
```
        Dashboard: {
                screen: createStackNavigator({
                    Dashboard: {
                        screen: Dashboard,
                        navigationOptions: createIOSNavOptions(
                            s.MENU_LABEL_DASHBOARD,
                            null,
                            null,
                        ),
                    },
                }),
                navigationOptions: {
                    tabBarLabel: s.MENU_LABEL_DASHBOARD,
                },
            }, ...
```

Viditelným rozdílem u obou hlavních navigací jsou definované akce pro navigační lištu. Zatímco název je společnou částí obou funkcí, pro iOS je možnost definovat obě funkce navigační lišty, Android část má definovanou pouze jednu (levou) funkci. Důvodem je využití pouze levé části navigační lišty platformou Android pro zobrazení ikony s akcí otevření Navigation Draweru nebo pro navigaci o obrazovku zpět.

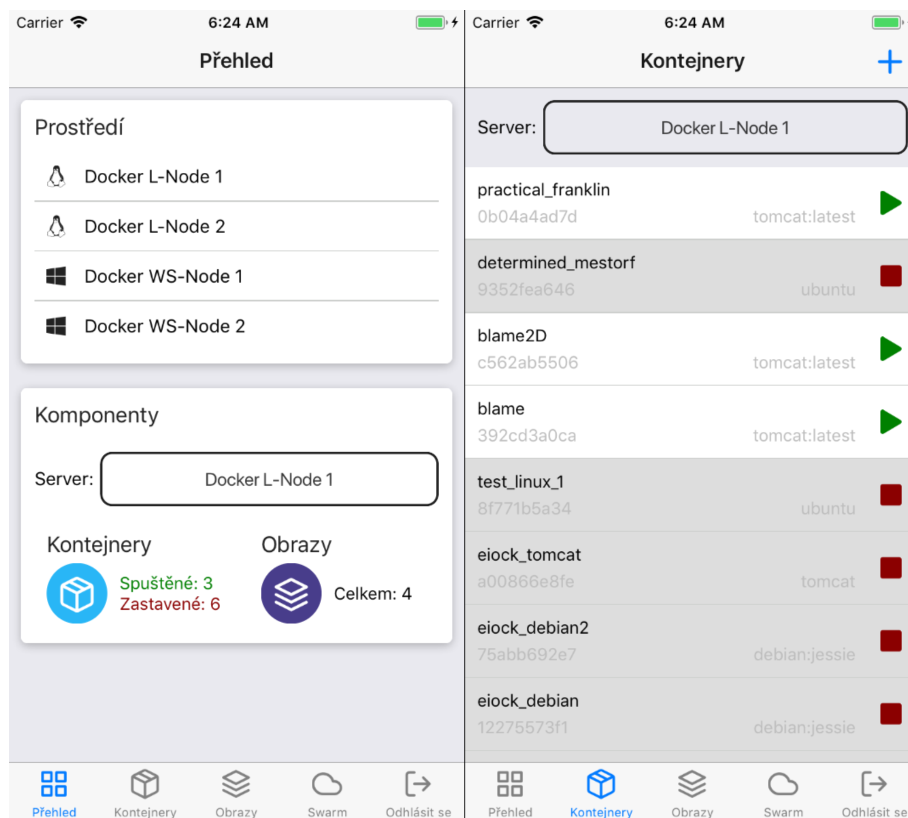
```
        export const createIOSNavOptions = (title, leftAction, rightAction) => ({
            headerTitle: title,
            headerLeft: leftAction,
            headerRight: rightAction,
        });

        export const createAndNavOptions = (title, leftAction) => ({
            headerTitle: title,
            headerLeft: leftAction,
            headerStyle: { backgroundColor: c.lightBlue },
            headerTintColor: c.colorWhite,
        });
```

Výsledkem této složené navigační struktury je nativně působící navigace pro obě cílové platformy zobrazená na Obrázek 20 Obrázek 21.



Obrázek 20 Aplikace pro platformu Android. zdroj: [autor]



Obrázek 21 Aplikace pro platformu iOS. zdroj: [autor]

8 Závěr

Cílem této diplomové práce bylo vytvoření multiplatformní mobilní aplikace pro správu produkčního Docker prostředí Institutu moderních informačních technologií pro operační systémy Android a iOS.

Součástí teoretické části je seznámení s virtualizačními a kontejnerizačními technologiemi a také seznámení s problematikou vývoje multiplatformních mobilních aplikací.

Pro komunikaci se serverovým prostředím byl vytvořen middleware, který přijímá požadavky klienta, předává je serverovému prostředí a vrací klientovi odpovědi serveru.

Pro platformy Android a iOS byla implementována mobilní aplikace pro správu Docker prostředí pomocí technologie React Native.

Mezi plánované budoucí rozšíření této aplikace je přidání monitorování logů jednotlivých kontejnerů v reálném čase, připojení do kontejneru v interaktivním módu a přidání možnosti vytváření většího počtu kontejnerů najednou.

9 Seznam použité literatury

[1] **RIGHTSCALE.** State of the Cloud 2018. *RightScale.* [Online] 2018. [Citace: 11. 03 2018.] https://www.rightscale.com/lp/2018-state-of-the-cloud-report.

[2] **THE LINUX FOUNDATION.** A Brief Look at the Roots of Linux Containers. *The Linux Foundation.* [Online] 2018. [Citace: 12. 03 2018.] https://www.linuxfoundation.org/blog/a-brief-look-at-the-roots-of-linux-containers/.

[3] **ATLASSIAN.** What is DevOps? *Software Development and Collaboration Tools.* [Online] Atlassian, 2018. [Citace: 11. 03 2018.] https://www.atlassian.com/devops.

[4] **RIGHTSCALE.** RightScale Cloud Management. *RightScale.* [Online] 2018. [Citace: 11. 03 2018.] https://www.rightscale.com.

[5] —. State of the Cloud 2015. *RightScale.* [Online] 2015. [Citace: 11. 03 2018.] https://www.rightscale.com/lp/2015-state-of-the-cloud-report.

[6] —. State of the Cloud 2016. *RightScale.* [Online] 2016. [Citace: 11. 03 2018.] https://www.rightscale.com/lp/2016-state-of-the-cloud-report.

[7] —. State of the Cloud 2017. *RightScale.* [Online] 2017. [Citace: 11. 03 2018.] https://www.rightscale.com/lp/2017-state-of-the-cloud-report.

[8] **ALAPATI, Sam R.** *Modern Linux Administration: How to Become a Cutting-Edge Linux Administrator.* Sebastopol, Kalifornie, USA : O'Reilly Media, 2018. ISBN 1491935952.

[9] **OSNAT, Rani.** A Brief History of Containers: From the 1970s to 2017. *Aqua Blog.* [Online] [Citace: 01. 08 2018.] https://blog.aquasec.com/a-brief-history-of-containers-from-1970s-chroot-to-docker-2016.

[10] **DOCKER INC.** What is Docker. *Docker.* [Online] [Citace: 01. 08 2018.] https://www.docker.com/what-docker.

[11] —. About images, containers, and storage drivers. *Docker Documentation.* [Online] [Citace: 30. 07 2018.] https://docs.docker.com/engine/userguide/storagedriver/imagesandcontainers/.

[12] **KASIREDDY, Preethi.** A Beginner-Friendly Introduction to Containers, VMs and Docker. *freeCodeCamp.* [Online] [Citace: 28. 07 2018.] https://medium.freecodecamp.org/a-beginner-friendly-introduction-to-containers-vms-and-docker-79a9e3e119b.

[13] **NICKOLOFF, Jeff.** *Docker in action.* Shelter Island, NY : Manning Publications, Co., 2016. ISBN 9781633430235.

[14] **DOCKER INC.** Docker Engine. *Docker - Overview.* [Online] [Citace: 28. 07 2018.] https://docs.docker.com/engine/docker-overview/#docker-engine.

[15] —. Best practices for writing Dockerfiles. *Docker Documentation.* [Online] [Citace: 30. 07 2018.] https://docs.docker.com/develop/develop-images/dockerfile_best-practices/.

[16] —. What is container. *Docker.* [Online] [Citace: 30. 07 2018.] https://www.docker.com/what-container.

[17] —. How services work. *Docker Documentation.* [Online] [Citace: 30. 07 2018.] https://docs.docker.com/engine/swarm/how-swarm-mode-works/services/.

[18] —. Swarm mode overview. *Docker Documentation.* [Online] [Citace: 30. 07 2018.] https://docs.docker.com/engine/swarm/.

[19] —. Swarm mode key concepts . *Docker Documentation.* [Online] [Citace: 30. 07 2018.] https://docs.docker.com/engine/swarm/key-concepts/.

[20] **STATCOUNTER.** Mobile Operating System Market Share Worldwide. *StatCounter Global Stats - Browser, OS, Search Engine including Mobile Usage Share.* [Online] [Citace: 26. 06 2018.] http://gs.statcounter.com/os-market-share/mobile/worldwide/#yearly-2009-2018.

[21] **GOOGLE.** Android 8.1.0 Release 1. *Google Git.* [Online] [Citace: 26. 06 2018.] https://android.googlesource.com/platform/build/+/android-8.1.0_r1.

[22] —. Android Platform | Android Developers. *Android Developers.* [Online] [Citace: 30. 06 2018.] https://developer.android.com/about/.

[23] —. Platform Architecture. *Android Developers.* [Online] [Citace: 26. 06 2018.] https://developer.android.com/guide/platform/.

[24] —. Kernel | Android Open Source Project. *Android Open Source Project.* [Online] [Citace: 26. 06 2018.] https://source.android.com/devices/architecture/kernel/.

[25] —. Hardware Abstraction Layer | Android Open Source Project. *Android Open Source Project.* [Online] [Citace: 26. 06 2018.] https://source.android.com/devices/#Hardware%20Abstraction%20Layer.

[26] —. ART and Dalvik | Android Open Source Project. *Android Open Source Project.* [Online] [Citace: 26. 06 2018.] https://source.android.com/devices/tech/dalvik/.

[27] **MACINKA, Michal.** *Implementace hry pro OS Android.* Hradec Králové : Univerzita Hradec Králové, Fakulta informatiky a managementu. Vedoucí práce Doc. Mgr. Tomáš Kozel, Ph.D., 2016.

[28] **SHAFIROV, Maxim.** Kotlin on Android. Now official. *Kotlin Blog.* [Online] [Citace: 26. 06 2018.] https://blog.jetbrains.com/kotlin/2017/05/kotlin-on-android-now-official/.

[29] **BHAVE, Mahesh P. a SUNIL, Patekar A.** *Programming with Java.* Delhi : Pearson Education, 2009. ISBN 9788131761571.

[30] **TIOBE.** TIOBE Index for June 2018. *TIOBE Index.* [Online] [Citace: 26. 06 2018.] https://www.tiobe.com/tiobe-index/.

[31] **JETBRAINS S.R.O.** FAQ - Kotlin Programming Language. *Reference - Kotlin Programming Language.* [Online] [Citace: 26. 06 2018.] https://kotlinlang.org/docs/reference/faq.html.

[32] **LEIVA, Antonio.** *Kotlin for Android Developers: Learn Kotlin the easy way while developing an Android App.* Madrid : CreateSpace Independent Publishing Platform, 2016. ISBN 1530075610.

[33] **COSTELLO, Sam.** The History of iOS, from Version 1.0 to 11.0. *Lifewire.* [Online] [Citace: 29. 06 2018.] https://www.lifewire.com/ios-versions-4147730.

[34] **DANIEL, Steven F.** *Xcode 4 iOS development: beginner's guide : use the powerful Xcode 4 suite of tools to build applications for the iPhone and iPad from scratch.* Birmingham, U.K. : Packt Pub., 2011. ISBN 1849691304.

[35] **SHERIFF, Mark.** iOS Architecture. *CS 4720 - Mobile Application Development .* [Online] [Citace: 29. 06 2018.] https://cs4720.cs.virginia.edu/slides/CS4720-MAD-iOSArchitecture.pdf.

[36] **HALVORSEN, Ole Henry a DOUGLAS, Clarke.** *OS X and iOS Kernel Programming.* Berkeley, CA : Apress, 2011. ISBN 9781430235378.

[37] **APPLE INC.** Automatic Reference Counting. *The Swift Programming Language.* [Online] [Citace: 29. 06 2018.] https://docs.swift.org/swift-book/LanguageGuide/AutomaticReferenceCounting.html.

[38] —. UIKit. *UIKit | Apple Developer Documentation.* [Online] [Citace: 29. 06 2018.] https://developer.apple.com/documentation/uikit.

[39] —. Apple Developer Documentation. *Apple Developer.* [Online] 2018. [Citace: 11. 03 2018.] https://developer.apple.com/documentation/.

[40] **KOCHAN, Stephen G.** *Objective-C 2.0: výukový kurz programování pro Mac.* Brno : Computer Press, 2010. ISBN 978-80-251-2654-7.

[41] **APPLE INC.** About Swift. *Swift.org.* [Online] [Citace: 29. 06 2018.] https://swift.org/about/.

[42] —. Swift - Apple Developer. *Apple Developer.* [Online] 2018. [Citace: 11. 03 2018.] https://developer.apple.com/swift/.

[43] —. Swift - Apple. *Apple.* [Online] [Citace: 29. 06 2018.] https://www.apple.com/lae/swift/.

[44] **GITHUB INC.** Topic: Swift. *GitHub.* [Online] [Citace: 29. 06 2018.] https://github.com/topics/swift?l=swift.

[45] **CORRAL, Luis, SILLITTI, Alberto a SUCCI, Giancarlo.** Mobile multiplatform development: An experiment for performance analysis. *Procedia Computer Science.* 2012, 10.

[46] **FACEBOOK.** Getting Started · React Native. *React Native · A framework for building native apps using React.* [Online] 2018. [Citace: 11. 03 2018.] https://facebook.github.io/react-native/docs/getting-started.html.

[47] **EISENMAN, Bonnie.** *Learning React Native: building mobile applications with JavaScript.* Beijing : O'Reilly, 2015. ISBN 1491929006.

[48] **EVKOSKI, Blagoja.** React Native: What it is and how it works. *Medium.* [Online] [Citace: 20. 07 2018.] https://wetalkit.xyz/react-native-what-it-is-and-how-it-works-e2182d008f5e.

[49] **FACEBOOK INC.** React Native - Components and APIs. *Facebook - React Native.* [Online] [Citace: 20. 07 2018.] https://facebook.github.io/react-native/docs/components-and-apis.

[50] **GOOGLE.** WebView. *Android Developers.* [Online] [Citace: 20. 07 2018.] https://developer.android.com/reference/android/webkit/WebView.

[51] —. Flutter Documentation. *Flutter - Beautiful native apps in record time.* [Online] 2018. [Citace: 11. 03 2018.] https://flutter.io/docs/.

[52] —. Overview: The Dart Language. *The Dart Langugage.* [Online] [Citace: 22. 07 2018.] https://www.dartlang.org/guides/language.

[53] —. Flutter - FAQ. *Flutter.* [Online] [Citace: 22. 07 2018.] https://flutter.io/faq.

[54] **MICROSOFT.** Xamarin. *Visual Studio.* [Online] [Citace: 26. 07 2018.] https://visualstudio.microsoft.com/xamarin/.

[55] **Mono Project.** Mono. *Cross platform, open source .NET framework .* [Online] [Citace: 26. 07 2018.] http://www.mono-project.com/.

[56] **ICAZA, Miguel.** Announcing Xamarin. *Miguel de Icaza's Blog.* [Online] [Citace: 26. 07 2018.] http://tirania.org/blog/archive/2011/May-16.html.

[57] **GUTHRIE, Scott.** Microsoft to acquire Xamarin and empower more developers to build apps on any device. *Official Microsoft Blog.* [Online] [Citace: 26. 07 2018.] https://blogs.microsoft.com/blog/2016/02/24/microsoft-to-acquire-xamarin-and-empower-more-developers-to-build-apps-on-any-device/.

[58] **ALTEXSOFT.** The Good and The Bad of Xamarin Mobile Development. *Altexsoft.* [Online] [Citace: 26. 07 2018.] https://www.altexsoft.com/blog/mobile/pros-and-cons-of-xamarin-vs-native/.

[59] **MOZILLA FOUNDATION.** Progressive web apps. *MDN web docs.* [Online] [Citace: 23. 07 2018.] https://developer.mozilla.org/en-US/Apps/Progressive.

[60] **GOOGLE.** Lighthouse. *Web | Google Developers.* [Online] [Citace: 23. 07 2018.] https://developers.google.com/web/tools/lighthouse/.

[61] **MOZILLA FOUNDATION.** Introduction to progressive web apps. *MDN web docs.* [Online] [Citace: 23. 07 2018.] https://developer.mozilla.org/en-US/Apps/Progressive/Introduction.

[62] **BRISTOWE, John.** What is a Hybrid Mobile App? *Telerik - Developer Network.* [Online] [Citace: 23. 07 2018.] https://developer.telerik.com/featured/what-is-a-hybrid-mobile-app/.

[63] **APACHE CORDOVA.** Overview. *Apache Cordova Documentation.* [Online] [Citace: 25. 07 2018.] https://cordova.apache.org/docs/en/latest/guide/overview/index.html.

[64] —. Cordova Plugins. *Apache Cordova.* [Online] [Citace: 25. 07 2018.] https://cordova.apache.org/plugins/.

[65] **IONIC.** Ionic Framework Documentation. *Ionic.* [Online] [Citace: 25. 07 2018.] https://ionicframework.com/docs/.

[66] —. The Ionic Book. *Ionic.* [Online] [Citace: 25. 07 2018.] https://ionicframework.com/docs/v1/guide/.

[67] —. Ionic Native. *Ionic Documentation.* [Online] [Citace: 25. 07 2018.] https://ionicframework.com/docs/native/.

[68] **MAHARANA, Netrananda.** Cross Platform Mobile Apps and Its Pros and Cons. *Medium.* [Online] https://medium.com/andolasoft/cross-platform-mobile-apps-and-its-pros-and-cons-9c257ec64e94.

[69] **JWMGLOBAL.** Pros & Cons of Cross-Platform Mobile App Development. *JWM GLOBAL.* [Online] [Citace: 26. 07 2018.] https://jwm.global/pros-cons-cross-platform-mobile-app-development/.

[70] **DOCKER INC.** UCP System requirements. *Docker Documentation.* [Online] [Citace: 18. 07 2018.] https://docs.docker.com/v17.09/datacenter/ucp/2.2/guides/admin/install/system-requirements/.

[71] **FACEBOOK INC.** Flux - In Depth Overview. *Flux APPLICATION ARCHITECTURE FOR BUILDING USER INTERFACES.* [Online] [Citace: 22. 07 2018.] https://facebook.github.io/flux/docs/in-depth-overview.html#content.

[72] **GELMAN, Ilya a DINKEVICH, Boris.** *The Complete Redux Book.* Seattle : Amazon Digital Services LLC, 2017. ISBN 978-965-92642-0-9.

[73] **TAHIR, Nish.** Lessons Learned Implementing Redux on Android. *Hacker Noon.* [Online] [Citace: 22. 07 2018.] https://hackernoon.com/lessons-learned-implementing-redux-on-android-cba1bed40c41.

[74] **RASMUSSEN, Erik.** Ducks: Redux Reducer Bundles. *GitHub.* [Online] [Citace: 21. 07 2018.] https://github.com/erikras/ducks-modular-redux.

[75] **MACINKA, Michal.** MiddleWhale - Docker API Middleware. *Heroku.* [Online] [Citace: 18. 07 2018.] https://middlewhale.herokuapp.com/.

[76] **CAMPBELL, Nick.** bcrypt for NodeJs. *GitHub.* [Online] 18. 07 2018. https://github.com/kelektiv/node.bcrypt.js.

[77] **REACT NAVIGATION.** React Navigation - Getting started. *React Navigation.* [Online] 14. 08 2018. https://reactnavigation.org/docs/en/getting-started.html.

10 Seznam obrázků

[1] Použití kontejnerizační technologií. Zdroj: [1] ... 1
[2] Využití nástroje Docker od roku 2015 do současnosti. Zdroj: [autor] ... 2
[3] Struktura virtuálního stroje a kontejneru Zdroj: [12] ... 7
[4] Architektura Docker Engine. Zdroj: [14] ... 9
[5] Podíl mobilních operačních systémů od roku 2009 do 2018. Zdroj: [20] ... 13
[6] Hlavní komponenty platformy Android. Zdroj: [23] ... 14
[7] Graf oblíbenosti programovacích jazyků v 06/2018. Zdroj: [autor] [30] ... 17
[8] Vrstvy iOS architektury. Zdroj: [35] ... 18
[9] Komunikace mezi vlákny. Zdroj: [48] ... 23
[10] Architektura Flutter SDK. Zdroj: [53] ... 25
[11] Sdílení společné části kódu se všemi cílenými platformami. Zdroj: [58] ... 26
[12] Architektura Cordova aplikace. Zdroj [63] ... 28
[13] Architektura komunikace mezi jednotlivými vrstvami na oddělení IMIT. Zdroj: [autor] ... 32
[14] Architektura knihovny Redux. Zdroj: [73] ... 33
[15] Architektura mobilní aplikace. Zdroj: [autor] ... 34
[16] Případy použití mobilní aplikace. Zdroj: [autor] ... 35
[17] Proces ověření uživatele. Zdroj: [autor] ... 36
[18] Navigační struktura aplikace. Zdroj: [autor] ... 39
[19] Navigace platforem Android a iOS. Zdroj: [autor] ... 41
[20] Aplikace pro platformu Android. zdroj: [autor] ... 43
[21] Aplikace pro platformu iOS. zdroj: [autor] ... 43

11 Seznam tabulek

[1] Konfigurace jednotlivých Docker serverů v clusteru. Zdroj: [autor] 38

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2016/2017

Studijní program: Aplikovaná informatika
Forma: Prezenční
Obor/komb.: Aplikovaná informatika (ai2-p)

Podklad pro zadání DIPLOMOVÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Bc. Macinka Michal	Lhotka 198, Česká Třebová - Lhotka	I1600888

TÉMA ČESKY:

Mobilní management Dockeru

TÉMA ANGLICKY:

Docker Mobile Management

VEDOUCÍ PRÁCE:

doc. Mgr. Tomáš Kozel, Ph.D. - KIKM

ZÁSADY PRO VYPRACOVÁNÍ:

Cíl práce:
Popsat problematiku vzdáleného ovládání platformy Docker, prozkoumat potenciál využití její správy prostřednictvím mobilního zařízení a navrhnout a implementovat multiplatformní mobilní aplikaci pro vzdálenou správu Dockeru pro OS Android a iOS.

Osnova:
1. Úvod
2. Docker
3. Multiplatformní vývoj
4. Návrh aplikace
5. Realizace aplikace
6. Závěr

SEZNAM DOPORUČENÉ LITERATURY:

NICKOLOFF, Jeff. Docker in action. Shelter Island, NY: Manning Publications, Co., 2016. ISBN 978-1633430235.
EISENMAN, Bonnie. Learning React Native: building mobile applications with JavaScript. Beijing: O'Reilly, 2015. ISBN 1491929006.

Podpis studenta: .... **Datum:** 18.9.17

Podpis vedoucího práce: .... **Datum:** 18.9.17

(c) IS/STAG , Portál - Podklad kvalifikační práce , I1600888 , 18.09.2017 12:26