Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Infrastruktura pro perzonalizaci a vzdálenou správu nositelných zařízení

Plzeň, 2024 Bc. David Markov

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2023/2024

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. David MARKOV**
Osobní číslo:	**A21N0059P**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Softwarové inženýrství**
Téma práce:	**Infrastruktura pro perzonalizaci a vzdálenou správu nositelných zařízení**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se se systémem SmartCGMS a možnostmi nasazení jeho komponent na nositelných zařízení.
2. Navrhněte vhodnou architekturu pro realizaci softwarové platformy pro snadný onboarding nositelných zařízení, jejich vzdálenou správu a sběr dat.
3. Platformu navrhněte tak, aby obsahovala uživatelský webový portál a backend pro obsluhu nositelných a mobilních zařízení. Navrhněte uživatelsky snadný způsob perzonalizace nositelných a mobilních zařízení. Všechny relevatní případy užití zdokumentujte podrobnými procesními diagramy.
4. Navrhněte integraci vhodné autentikační a autorizační služby, která umožní samoobslužnou registraci uživatelů do portálu. U autorizace rozlišujte tyto základní uživatelské role: administrátor, zdravotník a pacient.
5. Navržená řešení implementuje a v závěru zhodnoťte.

Rozsah diplomové práce: **doporuč. 50 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

dodá vedoucí diplomové práce

Vedoucí diplomové práce: **Ing. Maxmilián Otta, Ph.D.**
Katedra informatiky a výpočetní techniky

Datum zadání diplomové práce: **8. září 2023**
Termín odevzdání diplomové práce: **16. května 2024**

L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 11. října 2023

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 15. května 2024

Bc. David Markov

Abstract

The goal of this work is to implement a cloud platform for remote management and personalization of wearable devices. The theoretical part introduces the issues of deploying applications to wearable devices and data processing in IoT. Subsequently, the concept of cloud applications and their difference from traditional software is proposed, on the principles of which the developed platform will be bulilt. The analytical part presents the different technologies and services that will be used for the implementation of the platform. The practical part of the thesis deals with the implementation of the web portal, configuration of the supporting infrastructure and deployment of the whole system in the cloud environment. The conclusion of the thesis is dedicated to the evaluation of the developed system.

Abstrakt

Cílem této práce je implementace cloudové platformy pro vzdálenou správu a personalizaci nositelných zařízení. V teoretické části je představena problematika nasazování software na nositelná zařízení a zpracování dat v oblasti IoT. Následně je vysvětlen koncept cloudových aplikací a jejich odlišnost od tradičního software, na jejichž principech bude vyvíjená platforma stavět. V analytické části jsou představeny jednotlivé technologie a služby, které budou pro implementaci platformy využity. Praktická část práce se zabývá implementací webového portálu, konfigurací podpůrné infrastruktury a nasazením celého systému do cloudového prostředí. Závěr práce se následně věnuje zhodnocení vyvinutého systému.

Poděkování

Rád bych poděkoval všem, kteří mi v průběhu psaní práce poskytli pomoc a podporu. Především bych chtěl poděkovat panu Ing. Maxmiliánu Ottovi, PhD. za jeho odborné rady, zkušené vedení v průběhu psaní diplomové práce a za sdílení jeho zkušeností v oblasti webových aplikací a cloudových infrastruktur.

Obsah

1 Úvod

Katedra informatiky a výpočetní techniky Západočeské univerzity v Plzni dlouhodobě vyvíjí systém SmartCGMS zaměřený na zpracování fyziologických signálů. Přirozeným cílem projektu je nasazení systému na nositelná zařízení jako jsou chytré hodinky, jejichž senzorů a dalších součástí by mohl systém využít. Data, která senzory a samotný SmartCGMS produkují, je z dlouhodobého hlediska žádoucí uchovávat pro možnosti dalšího zpracování a provádění analýz. Proces nasazení software na vestavěná zařízení, správa jeho verzí a správa zařízení samotných je dalším významným problémem, který je nutné u takovéhoto projektu adresovat.

Cílem této práce je implementace cloudové platformy pro vzdálenou správu a personalizaci těchto nositelných zařízení. Součástí navrženého systému by měl být i mechanismus pro efektivní sběr dat ze spravovaných zařízení.

V kapitole 2 bude představena problematika Internet of Things (IoT), bude detailněji představen systém SmartCGMS, a bude přiblížen proces jeho nasazení na nositelná zařízení.

Obsahem kapitoly 3 bude představení pojmů *cloud computing*, cloudové služby a nativní cloudové aplikace. V rámci této kapitoly budou představeny služby a metodiky, které jsou v praxi pro jejich vývoj využívány. Kapitola 4 následně naváže představením metodiky DevOps a pojmů Continuous Integration (CI), Continuous Delivery (CDE), Continuous Deployment (CD) a Infrastructure as Code (IaC), které jsou s vývojem nativních cloudových aplikací úzce spjaty.

V kapitole 5 budou specifikovány požadavky na systém a kapitola 6 se bude následně zabývat návrhem vhodných technologií a nástrojů pro implementaci webového portálu, výběrem vhodné autentizační služby, její integrací do systému a způsobem správy vytvořené infrastruktury.

Kapitola 7 představí konfiguraci podpůrných služeb infrastruktury pro běh v lokálním i cloudovém prostředí pomocí na míru vytvořených konfiguračních souborů. V kapitole 8 bude následně představena detailní implementace webového portálu, jeho dekompozice a architektura a všechny jeho části budou připraveny pro nasazení do produkčního prostředí.

Kapitola 9 se bude následně věnovat nasazení systému do cloudového prostředí a způsobem správy jeho infrastruktury.

2 Internet věcí a SmartCGMS

V této kapitole bude vysvětlen pojem internet věcí a jeho využití v dnešním světě. Budou stručně zmapována zařízení, která lze řadit do ekosystému internetu věcí a bude představen systém SmartCGMS, včetně jeho vztahu k nositelným zařízením a internetu věcí. V neposlední řadě budou představeny možnosti nasazení software na vestavěná zařízení.

2.1 Internet věcí

Termín Internet věcí [angl. *Internet of Things (IoT)*, dále jen IoT] byl poprvé použit pro označení systémů, ve kterých mohly být objekty z fyzického světa připojeny k Internetu pomocí senzorů. Dnes IoT popisuje scénáře, ve kterých jsou drobná zařízení, senzory a každodenní předměty schopny provádět výpočty a připojit se k Internetu [61].

K podmnožinám IoT zařízení patří například systémy uvnitř dopravních prostředků, monitorující stav opotřebení a servisu, městské senzory, které snímají dopravní situace nebo monitorují životní prostředí, či zařízení uvnitř budov kontrolující zabezpečení a stav objektu. Specifickou skupinou IoT systémů jsou zařízení v bezprostředním okolí člověka, nebo dokonce uvnitř něj. Jedná se často o snímače vitálních funkcí uvnitř chytrých hodinek či fitness náramků nebo o senzory, které jsou součástí medicínských zařízení, jako například snímače systémů pro kontinuální měření glykémie umístěné v podkoží pacienta [34].

Samostatným tématem je sběr dat z těchto zařízení. Sektor IoT produkuje obrovské množství dat, které je nutné odeslat a zpracovat vhodným způsobem, který bude zároveň vhodně škálovat s narůstajícím počtem těchto zařízení. Tyto metody často staví na principu cloudových systémů, cloud computing a distribuovaných systémů [57]. Cloud, cloud computing a principy cloudových aplikací budou dále představeny a popsány v kapitole 3.

2.2 SmartCGMS

SmartCGMS, vyvíjený na Katedře informatiky a výpočetní techniky Západočeské univerzity v Plzni, přináší inovativní přístup ke zpracování fyziologických signálů s cílem extrahovat relevantní závěry. Jeho návrh klade důraz

na vysokou modularitu a využívá principy High-Level Architecture (HLA) [34]. Klíčovým stavebním kamenem systému je efektivní interakce s inzulinovou pumpou, transformace příchozích fyziologických signálů, extrakce relevantních závěrů a jejich případná prezentace uživateli.

Implementace SmartCGMS je navržena tak, aby minimalizovala nároky na výpočetní, paměťové a energetické zdroje, čímž umožňuje plynulý běh systému na vestavěných zařízeních. Software podporuje běh na širokém spektru operačních systémů, včetně Windows, Linux, MacOS, Androi pro chytré telefony a Raspberry Pi. Konečným cílem produktu je integrace na nositelná zařízení, zejména na chytré hodinky.

V rámci snahy o sjednocení funkcionality na různých nositelných zařízeních vznikla propozice architektury, která je založena na mikrokontejnerech a interpretaci WebAssembly [56]. Tento přístup umožňuje optimalizovaný běh SmartCGMS na široké škále nositelných zařízení. Jedním z klíčových prvků navržené platformy je také systém pro plynulé zprovoznění a správu nositelných zařízení, na který se zaměří další obsah této práce.

2.3 Nasazení na nositelná zařízení

Aby bylo možné efektivně vyvíjet novou funkcionalitu a snadno ji dostat do rukou uživatele, tedy aktualizovat software na nositelném zařízení, je nutné zvládnout proces kontinuálního nasazení. Ekosystém nositelných zařízení nabízí využití následujících (i dalších nezmíněných) variant nasazení.

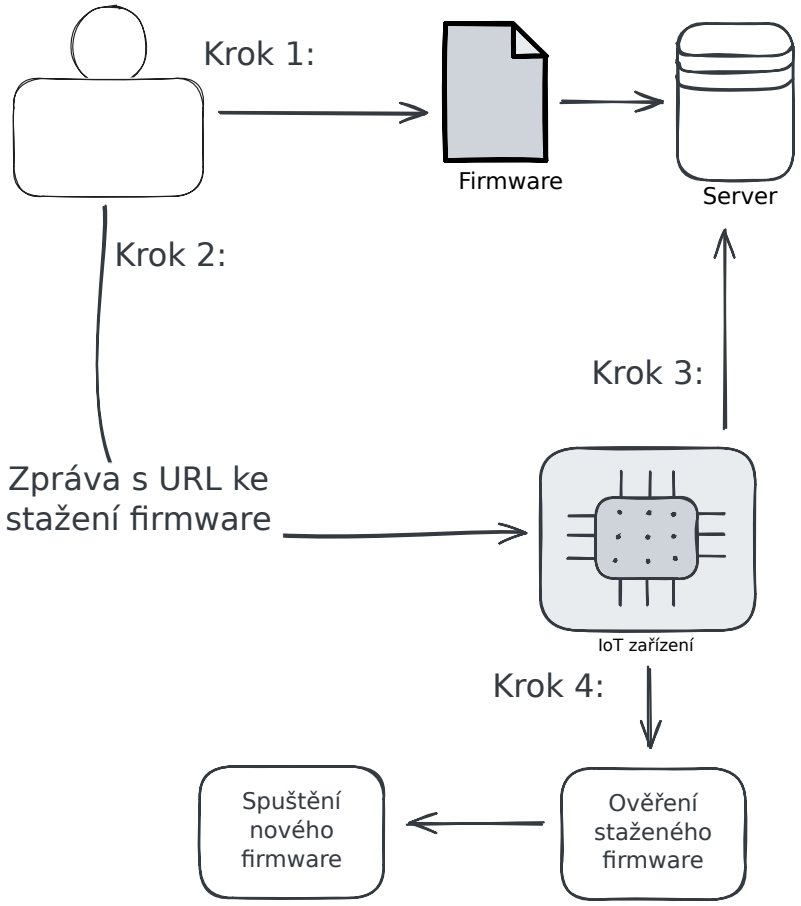
Je-li zařízení založeno na operačním systému Android, respektive jeho nadstavbě zvané WearOS vyvinuté společností Google, je možné po dokončení vývoje aplikaci publikovat na Google Play. Tato cesta zahrnuje proces schvalování, během kterého společnost Google ověří, že aplikace splňuje všechny stanovené předpisy [26]. Po schválení a úspěšné publikaci na obchod Play lze aplikaci stáhnout a nainstalovat přímo prostřednictvím nositelného zařízení.

Jedním z nejrozšířenějších mikrokontrolerů na vestavěných zařízeních je ESP32 [4], využívající Espressif IoT Development Framework (IDF) [17]. Tento nástroj poskytuje API pro aktualizaci firmware[1] přes HTTP protokol, známou jako Over The Air (OTA) aktualizace [18]. První instalace firmware se zpravidla provádí fyzickým nahráním do flash paměti, nicméně následné aktualizace lze realizovat přes WiFi či Bluetooth.

[1]Firmware obecně označuje software poskytující nízkoúrovňovou kontrolu nad fyzickým hardware zařízení. U jednoduchých zařízení se tímto pojmem označuje software, který na něm jednoduše běží. Naopak u komplexnějších zařízení může firmware poskytovat abstrakci hardware pro vysokoúrovňový software, například operační systém.

V neposlední řadě lze využít nahrávacích možností nástroje PlatformIO, je-li při vývoji systému využit. PlatformIO představuje univerzální nástroj pro vývojáře softwaru, kteří budují aplikace pro vestavěná zařízení [58]. Jedná se o ekosystém otevřených zdrojových kódů zaměřený na vývoj pro IoT, podporující různé platformy včetně ESP32.

Pomocí příkazového řádku (CLI) je možné nahrát sestavený firmware na připojené zařízení podle poskytnuté konfigurace. Nahrání lze provádět přes USB port, Over The Air (OTA) aktualizací, nebo přímou cestou do flash paměti.



Obrázek 2.1: Běžný způsob vzdálené aktualizace firmware na vestavěném zařízení Over The Air.

2.4 Shrnutí

Internet věcí představuje síť propojených fyzických objektů vybavených senzory a softwarovými prostředky pro sběr a sdílení dat. Tato technologie

nachází široké uplatnění v rozličných oblastech, od průmyslu a dopravy až po zdravotnictví a chytré domácnosti. V tomto kontextu byl vyvinut systém *SmartCGMS* zaměřený na zpracování fyziologických signálů. Systém klade důraz na modularitu, škálovatelnost a kompatibilitu s různými platformami, včetně nositelných zařízení. Pro zajištění snadné aktualizace softwaru a zpřístupnění nejnovějších funkcí uživatelům je nutné precizně zvládnout proces jeho nasazení. V závislosti na operačním systému a typu zařízení existuje několik možností zahrnující publikování aplikace na Google Play, OTA aktualizace pro zařízení s mikrokontrolerem ESP32 či využití nástroje PlatformIO.

Data, která IoT zařízení generují je nutné efektivně zpracovávat. Pokud není navíc distribuovaný software určen pro širokou veřejnost, je třeba řešit centrální správu nositelných zařízení, na které bude nasazen. Řešením obou problémů je implementace škálovatelné webové infrastruktury, která bude schopna příchozí tok dat zpracovávat a zároveň bude poskytovat vhodné rozhraní pro manipulaci s jednotlivými zařízeními, která data odesílají. Charakteristiky takové infrastruktury budou představeny v následující kapitole.

3 Architektura cloudové aplikace

V této kapitole bude vysvětlen pojem *cloud computing*, vlastnosti cloudových aplikací a budou představeny jednotlivé druhy cloudových služeb, které je možné u různých poskytovatelů využít. Vysvětlen bude i princip architektury mikroslužeb, který je s cloudovými aplikacemi úzce spjatý a budou uvedeny klíčové podpůrné služby, které jsou běžnými součástmi jejich infrastruktury. Nakonec budou představeny takzvané *cloud-native* aplikace, jejich rozdíly oproti tradičním aplikacím, a metodika Dvanáctifaktorová aplikace, která z jejich principů vychází.

3.1 Cloud computing

Cloud computing je model umožňující síťový přístup na vyžádání ke sdíleným výpočetním zdrojům (např. sítím, serverům, úložištím, aplikacím a službám), které lze rychle poskytnout a uvolnit s minimálními nároky na správu a bez nutnosti přímé interakce s poskytovatelem služeb [39]. Aplikace tedy neoperují přímo na stroji uživatele, nýbrž jsou dostupné přes Internet například pomocí webového prohlížeče. Výhodou tohoto přístupu je vysoká škálovatelnost cloudových aplikací, prakticky neomezený datový prostor i výpočetní výkon a dostupnost systému prakticky odkudkoli, kde je přístup k Internetu [24].

Národní institut standardů a technologií (NIST) přiřazuje cloudu několik klíčových charakteristik [39].

Samoobsluha na vyžádání Uživatelé mohou samostatně získávat výpočetní zdroje, jako je serverový čas a síťové úložiště, automaticky a bez přímé interakce s poskytovateli služeb.

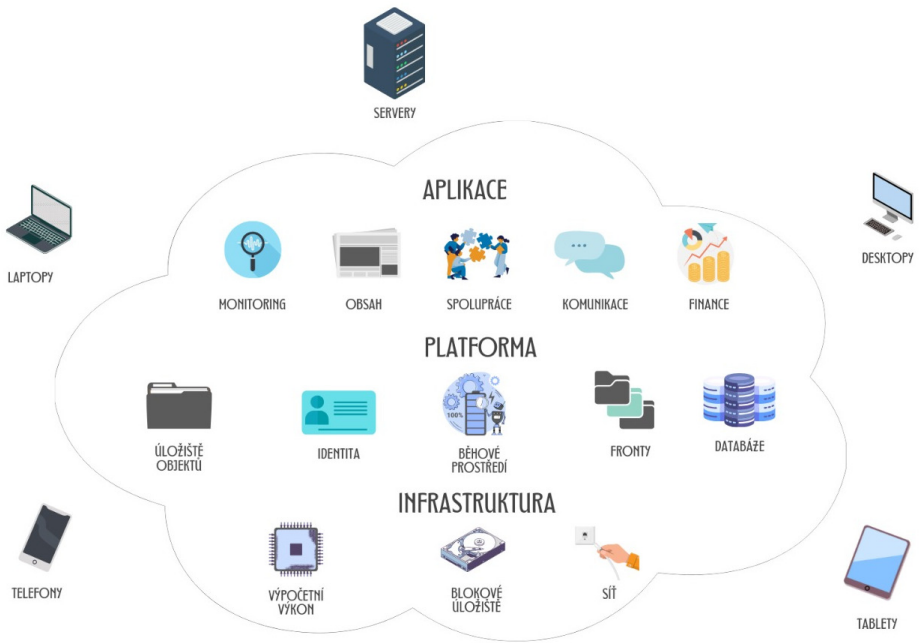
Široký přístup k síti Služby jsou přístupné přes síť prostřednictvím standardních mechanismů a je možné je využívat pomocí různorodých klientských platforem, včetně mobilních zařízení, tabletů, notebooků a pracovních stanic.

Sdílení zdrojů Poskytovatel konsoliduje výpočetní zdroje do jednotlivých instancí systému, kterými poskytuje služby více spotřebitelům. Fyzické a

virtuální zdroje se dynamicky přesouvají na základě poptávky spotřebitelů s určitou mírou nezávislosti na místě. Spotřebitelé sice nemají přesnou kontrolu nad umístěním zdrojů, mohou však určit abstrakce vyšší úrovně, jako je země, stát nebo datové centrum. Zdroje zahrnují úložiště, zpracování, paměť a šířku pásma sítě.

Rychlá pružnost Výpočetní kapacity lze rychle poskytovat a uvolňovat, často automaticky, což umožňuje navýšení či snížení objemu výpočetních či jiných zdrojů v reakci na výkyvy poptávky. Z pohledu spotřebitele se dostupnost zdrojů zdá být prakticky neomezená a lze k nim přistupovat v libovolném množství a kdykoli.

Měřená služba Cloudové systémy využívají automatické řízení a optimalizaci zdrojů s využitím možností měření přizpůsobených konkrétnímu typu služby (např. úložiště, zpracování, šířka pásma a aktivní uživatelské účty). To umožňuje monitorování, kontrolu a transparentní vykazování a zpoplatnění využití zdrojů, z čehož těží jak poskytovatel, tak spotřebitel služby.



Obrázek 3.1: Znázornění standardních prvků infrastruktury, tvořící cloudový výpočetní systém.

3.2 Druhy cloudových služeb

Cloudové služby lze rozdělit do několika modelů, přičemž každý odpovídá odlišným potřebám podniků, představuje různou úroveň abstrakce a poskytuje odlišné funkce a výhody.

3.2.1 Infrastruktura jako služba

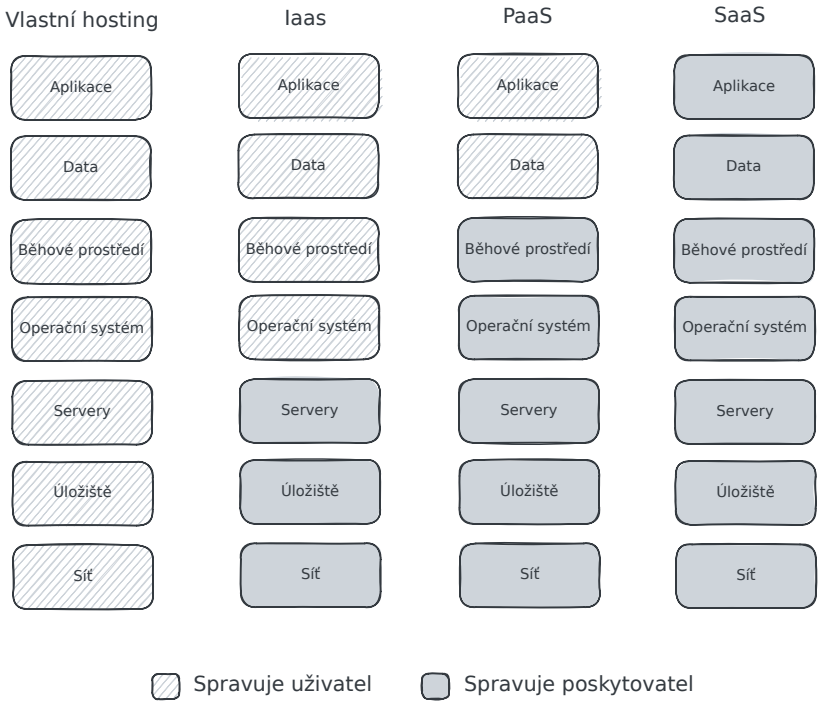
Infrastruktura jako služba (angl. Infrastructure as a Service (IaaS), dále jen IaaS) je cloud computing model, který poskytuje virtualizované výpočetní zdroje prostřednictvím Internetu. Uživatelé si mohou pronajmout virtuální počítače, úložiště a síťovou infrastrukturu na bázi platby podle potřeby. IaaS nabízí nejvyšší úroveň flexibility a umožňuje podnikům škálovat specifické prvky infrastruktury v závislosti na jejich potřebách. Mezi oblíbené poskytovatele IaaS patří Amazon Web Services (AWS), Microsoft Azure a Google Cloud Platform (GCP). Tento model je vhodný zejména pro podniky, které vyžadují detailní kontrolu nad svou infrastrukturou bez zátěže spojené s údržbou fyzického hardware [24].

3.2.2 Platforma jako služba

Principem platformy jako služby (angl. Platform as a Service (PaaS), dále jen PaaS) je sdílení jedné infrastrukturní platformy a společných služeb, jako je ověřování, autorizace a účtování, vícero aplikacemi. PaaS umožňuje vývojářům vytvářet webové aplikace bez nutnosti instalovat nástroje do svých počítačů a nasazovat je, aniž by se starali o hardware nebo se o něj zajímali [7]. PaaS je postaveno na IaaS a využívá víceuživatelské nástroje pro vývoj software společně s víceuživatelskou infrastrukturou pro jeho nasazení [2]. Vývojáři programují aplikace a definují potřebné závislosti, a platforma je následně odpovědná za správu, provoz a publikaci těchto aplikací spolu s vytvořením potřebné infrastruktury. Tu spravuje poskytovatel platformy v kontrastu s IaaS, kdy je vyžadována správa infrastruktury uživatelem [24]. Mezi nejznámější platformy patří například Google App Engine, Amazon Web Services a Azure Cloud Services.

3.2.3 Software jako služba

Software jako služba [angl. Software as a Service (SaaS), dále jen SaaS] poskytuje softwarové aplikace přes Internet, takže uživatelé nemusejí aplikace instalovat, spravovat ani udržovat. Software běží na infrastruktuře poskytovatele (potenciálně na infrastruktuře jeho poskytovatele), ačkoli vypadá jako



Obrázek 3.2: Vrstvy infrastruktury a jejich odpovědné osoby napříč cloudovými službami.

součást zákazníkovy interní sítě. Veškerá zodpovědnost za provoz a správu aplikace i serverů spadá na poskytovatele, zatímco zákazník si zachovává veškerou administrativní kontrolu [72]. Uživatelé mohou k software přistupovat prostřednictvím webového prohlížeče, což z něj činí pohodlné a nákladově efektivní řešení. Mezi příklady SaaS patří Google Workspace, Microsoft 365 a Salesforce. Služba SaaS je výhodná pro podniky, které se chtějí soustředit na používání software a neřešit složitosti spojené s jeho nasazením a údržbou.

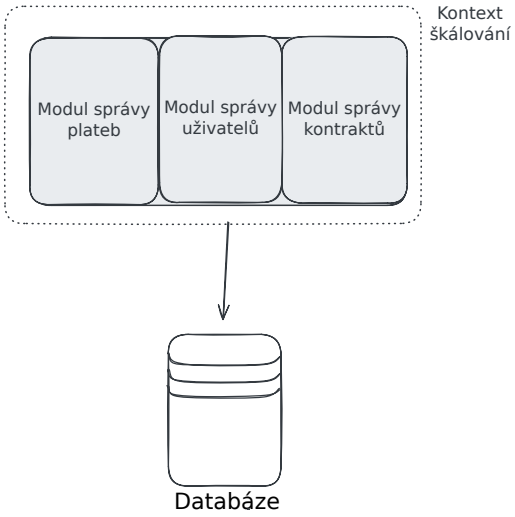
Z hlediska vývoje cloudových aplikací jsou tedy zajímavé modely IaaS a PaaS, umožňující různé způsoby nasazení a správy software, který lze dále poskytovat uživatelům jako službu. Konkrétní výběr již závisí na specifických požadavcích a možnostech poskytovatele software v závislosti na dříve zmíněných charakteristikách obou modelů.

3.3 Architektura mikroslužeb

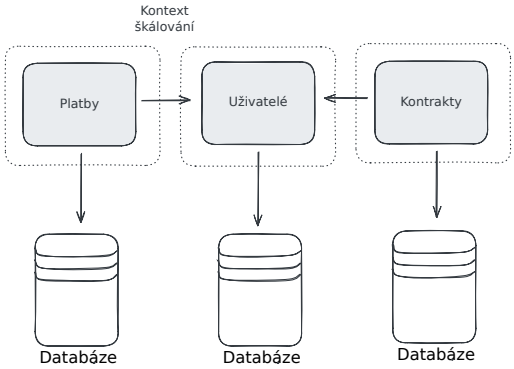
Moderním přístupem k vývoji cloudových aplikací se v dnešní době stává využití architektury mikroslužeb [1]. Mikroslužby lze charakterizovat jako malé, samostatné aplikace, které mají několik klíčových vlastností. Jsou nezávisle nasaditelné, lze je samostatně škálovat bez ohledu na zbytek systému,

a lze samostatně verifikovat jejich funkčnost. V neposlední řadě se mikroslužby řídí principem jediné zodpovědnosti (SRP), kdy mají jednu jedinou zodpovědnost a zároveň i jediný důvod se měnit nebo být nahrazeny [69].

Opačným přístupem k mikroslužbám je architektura monolitická. Monolitické architektury zapouzdřují veškerou funkcionalitu do jediné aplikace, což s sebou pro méně složité systémy přináší řadu výhod včetně snadného vývoje, testovatelnosti, nasazení i škálování. S růstem systému se ovšem začínají projevovat slabiny tohoto návrhu, například složitější údržba nebo dokonce zpomalení cyklu vývoje funkcionalit a oprav chyb [11]. Rozdíl mikroslužeb oproti monolitickým aplikacím je vyobrazen na obrázcích 3.3 a 3.4.



Obrázek 3.3: Monolitická aplikace je nasazována a škálována jako jeden celek. Při drobné změně jednoho modulu je nutné nasadit kompletní aplikaci.



Obrázek 3.4: Mikroslužby jsou nasazovány a škálovány samostatně. Pokud navíc dojde k selhání jedné, ostatní mohou dále fungovat v omezeném režimu.

K dalším výhodám architektury mikroslužeb je možnost heterogenity využitých technologií a výsledná robustnost celého systému. Pokud část systému selže, zbytek může pokračovat ve výpočtu, pokud je možné selhání vhodně izolovat. Ve světě monolitických aplikací systém přestane fungovat, pokud kterákoli jeho část selže. Mikroslužby lze vytvořit takovým způsobem, že jsou schopny ustát selhání ostatních mikroslužeb a jejich chování pouze vhodným způsobem degraduje [49]. Mikroslužby také významným způsobem snižují komplexitu původně monolitického systému, kterou z vnitřní implementace komponent a modulů aplikace přesouvají na infrastrukturu [69].

Mnoho výhod této architektury lze obecně přiřadit libovolnému distribuovanému systému. Oproti standardním distribuovaným architekturám lze využitím architektury mikroslužeb, která striktněji předepisuje hranice mezi jednotlivými mikroslužbami, dosáhnout významných benefitů díky využití principů ukrývání informací a doménově řízeného návrhu [49].

3.4 Podpůrné služby

Pro běh cloudových aplikací je často zapotřebí nasadit i další podpůrné služby jako součást jejich infrastruktury. Cloudové aplikace by měly logovat informace o průběhu výpočtů, které provádějí, aby bylo možné sledovat jejich průběh a v případě selhání jeho důvod zpětně dohledat [37]. Logy je nutné shromažďovat a odesílat do služby pro agregaci logů, tzv. agregátoru. Agregátory logů běžně vyžadují další specializovaná úložiště, kde je možné logy ukládat pro dlouhodobé využití [32]. Další samostatnou službou bývá frontend pro vizualizaci logů a metrik, který je připojen na API agregátoru logů, na kterém logy získává pomocí strukturovaných dotazů. Kromě logů lze aplikaci monitorovat sběrem interních metrik, jako např. délka trvání obsluhy požadavků, doba odpovědi externí služby na požadavek či doba odezvy databáze. K tomu slouží monitorovací služby, jako je Prometheus [70], které lze v aplikačním kódu nakonfigurovat pro sledování konkrétních událostí.

Součástí cloudových aplikací bývá i centrální autorizační infrastruktura, o kterou se ostatní aplikace opírají. Implementace vlastní autentizace v každé aplikaci je v rozsáhlém prostředí neefektivní a tudíž je centrální autentizace ideálním řešením. Díky centralizovaně řízené autorizaci mohou aplikace sdílet řízení přístupu k jednotlivým zdrojům podle uživatelských rolí a jejich oprávnění [13].

Pro implementaci podobných mechanismů je často využíván koncept *API gateway*, kdy je samotná aplikace veřejně nepřístupná a do sítě je vysta-

vena pouze jedna centrální služba, která kontroluje veškerý provoz, ověřuje autentizaci a autorizaci jednotlivých požadavků, řeší limity dotazů na rozhraní aplikace a chová se jako reverzní proxy[2] [45].

Komunikace mezi samotnými aplikacemi může být synchronní, např. přes jejich API pomocí HTTP, nebo asynchronní, pokud je nutné architekturu dekomponovat do pružnějšího, distribuovaného celku. K tomu jsou běžně využívány fronty zpráv pracující na *publisher-subscriber* principu[3] [71].

Relevance některých výše zmíněných podpůrných služeb závisí na konkrétní aplikaci, nicméně služby jako sběr metrik nebo centrální sběr a agregace logů jsou dnes de-facto standardním nástrojem pro pozorování chování cloudových aplikací v čase. V následující části bude představen princip *cloud-native* aplikací, pro jejichž vývoj jsou zmíněné podpůrné služby často využívány.

3.5 Nativní cloudové aplikace

Nativní cloudové (*cloud-native*) aplikace jsou navrženy a optimalizovány tak, aby plně využívaly možnosti cloud computingu a dynamické a škálovatelné povahy cloudových prostředí. Od tradičních aplikací se cloud-native liší několika klíčovými vlastnostmi.

Architektura mikroslužeb Cloudové aplikace jsou často implementovány architekturou mikroslužeb, kdy je aplikace rozdělena na malé nezávislé služby, které lze samostatně vyvíjet, nasazovat a škálovat. Tento modulární přístup zvyšuje flexibilitu celé infrastruktury, agilitu vývojového týmu a umožňuje optimalizovat využití zdrojů jednotlivých služeb [24].

Kontejnerizace Cloudové aplikace běžně využívají kontejnerizační technologie, jako například Docker, k zapouzdření a zabalení každé mikroslužby spolu s jejími závislostmi. Kontejnery zajišťují konzistentní chování napříč různými prostředími, usnadňující bezproblémové nasazení a škálování těchto aplikací. Samotné instance aplikačních kontejnerů jsou dále spravovány orchestračními nástroji, jako jsou Kubernetes [65], které je automaticky škálují

[2]Reverzní proxy je prostředník mezi klientskou aplikací a serverem, která přijímá komunikaci, určenou serveru, kterou mu dále předává. Odpovědi serveru jsou opět prostřednictvím reverzní proxy předány klientovi. Účelem nasazení reverzní proxy může být balancování zátěže, caching nebo zvýšení bezpečnosti.

[3]*Publisher* publikuje zprávy do fronty a dále nečeká na jejich zpracování. *Subscriber* čeká na příchozí zprávy, které následně zpracovává.

a uvádí celkový stav infrastruktury do souladu se stavem definovaným v konfiguraci [23].

Praktiky DevOps Cloud-native vývoj je úzce spojen s postupy DevOps, které kladou důraz na spolupráci mezi vývojovými a provozními týmy a automatizaci všech procesů spojených s vývojem a nasazováním. K automatizaci testování, integrace a nasazení se používají procesy kontinuální integrace (CI) a kontinuálního nasazování (CD), umožňující rychlejší a spolehlivější vydávání nových verzí aplikace [5]. Pojem DevOps bude dále představen v sekci 4.4.

Škálovatelnost a odolnost Cloudové aplikace jsou navrženy tak, aby se dynamicky škálovaly na základě poptávky a upřednostňovaly robustnost a odolnost proti chybám. Mohou využívat funkce automatického škálování v cloudových platformách a efektivně se přizpůsobovat měnícímu se pracovnímu zatížení. Vyrovnávání zátěže a mechanismy automatické obnovy zajišťují jejich vysokou dostupnost [23].

Cloud-native vývoj se od tradičního vývoje aplikací liší v mnoha aspektech. V tradičních aplikacích se často používá monolitická architektura, kdy je celá aplikace vytvořena jako jeden pevně integrovaný celek. Horizontální škálování či samostatná aktualizace konkrétních systémových komponent proto může být složitá. Naproti tomu architektura mikroslužeb cloud-native aplikací usnadňuje údržbu a zjednodušuje úpravy dílčích komponent bez dopadu na celý systém [49].

Díky využití kontejnerizace jsou cloud-native aplikace přenositelné napříč více prostředími, ve kterých se chovají předvídatelně. Tradiční aplikace jsou naopak často silně integrovány do konkrétních běhových prostředí a při přechodu do prostředí nových mohou vznikat nečekané potíže.

Osvojením praktik DevOps jsou vývojové týmy cloud-native aplikací schopny efektivně spolupracovat na vývoji i nasazení systému pomocí automatizace CI/CD. Tradiční oddělení vývojových a provozních týmů naproti tomu může vést k pomalejším cyklům vydávání nových verzí a zvýšenému potenciálu výskytu chyb.

3.6 Dvanáctifaktorová aplikace

Na základě pozorování a nasazování stovek cloudových aplikací vznikla metodika Dvanáctifaktorová aplikace, která si klade za cíl eliminovat systema-

ticky vznikající problémy při provozu cloudových aplikací a definovat sadu pravidel a zásad usnadňujících jejich vývoj [29]. Tato metodika vznikla ve spolupráci členů týmu společnosti Heroku, která je významným poskytovatelem cloudových služeb v modelu Platform as a Service (PaaS).

Nativní cloudová aplikace by měla explicitně deklarovat a izolovat vnější závislosti, externalizovat konfiguraci a uchovávat ji v běhovém prostředí, podpůrné služby využívat jako vnější zdroje a maximalizovat svoji robustnost pomocí krátkých časů startu a šetrného vypnutí. Každá aplikace by měla být spouštěna jako jeden či více izolovaných procesů a škálovat spouštěním procesů nových [74]. Fáze sestavení, testování a nasazení by měly být striktně odděleny a vývojové, testovací a produkční prostředí by si měla být co nejvíce podobná, aby se zabránilo případným chybám kvůli jejich nekonzistenci.

3.7 Shrnutí

Tato kapitola se zaměřila na koncept cloud computing a jeho principy. Byly charakterizovány různé modely cloudových služeb, včetně IaaS, PaaS a SaaS, a jejich specifické vlastnosti.

Představena byla i architektura mikroslužeb, která představuje moderní přístup k vývoji cloudových aplikací. Byly zdůrazněny její benefity, jako je flexibilita, škálovatelnost a robustnost, a uvedeny rozdíly oproti tradičním monolitickým systémům.

Kapitola se dále věnovala podpůrným službám, které tvoří součást cloudové infrastruktury. Byly zmíněny nástroje pro logování, monitorování, autentizaci, autorizaci a komunikaci mezi mikroslužbami.

Byly charakterizovány tzv. cloud-native aplikace, které plně využívají dynamické a škálovatelné vlastnosti cloudového prostředí. Nakonec byla představena metodika Dvanáctifaktorová aplikace, která definuje sadu principů pro usnadnění vývoje a nasazování cloudových aplikací.

V následující kapitole budou představeny konkrétní metodiky a postupy, využitelné pro nasazování a správu cloudových aplikací, včetně jejich infrastruktury a podpůrných služeb.

4 Kontinuální integrace a nasazování pro cloud

V této kapitole budou představeny principy DevOps, kontinuální integrace a nasazování, jejich role v moderním vývoji software a specifika těchto metodik při vývoji nativních cloudových aplikací.

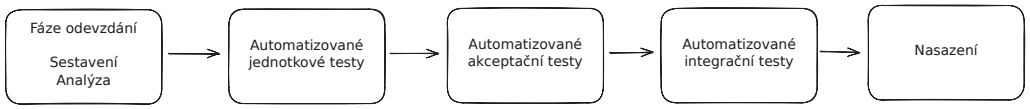
4.1 Kontinuální integrace

Kontinuální integrace [angl. Continuous Integration (CI), dále jen CI] je proces vývoje software, při kterém každý člen týmu pravidelně začleňuje své vlastní změny, a změny svých kolegů, do základny zdrojového kódu. Tyto integrace se ověřují pomocí automatizovaných sestavení (včetně testů), aby se co nejrychleji odhalily možné integrační chyby [22]. Díky CI jsou softwarové společnosti schopny zkrátit cykly vydávání nových verzí, zvýšit kvalitu vydávaného software a zvýšit týmovou produktivitu [21].

4.2 Kontinuální dodávka

Cílem kontinuální dodávky [angl. Continuous Delivery (CDE), dále jen CDE] je zajištění produkční připravenosti software, jakmile projde sadou povinných automatizovaných testů a procesem zajištěním kvality [19]. CDE je přirozenou progresí od CI, jejíž principy jsou i nadále aplikovány a jsou rozšířeny o automatizaci dalších testů a dodání software do produkčního prostředí [63]. Hlavním benefitem CDE je eliminace rizik spojených s manuálním nasazením do produkčního prostředí. Zároveň dochází ke snížení nákladů na nasazení a zkrácení doby trvání dodávky nových funkcí zákazníkovi [10]. Primárním prostředkem realizace CDE je tzv. *deployment pipeline*[4]. Jedná se o automatizovanou implementaci procesu sestavení, testování a nasazení aplikace, jejíž specifika se liší v závislosti na potřebách konkrétních organizací. Principy ovšem zůstávají stejné [19]. Konečné potvrzení automatického nasazení nové verze ovšem zpravidla zůstává manuální. Příklad *deployment pipeline* je znázorněn na obrázku 4.1.

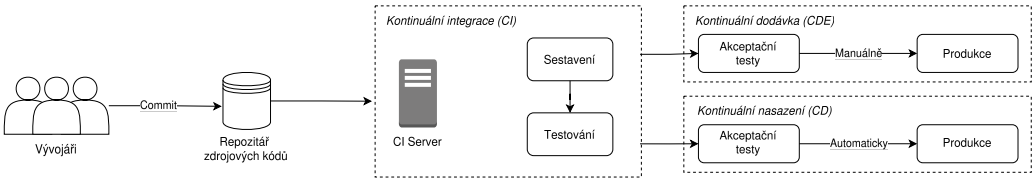
[4]V doslovném překladu potrubí. V kontextu CI se jedná o sekvenci několika po sobě jdoucích, potenciálně paralelních, kroků.



Obrázek 4.1: Příklad deployment pipeline.

4.3 Kontinuální nasazení

Kontinuální nasazení [angl. Continuous Deployment (CD), dále jen CD], je praktikou, při které je prostřednictvím *deployment pipeline* nasazena změna do produkce automaticky, jakmile dojde k odevzdání změn vývojáři [19]. CD je tedy dalším přirozeným krokem od CDE, obdobně jako CDE následuje po CI, při kterém dochází k automatizaci nasazení a eliminaci jakéhokoli manuálního kroku [63]. Lze tedy odvodit implikaci praxe CDE při praktikování CD. Tato implikace ovšem není obousměrná [19]. V literatuře nicméně stále existuje mnoho probíhajících diskusí o rozdílech a definicích CDE a CD [21, 73]. Vzájemný vztah jednotlivých praktik je znázorněn na obrázku 4.2.



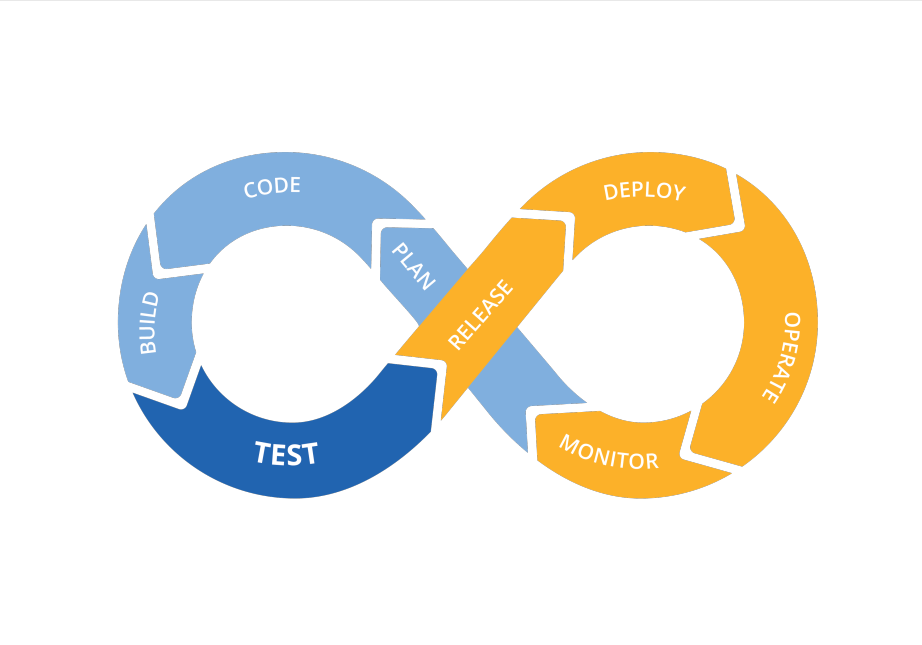
Obrázek 4.2: Vizualizace praktik CI, CD, CDE a jejich vztah.

4.4 DevOps

DevOps je IT metodika, která úzce propojuje vývojářské a provozní týmy, přičemž klade důraz na proaktivní integraci jejich aktivit. Cílem DevOps je zkrácení doby potřebné pro produkci a nasazení software a současně zachování jeho vysoké kvality a stability. Oproti standardním metodikám, ve kterých jsou odděleny týmy vývoje a provozu, vyžaduje metodika DevOps zásadní organizační změnu [36].

Týmy jsou multifunkční a jsou složené z vývojářů, provozních pracovníků a dalších specialistů, kteří se podílí na celém životním cyklu software. Každý člen týmu se podílí na vývoji nových funkcí systému a je zodpovědný za jejich doručení do rukou uživatele. Klíčovým principem DevOps je automatizace procesů integrace, testování, nasazení a monitoringu změn pomocí CI/CD, jejíž výsledkem jsou efektivnější a konzistentnější pracovní postupy, nižší podnikové náklady, dodávka systému se stává předvídatelnější a spolehlivější

a zkracuje se prodleva mezi vývojem funkcionality a uživatelskou zpětnou vazbou. Zároveň jsou eliminovány problémy způsobené špatnou komunikací napříč původně samostatnými týmy [15, 36]. Prolínání aktivit vývojových a operačních týmů je znázorněno na obrázku 4.3.



Obrázek 4.3: DevOps propojuje vývojové, testovací a provozní týmy sjednocením dílčích procesů do kontnuálního procesu vývoje, testování, nasazení a monitorování.

Zdroj: https://www.logolynx.com/images/logolynx/e7/e752ab38cf15696a9dcc96b811605b12.png

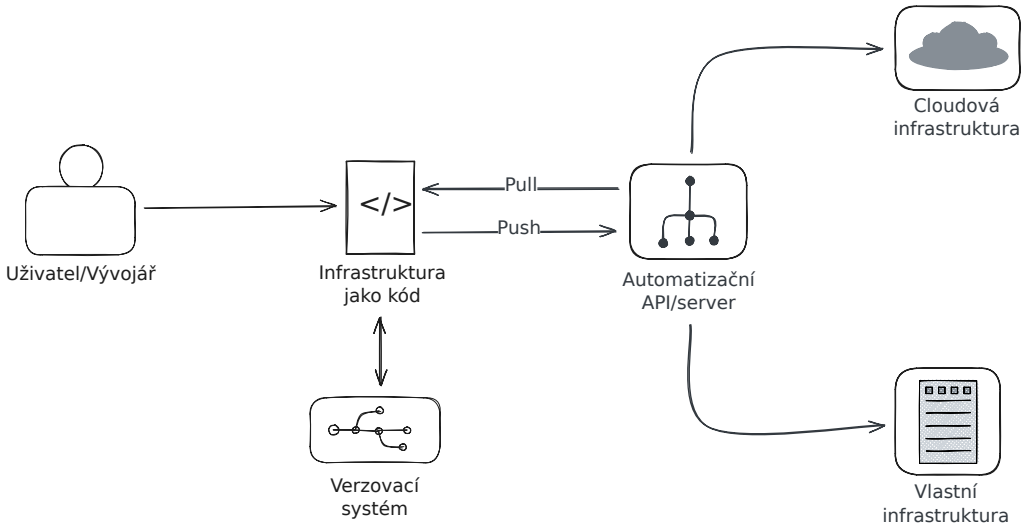
Ačkoli je možné využít praktiky DevOps i v monolitických projektech, implementace této metodiky je v praxi nejefektivnější spolu s architekturou mikroslužeb, jelikož oba přístupy kladou důraz na vývoj v malých týmech [6]. Při migraci na architekturu mikroslužeb nebo přechodu do cloudu se osvojení DevOps jeví jako klíčové [5].

4.5 Infrastruktura jako kód

Moderní poskytovatelé cloudových služeb běžně poskytují webové rozhraní pro tvorbu a správu všech prvků infrastruktury. Tento způsob údržby se ovšem dlouhodobě jeví jako neudržitelný, jelikož v podstatě neexistuje audit, který by zaznamenával historii změn. Vytvoření testovacího prostředí, které

odpovídá tomu produkčnímu, nelze provést automaticky a jeho manuální nastavení je náchylné na chyby. Pokud dojde k systémovému výpadku, je nutné manuálně obnovit každou část infrastruktury do původního stavu [46].

Infrastruktura jako kód (angl. Infrastructure as Code (IaC), dále jen IaC) se v řešení těchto problémů inspiruje poznatky, které inženýři shromáždili v průběhu dekád vývoje software. IaC, jak je již z názvu patrné, je založeno na principu uchování konfigurace infrastruktury ve zdrojovém kódu. Konfiguraci lze interpretovat pomocí specializovaných nástrojů a ověřit její konzistenci s reálným stavem nasazené infrastruktury. V neposlední řadě lze kód spravovat pomocí verzovacích systémů a automatizovaně ho testovat [46]. Důraz je kladen na využití konzistentních, opakovatelných postupů pro zajištění a změnu infrastruktury a její konfigurace. Kód infrastruktury specifikuje jednotlivé elementy a jejich nastavení, nástroj pro správu infrastruktury následně sjednocuje stav konkrétní instance s aktuálním stavem kódu. Je vytvořena nová pokud neexistuje, nebo je upravena tak, aby odpovídala definici. Tento princip je znázorněn na obrázku 4.4.



Obrázek 4.4: Infrastruktura jako kód.

IaC vzniká z principů DevOps, který propaguje definici konfigurací pomocí standardizovaného jazyka. Stejně tak, jako je definován aplikační kód, který udává seznam svých závislostí, je zaznamenána podoba infrastruktury ve strukturovaném jazyce, který lze strojově interpretovat [3]. Pomocí automatizačních nástrojů lze jednoduchým příkazem zajistit spuštění kompletní produkční infrastruktury, případně zajistit vedlejší testovací instanci, konzistentní se stavem produkce [31]. Oproti dříve zmíněným možnostem tvorby a správy infrastruktury ve webových rozhraních nebo na míru vytvořených

CLI poskytuje IaC potřebnou konzistenci, transparentnost a znovupoužitelnost, díky které lze využít rychlost nasazování změn metodik DevOps a CI/CD pro zajištění kvality systému [46]. Příklad definice infrastruktury jako kód pomocí nástroje Terraform [8] je uveden v úryvku 4.1.

```
terraform {
# Nasazujeme na Google Cloud,
#   proto uvedeno GCP jako požadovaný poskytovatel
  required_providers {
    google = {
      source = "hashicorp/google"
      version = "4.51.0"
    }
  }
}

# Inicializace poskytovatele s~ID našeho projektu
provider "google" {
  project = "<PROJECT_ID>"
}

# Vytvoření vlastní VPC
#   jako části naší infrastruktury
resource "google_compute_network" "vpc_network" {
  name = "terraform-network"
}
```

Úryvek 4.1: Příklad definice infrastruktury jako kód pomocí nástroje Terraform.

4.6 Shrnutí

Implementace principů CI/CD a DevOps, které byly v této kapitole představeny, v kombinaci s cloudovými technologiemi umožňuje efektivní vývoj a nasazování nativních cloudových aplikací s vysokou kvalitou a rychlostí. Aplikování těchto principů vede ke zkrácení vývojového cyklu, zvýšení spolehlivosti a škálovatelnosti aplikací a celkově k zefektivnění celého procesu vývoje a provozu softwaru v cloudovém prostředí.

V neposlední řadě byl vysvětlen pojem Infrastructure as Code (IaC), kdy je konfigurace infrastruktury udržována ve zdrojovém kódu pro zajištění konzistence, transparentnosti a znovupoužitelnosti.

Následující kapitola bude zaměřena na specifikaci požadavků na navrhovanou cloudovou platformu pro vzdálenou správu nositelných zařízení a sběr dat produkovaných těmito zařízeními. Ke specifikaci i následnému návrhu infrastruktury budou využity doposud vysvětlené principy a metodik vývoje cloudových aplikací.

5 Specifikace požadavků a případy užití

V této kapitole bude popsána idea poptávaného systému a bude uveden seznam obecných a technických požadavků stanovených zadavatelem. V závěru kapitoly budou představeny navržené případy užití, které byly zadavatelem akceptovány.

5.1 Specifikace požadavků

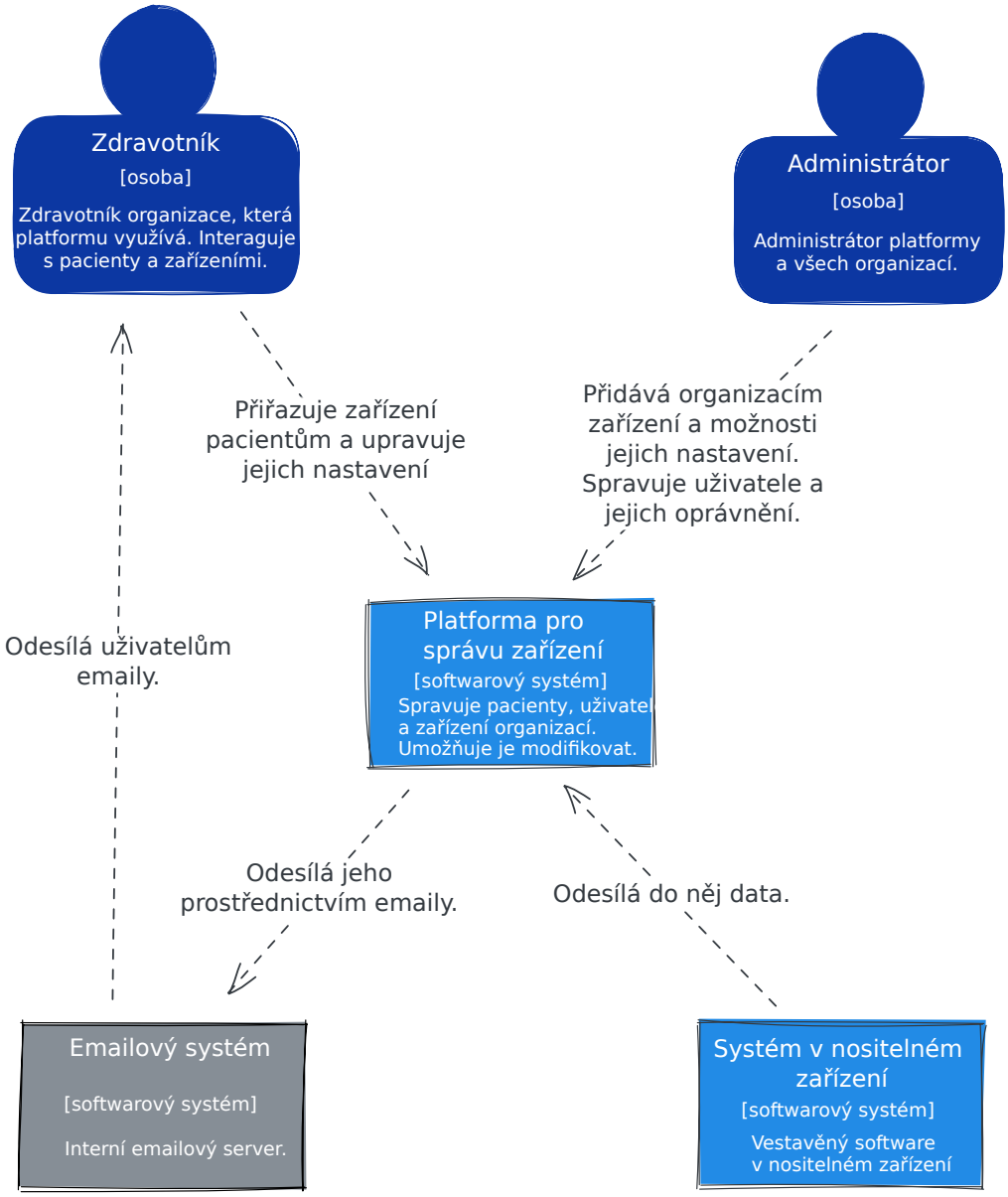
Navrhovaný systém je určen pro využití v rámci zdravotnických organizací. Každá organizace má vlastní zdravotníky, pacienty a disponuje množinou nositelných zařízení, která jsou pacientům přiřazována zdravotnickým personálem. Cílem implementovaného systému je fungování ve víceuživatelském režimu, tedy jediná instance systému bude obsluhovat několik organizací a jejich personál. Bude se tedy jednat o Software as a Service (SaaS).

Vzdálená správa nositelných zařízení Platforma by měla umožňovat uživatelsky snadné přidání nových zařízení organizaci, které bude možné jejím prostřednictvím vzdáleně spravovat. Přiřadit organizaci zařízení by mělo být umožněno pouze administrátoru platformy.

Uživatelsky snadný způsob personalizace zařízení Nové zařízení by mělo být možné pohodlně spárovat s pacientem a personalizovat ho pomocí konfigurace odpovídající využití zařízení. Proces personalizace by měl být zahájen naskenováním QR kódu [30] zobrazeným na displeji nositelného zařízení, následován nastavením zařízení a přiřazením pacientovi. Tento úkon je primárně určen pro zdravotnický personál, který je s pacienty v kontaktu.

Sběr dat Software, který na spravovaných zařízeních běží, produkuje množství dat určených ke kolekci a zpracování. Je nutné implementovat jejich efektivní a škálovatelný způsob sběru, jelikož je žádoucí je později vyhodnocovat a monitorovat. Vestavěný software na nositelných zařízeních je pod kontrolou zadavatele, tudíž předpokládáme, že je v zařízení možné implementovat vlastní způsob odesílání dat.

Tyto vysokoúrovňové požadavky na systém budou dále upřesněny pomocí požadavků na softwarovou realizaci systému. Vysokoúrovňový pohled na navrhovaný systém je znázorněn na obrázku 5.1 pomocí notace modelu C4 [9].



Obrázek 5.1: Diagram kontextu systému v notaci modelu C4.

5.2 Požadavky na realizaci systému

Zadavatelem práce byla předem stanovena sada požadavků týkajících se technické realizace, kterou je pro akceptaci projektu nutné dodržet. Tyto požadavky budou v následujících odstavcích krátce přestaveny.

Webové rozhraní a server s databází Realizace projektu bude obsahovat webové rozhraní, prostřednictvím kterého budou uživatelé se systémem interagovat. Prohlížečová aplikace bude komunikovat se serverem, který bude požadavky obsluhovat a validovat. Server by měl zároveň být schopen zpracovávat data, která spravovaná nositelná zařízení vysílají. Perzistentním úložištěm bude databáze, do které bude server ukládat veškerá data.

Integrace autentizační a autorizační služby Součástí platformy by měla být autentizační služba, která bude obsluhovat správu uživatelů. Měla by provádět jejich autentizaci[5] i autorizaci[6]. Respektovat by měla předem zadané uživatelské role *administrátor*, *zdravotník* a *pacient*. V závislosti na uživatelských rolích by měl být umožněn, resp. zamítnut, přístup k různým zdrojům.

Samoobslužná registrace uživatelů Přidání uživatelů do systému by mělo vyžadovat co nejmenší administraci ze strany správce aplikace. Uživatelé by měli obdržet pozvánku, přičemž jim bude umožněno si svůj uživatelský účet svépomocí založit a aktivovat.

V rámci realizace je důležité dbát na víceuživatelskou povahu systému. Platforma je zamýšlena pro využití více organizacemi zároveň, přičemž každá organizace disponuje vlastními zařízeními a spravuje vlastní zdravotnický personál a pacienty. Organizace se navzájem nesmí ovlivňovat a nesmí jim být umožněn přístup k datům jiné organizace. Všechny organizace bude možné spravovat jediným administrátorem celého systému.

5.3 Případy užití

Systém bude uživatelům umožněno využívat různými způsoby s odlišnými omezeními v závislosti na jejich rolích. Kompletní výčet všech případů užití

[5]Ověření totožnosti uživatele. Tedy, je-li uživatel tím, za koho se vydává. Často řešeno použitím přístupových klíčů.

[6]Kontrola oprávnění uživatele. Při autorizaci je ověřeno právo uživatele provést požadovanou akci na základě jeho role a přidělených přístupových práv.

je znázorněn pomocí diagramů případů užití v příloze A.

Administrátor Administrátor platformy bude oprávněn spravovat všechny organizace i jejich uživatele prostřednictvím autorizační služby. Umožněno mu bude vytvořit nového uživatele a odeslat mu pozvánku s přístupem do systému. Zároveň bude moci uživatelské účty odstranit či zablokovat.

Z oblasti správy nositelných zařízení bude pod administrátorský přístup omezena možnost přidávat do systému nová zařízení, přiřazovat je organizaci a případně je ze systému odstranit. V neposlední řadě bude administrátor oprávněn organizacím upravovat seznam různých profilů zařízení, které budou blíže specifikovat druh nositelného zařízení a způsob jeho využití. Kompletní diagram administrátorských případů užití je znázorněn na obrázku A.1.

Zdravotnický personál Po obdržení pozvánky se budou zdravotníci moci do systému zaregistrovat nastavením svého hesla. Profil bude možné později upravit změnou hesla nebo osobních údajů. Zvolené heslo půjde později změnit či obnovit v případě jeho zapomenutí. Zdravotníkům bude dále umožněno kompletně spravovat seznam pacientů organizace. Možné bude přidávat nové pacienty, upravovat jejich detailní informace a případně je ze systému odstranit.

Proces personalizace nositelných zařízení bude také spadat pod agendu zdravotníků, a to od spuštění zařízení až po přiřazení nového zařízení pacientovi. Následně budou moci detaily zařízení upravovat a případně jej od pacienta odpárovat. Odpárovaná zařízení bude možné opět přiřadit libovolnému pacientovi. Tedy i v případě, že mu již některé zařízení přiřazeno je. Diagram případů užití zdravotnického personálu je uveden na obrázku A.3.

Pacient Pacientům vstup do systému umožněn nebude. Pacientova role spočívá pouze v užívání nositelných zařízení, která mu budou zdravotnickým personálem přidělena. Tento jednoduchý případ užití je vyobrazen v diagramu A.2.

5.4 Shrnutí

Předmětem této kapitoly bylo představení klíčových charakteristik projektu a sepsání specifikace požadavků. V druhé části kapitoly byly navrženy případy užití rozdělené podle uživatelských rolí. Jejich kompletní výčet byl vizualizován pomocí diagramů případů užití v příloze A.

V následující kapitole bude navržena vhodná infrastruktura systému, která bude splňovat specifikované požadavky na systém a umožňovat realizaci případů užití, které byly definovány v této kapitole.

6 Návrh infrastruktury

V této kapitole budou představeny jednotlivé možnosti dekompozice architektury navrhovaného systému. Z rešerše možných řešení bude vybrána architektura vyhovující účelům tohoto projektu a uvedeny budou konkrétní technologie, kterými lze vybrané dekompozice docílit. Na závěr bude představen způsob nasazení celé infrastruktury prostřednictvím poskytovatele cloudových služeb.

6.1 Webový portál

Webovou aplikaci, pomocí které budou uživatelé se systémem interagovat lze koncepčně rozdělit na uživatelské rozhraní a server. Zároveň potřebujeme serverovou službu, která bude přijímat a zpracovávat data z nositelných zařízení, a službu pro obsluhu požadavků z mobilní aplikace. Nabízí se mnoho variant dekompozice takového systému.

6.1.1 Uživatelské rozhraní

Uživatelské rozhraní ve webovém prohlížeči lze implementovat několika způsoby.

HTML [75] můžeme vykreslovat na serveru a vracet ho jako odpověď na každý požadavek. Tento server může být samostatný, přičemž by komunikoval s druhým serverem obsahujícím logiku, nebo může zpracování požadavků obsluhovat přímo server s logikou.

Moderním přístupem k vývoji webových aplikací jsou tzv. *Single-Page aplikace* (SPA), kdy je serverem vrácena pouze kostra HTML dokumentu s vloženým skriptem v jazyce JavaScript [38]. Ten se následně stará o vykreslování všech elementů dokumentu, včetně případného směrování na jiné stránky [43]. To vše za cílem zajištění vyšší interaktivnosti aplikace.

Vykreslování uživatelského rozhraní na serveru s logikou znamená značnou závislost mezi těmito dvěma částmi systému. Abychom zachovali možnost snadného přechodu mezi technologiemi na obou stranách, tuto variantu zavrhneme. Zbylé dva přístupy jsou z hlediska projektu validní a nenesou s sebou žádná zásadní rizika či komplikace. Z osobních preferencí zvolíme pro implementaci uživatelského rozhraní variantu SPA. K nástrojům, které lze pro vývoj SPA využít patří např. React.js [40], Svelte [64] či Angular.js [27]. Každý z těchto nástrojů volí jiný způsob vykreslování HTML, ve svém nitru

jsou si ovšem všechny velmi podobné a snaží se docílit totožného výsledku. Ačkoli jsou všechny alternativy validní, pro vývoj bude vybrán nástroj React.js pro jeho širokou podporu, rozšíření na trhu a autorovy zkušenosti s tímto nástrojem. Dále bude pro vývoj zvolen jazyk TypeScript, který je nadstavbou jazyka JavaScript a umožňuje využití vlastností staticky typovaných jazyků pro vývoj webových uživatelských rozhraní [42].

6.1.2 Mobilní aplikace

Pro potřeby personalizace nositelných zařízení bude implementována vlastní mobilní aplikace. Proces personalizace bude započat zobrazením QR kódu na displeji zařízení. Uvnitř QR kódu bude jako řetězec uložena MAC adresa daného zařízení. Prostřednictvím mobilní aplikace půjde kód naskenovat a následně bude uživateli prezentován interaktivní formulář, ve kterém bude moci zadat název zařízení, zvolit jeho profil a přiřadit ho konkrétnímu pacientovi.

Pro potřeby tohoto případu užití byla v rámci předmětu KIV/OPSWI vytvořena mobilní aplikace v React Native [41], která tento proces obsluhuje. Mimo jiné je v mobilní aplikaci možné prohlížet seznam zařízení a upravovat jejich nastavení, případně je odpárovat od pacienta. Tato práce se dále nebude implementací mobilní aplikace zabývat.

6.1.3 Server

Z pohledu architektury serveru lze uvažovat tři potenciálně oddělitelné části aplikace:

- obsluha požadavků z webového rozhraní
- obsluha požadavků z mobilní aplikace
- příjem a zpracování dat z nositelných zařízení

Možnosti implementace

Všechny zmíněné části lze tradičním způsobem implementovat v podobě monolitického serveru, který bude zodpovědný za obsluhu veškerých požadavků z klientských aplikací spolu s obsluhou příchozích dat z nositelných zařízení. Toto řešení je méně flexibilní, jelikož nelze provádět nezávislé škálování jednotlivých komponent, nicméně jeho vývoj je výrazně jednodušší a přívětivější.

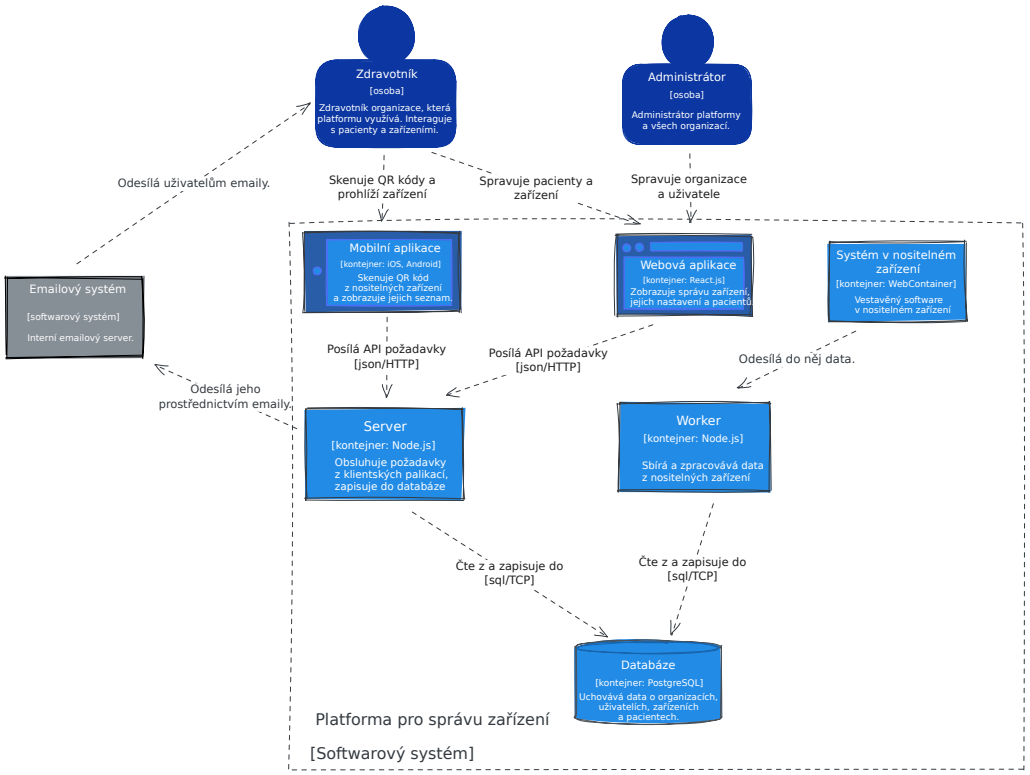
Druhou variantou je využití architektury mikroslužeb. Server by byl rozdělen do samostatných mikroslužeb podle dílčích funkcí, jako je správa organizací, správa pacientů, správa nositelných zařízení apod. Jedná se o velmi flexibilní řešení, kdy je možné jednotlivé mikroslužby nezávisle škálovat a vyvíjet. Z pohledu vývoje jsou ovšem mikroslužby složité na synchronizaci. Je nutné umět spouštět potenciálně mnoho služeb najednou a správně je nakonfigurovat, aby bylo možné provozovat navzájem komunikující mikroslužby i v lokálním prostředí. V krajním případě je nutné dodržovat pořadí spouštění jednotlivých služeb, pokud mezi nimi existují závislosti z pohledu jejich spouštění.

Střední cestou mezi kompletně monolitickým přístupem a mikroslužbami je rozdělení serveru do dvou služeb. Jedna služba by měla na starost příjem a zpracování dat z nositelných zařízení, druhá by obsluhovala požadavky z webových a mobilních klientských aplikací. Případy užití mobilní aplikace a webového portálu se z velké části překrývají, tudíž lze zachovat službu pro jejich obsluhu jednotnou. Díky tomu získáme možnost nezávislého škálování serveru webového portálu i serveru pro zpracování dat z nositelných zařízení. Z pohledu vývoje si navíc zachováme značnou míru přívětivosti, kterou bychom očekávali od monolitického projektu.

Výběr architektury

Vzhledem k tomu, že cílem práce je primárně vytvoření prototypu, předčasná optimalizace ve formě dekompozice do mikroslužeb nedává velký smysl. Naopak by znamenala značné komplikace z pohledu vývoje i nasazení. Využití architektury mikroslužeb by navíc nemělo být první volbou při vývoji nového projektu, pokud k tomu neexistuje množství opodstatněných důvodů. Naopak monolitická architektura je při tvorbě aplikací často validním odrazovým můstkem [49]. Nicméně, brzkou nutnost samostatného škálování služby pro zpracování dat ze spravovaných zařízení lze předvídat již ve fázi návrhu. Proto bude pro implementaci serverové části využita právě poslední zmíněná varianta, tedy dvě oddělené služby pro zpracování dat a pro obsluhu webového portálu a mobilních aplikací. Tato architektura je vizualizována v diagramu 6.1.

Obě služby budou implementovány jako HTTP servery s rozhraním typu REST API [20]. Toho lze docílit využitím mnoha programovacích jazyků a v nich implementovaných nástrojů pro vývoj webových serverů. K těm patří například Go server s využitím možností standardní knihovny, Node.js Express nebo Fastify server, Java server implementovaný nástrojem Spring a Spring Boot nebo server v jazyce Python s využitím knihovny Fast API.



Obrázek 6.1: Jednoduchý C4 diagram kontejnerů systému.

Java Spring server vývojáři předepisuje mnoho praktik, kterými by se měl při implementaci řídit a je vhodný především pro rozsáhlejší projekty s důrazem na jejich stabilitu a robustnost. Ostatní zmíněné varianty vynikají v implementaci služeb či mikroslužeb, u kterých se očekává prototypování či rapidní vývoj a změny. Proto bude pro implementaci zvolen ekosystém Node.js [52] s využitím knihovny Fastify. Jedná se o odlehčenou implementaci Express HTTP serveru, která je ideální právě pro rychlou implementaci a prototypování služeb či mikroslužeb [53]. Obdobně jako v případě uživatelského rozhraní bude server implementován v jazyce TypeScript, díky kterému bude možné využít všech benefitů staticky typovaných jazyků.

6.2 Databáze

Z množství paradigmat databázových systémů se omezíme pouze na relační databáze, jelikož povaha systému jednoznačně směřuje k modelování entit a jejich provázání pomocí relací. Na trhu existuje množství populárních relačních databází, například MySQL, Oracle, Microsoft SQL Server či PostgreSQL. Pro tuto úlohu je v principu vhodná libovolná relační databáze,

nicméně z jmenovaných jsou pouze MySQL a PostgreSQL licencovány jako otevřené kódy. Jelikož podpora PostgreSQL je navíc v ekosystému Node.js serverů hojně rozšířena, zvolíme proto právě tento databázový systém.

Za zmínku ovšem stojí i databázové systémy poskytované jako SaaS, tedy Database as a Service (DaaS). Tyto služby, např. Neon [48], jsou určeny pro systémy s nutností uchování velkého objemu dat, kterým poskytují všechny benefity PaaS - automatické škálování, záloha dat, CLI apod.

6.3 Autentizace a autorizace

Autentizační a autorizační služba musí umožňovat samoobslužnou registraci uživatelů do systému a v rámci autorizace umět rozlišovat různé uživatelské role.

Nejjednodušším řešením z pohledu infrastruktury je implementace autentizační vrstvy jako součásti serveru. Zpřístupněny by byly vyhrazené koncové body umožňující přihlášení, registraci a odhlášení. Zároveň by bylo nutné kompletně implementovat uživatelské rozhraní pro uživatelskou autentizaci a správu uživatelů a organizací administrátorem systému. Jedná se tedy o velmi pracné řešení, které je navíc integrováno do celého serveru. Při potenciálním přechodu na architekturu mikroslužeb by došlo k rozpadu a autentizace by se musela přesunout do samostatné služby.

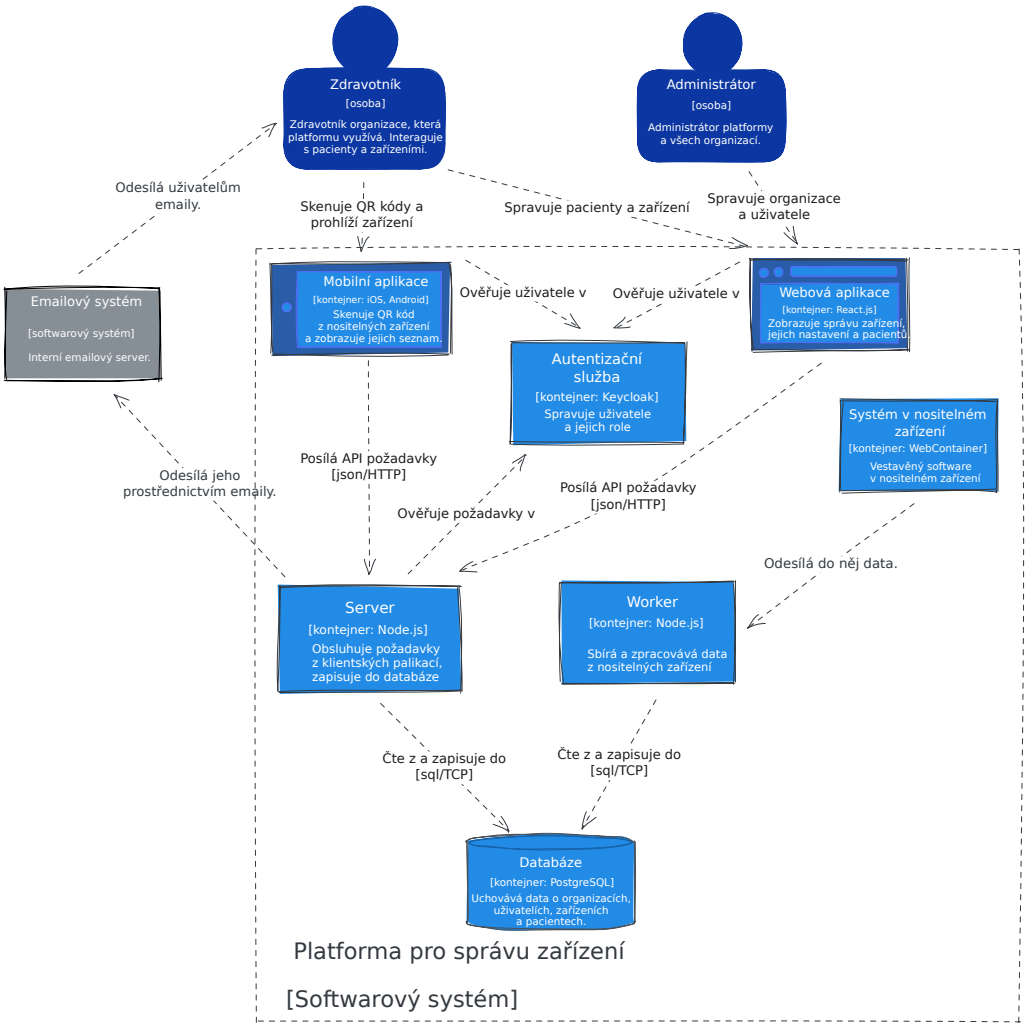
Nabízí se tedy implementace samostatné autentizační a autorizační služby, u které by se klientské aplikace ověřovaly a která by zároveň ověřovala všechny požadavky směřující na server. Vlastní implementace tohoto řešení je ovšem extrémně pracná a tato investice ve většině případů nedává smysl.

Existuje mnoho služeb licencovaných jako otevřený kód, např. Keycloak [67] či Ory Hydra [54], které jsou pro potřeby tohoto projektu naprosto dostačující. Tyto služby poskytují několik možných způsobů autentizace, včetně protokolu OAuth 2.0 [28] a jeho nadstavby OpenID Connect [62]. Často jsou distribuovány prostřednictvím konfigurovatelných kontejnerů, které lze snadno spustit a integrovat do zbytku systému. Klientské aplikace se mohou pomocí předem připravených SDK ke službě připojit a autentizovat. Server pak proti službě provádí autorizaci jednotlivých požadavků.

V neposlední řadě lze využít externích cloudových SaaS aplikací jako je Clerk [12] nebo Auth0 [51], které poskytují autentizaci jako službu. Často využívají protokol OAuth 2.0 a umožňují autentizaci prostřednictvím externích poskytovatelů jako jsou Google, GitHub, Facebook a další. Prakticky se jedná o služby zmíněné výše poskytované jako SaaS. Autentizační SaaS služby jsou ve většině případů dostupné prostřednictvím několika platebních

plánů, včetně bezplatného startovního plánu, sloužícího pro malé aplikace či jednoduché prototypy.

V rámci tohoto projektu je žádoucí, aby byla veškerá uživatelská data dostupná v databázi systému, jelikož je s daty pacientů a zdravotnického personálu nutné dále pracovat. Ta jsou navíc provázána s daty o zařízeních, tudíž je ideální, aby autentizační služba sdílela databázi se serverem. Z tohoto důvodu bude preferována varianta vlastní hostované autentizační služby. Pro tento účel bude využit nástroj Keycloak, jelikož je široce podporován a je licencován jako otevřený kód. Kromě správy identit uživatelů poskytuje možnost správy uživatelských rolí, odesílání emailů prostřednictvím emailového serveru a využití zabudovaného uživatelského rozhraní pro přihlašování a registraci. Způsoby autentizace zahrnují protokoly OAuth 2.0 i OpenID Connect, na kterém bude proces autentizace v tomto projektu postaven. Jeho začlenění do systému je naznačeno v diagramu 6.2.



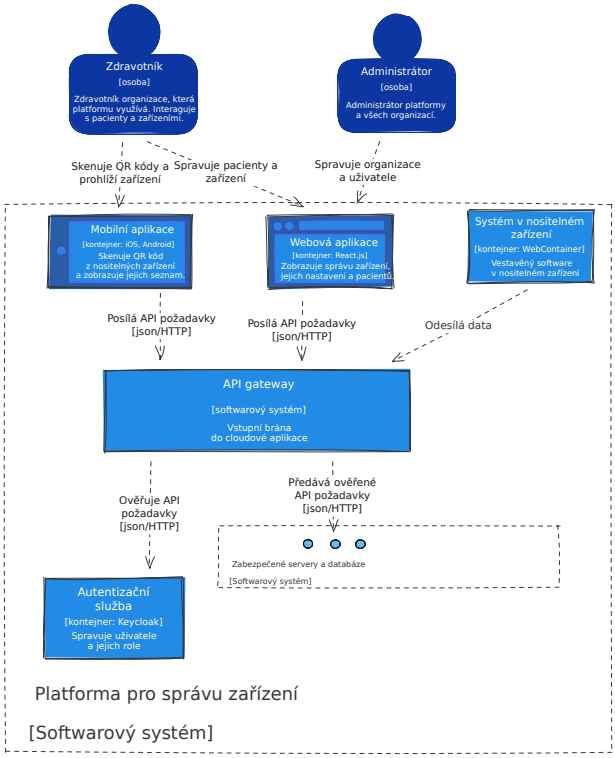
Obrázek 6.2: Diagram kontejnerů systému s využitím centrální autentizační služby.

6.3.1 Dekompozice autentizace a serveru

Ačkoli bude docíleno centralizace správy uživatelů do jediné služby, i nadále bude nutná její přímá integrace se všemi zabezpečenými službami a každá změna tohoto propojení všechny nevyhnutelně ovlivní. V ideálním scénáři jsou všechny zabezpečené služby odstíněny od veřejného přístupu a požadavky jsou automaticky ověřovány proti autentizační službě. Pouze ty, které úspěšně projdou procesem ověření, jsou dále předány ke zpracování zabezpečenou službou, která již dále zdroj požadavku nezkoumá. Toho lze docílit začleněním *API gateway* do infrastruktury systému.

API gateway bude zpracovávat veškeré příchozí požadavky. Podle jejich cílové destinace budou ověřovány u autentizační služby a následně dále pře-

dány ke zpracování serverem. Pokud požadavek nebude disponovat potřebnými informacemi k ověření totožnosti odesílatele, bude již na úrovni *API gateway* zamítnut a bude vrácena chybová odpověď, případně se provede přesměrování na autentizační službu, kde bude uživateli umožněno prokázat svoji identitu. Tím docílíme většího zabezpečení služeb s logikou a jejich odstínění od procesu autentizace. Potenciálně bude možné využít *API gateway* pro potřeby balancování zátěže či omezení příchozích požadavků. Komunikace mezi klientskými aplikacemi a zbytkem infrastruktury je znázorněna v diagramu 6.3.



Obrázek 6.3: Diagram kontejnerů systému po začlenění API gateway.

Na trhu se vyskytuje mnoho *API gateway* služeb, nabízejících různé placené i bezplatné tarify. Každá ovšem umožňuje odlišnou míru konfigurace a poskytuje různé funkce napříč jednotlivými tarify. K těmto službám patří například KrakenD [35], Kong [33] či Ory Oathkeeper [55]. Všechny tyto služby mají několik společných vlastností:

- lze je nasadit jako samostatný Docker kontejner či využívat jako SaaS
- lze je konfigurovat pomocí vlastního konfiguračního souboru specifickým strukturovaným jazykem

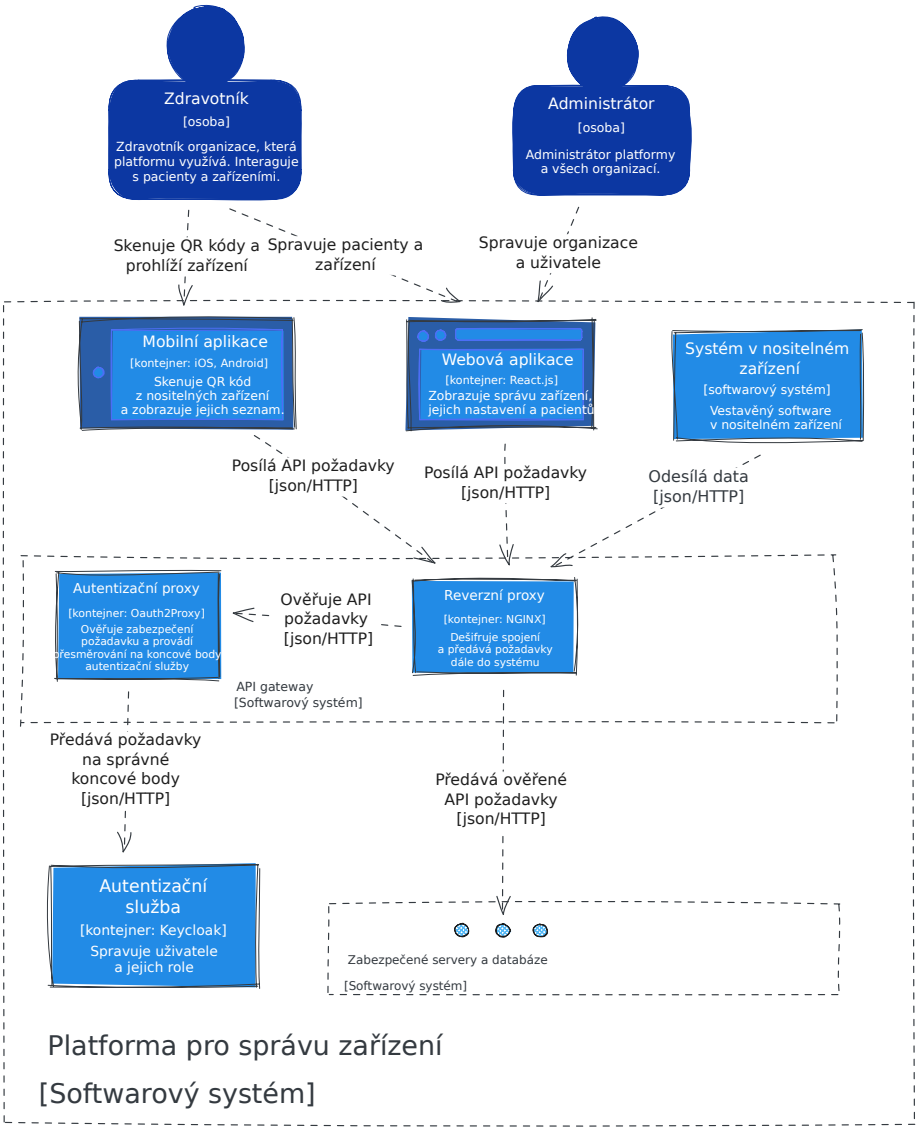
- lze je integrovat s externí autentizační službou, která podporuje protokol OAuth 2.0
- lze definovat nastavení pro dílčí koncové body tak, aby některé zůstaly veřejně přístupné a některé podléhaly ověření přes autentizační službu pomocí přesměrování požadavků

Nicméně, společné mají i to, že v bezplatné verzi žádná z těchto služeb nepodporuje protokol OpenID Connect, který je základním stavebním kamenem autentizace pomocí služby Keycloak. Při selhání ověření požadavku neumožňují automatické přesměrování na autorizační službu za cílem započetí procesu přihlášení. Požadavek je pouze zamítnut a je vrácena odpověď s chybovým kódem.

Je tedy nasnadě vyhledat jiné řešení, které by zprostředkovalo autentizaci u Keycloak instance a zároveň podporovalo protokol OpenID Connect, včetně případného přesměrování pro potřeby přihlášení. K tomu lze využít služby NGINX jako reverzní proxy na vstupu do *API gateway* v kombinaci s Oauth2 Proxy, která funguje jako reverzní proxy pro proces autentizace s Keycloak. Oauth2 Proxy je projekt otevřeného zdroje, distribuovaný jako Docker kontejner, který funguje jako zprostředkovatel autentizace u externích poskytovatelů, jako jsou Keycloak, Google GitHub a další, pomocí protokolů OAuth 2.0 a OpenID Connect [50]. NGINX je hojně rozšířený webový server, který lze detailně konfigurovat a využít pro množství účelů. Jedním z nich je jeho využití jako reverzní proxy, kdy NGINX jako první zpracuje příchozí požadavek, případně dešifruje spojení a na základě konfigurace se rozhodně jak z požadavkem dále naložit. Pomocí této konfigurace lze definovat omezení na jednotlivých koncových bodech, předávat požadavky k autentizaci na autentizační proxy, v případě selhání provést přesměrování na počáteční koncový bod protokolu OpenID Connect a následně předat požadavek dále na zabezpečený server. Začlenění těchto služeb do stávající infrastruktury je vizualizováno v diagramu 6.4. Vizualizace zpracování požadavků systémem je uvedena v sekvenčních diagramech v příloze B.

6.4 Sběr dat

Data naměřená pomocí senzorů na nositelných zařízeních je nutné shromažďovat a později vyhodnocovat. Proto je třeba navrhnout efektivní způsob jejich přenosu z koncových zařízení do systémové databáze. Předpokládáme, že na zařízeních je distribuován vlastní software, umožňující konfiguraci odesílání dat do libovolné destinace prostřednictvím Internetu.

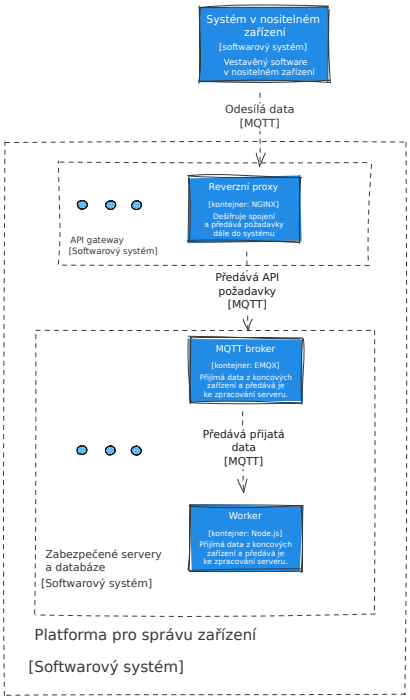


Obrázek 6.4: Zjednodušený diagram kontejnerů systému s použitím NGINX a Oauth2 Proxy jako API gateway.

Koncepčně nejsnazším řešením je vystavení koncového bodu na serveru, kde budou požadavky s naměřenými daty očekávány. Nositelná zařízení pak budou na tento koncový bod data odesílat pomocí HTTP protokolu. Ačkoli je implementace poměrně přímočará, tento návrh má několik zásadních nedostatků z hlediska škálování systému. Objem vysílaných dat z IoT zařízení je obrovský a i při drobném nárůstu počtu spravovaných zařízení může dojít k rapidnímu zpomalení obsluhy požadavků serverem, dokonce až k nedostupnosti systému. V takovém případě by bylo nutné dokázat server vhodným způsobem škálovat, přičemž by nutnost škálování serveru lineárně závisela na počtu spravovaných zařízení. Je tedy nasnadě navrhnout řešení, které je vyhotoveno přímo na míru problematice zpracování dat z IoT zařízení.

Z tohoto důvodu byl vytvořen protokol Message Queuing Telemetry Transport (MQTT), navržený pro vestavěná zařízení v omezených podmínkách s malou šířkou pásma a vysokou odezvou. Pracuje na principu *publisher-subscriber* a snaží se minimalizovat využití sítě i zdrojů zařízení a zároveň zachovat spolehlivost a určitou úroveň zajištění přenosu dat [47]. Součástí architektury MQTT protokolu je *MQTT broker*, který předává zprávy dále na server k dalšímu zpracování.

Pro potřeby této práce bude navržen sběr dat prostřednictvím MQTT brokera, přičemž implementovaný server bude zpracovávat příchozí data. Zajistíme tím dostatečnou volnost mezi koncovými zařízeními a serverem, aby byla co nejvíce eliminována nutnost rapidního škálování při nárůstu počtu spravovaných nositelných zařízení. Servery bude naopak možné jednoduše škálovat v případě potřeby rychlejšího zpracovávání požadavků z fronty. Software v nositelných zařízeních bude odesílat data přímo do MQTT brokera, který je bude dále předávat ke zpracování dedikovanému serveru. Na trhu existuje mnoho variant MQTT brokerů, jejichž rešerše zde nebude uvedena, jelikož se implementace platformy v této práci sběrem dat nezabývá. Pro účely tohoto projektu bude nasazen EMQX broker, jelikož je široce rozšířen a podporován, přičemž je licencován jako otevřený zdroj [16]. Komunikace prostřednictvím MQTT napříč infrastrukturou je znázorněna v diagramu 6.5.



Obrázek 6.5: Diagram kontejnerů systému pro zpracování IoT dat prostřednictvím protokolu MQTT.

6.5 Cíl a způsob nasazení

Nasazení infrastruktury lze provést buď na vlastní instance běžících serverů nebo využít cloudových IaaS či PaaS služeb. Nasazení na vlastní servery ovšem v tomto případě není možné, jelikož nedisponujeme potřebným hardware. Co se týče cloudových služeb, není žádoucí nasazení na infrastrukturu veřejných cloudových poskytovatelů, jelikož se jedná o interní projekt a vyžadujeme plnou kontrolu nad nasazenou infrastrukturou a uloženými daty. ZČU provozuje privátní cloudové služby prostřednictvím nástroje OpenNebula [44] určené pro výukové účely a pro provoz interních univerzitních systémů. Navrhovaná platforma tedy bude nasazena na tuto univerzitní infrastrukturu.

Univerzitní instance OpenNebula poskytuje IaaS služby, omezené na spouštění vlastních virtuálních strojů, které jsou veřejně dostupné z internetu. Pro aktuální účely projektu není nutné zohledňovat automatické škálování shluku kontejnerů pomocí nástrojů jako jsou Kubernetes. V rámci nasazení infrastruktury se tedy omezíme pouze na vytvoření virtuálního stroje s potřebnými závislostmi a spuštění shluku komunikujících kontejnerů.

6.6 Správa infrastruktury

Konzistentní stav infrastruktury bude spravován prostřednictvím IaC nástroje. Na trhu existuje několik používaných nástrojů, například Terraform, OpenTofu [68], Pulumi [59] či Ansible [60].

V této práci bude využita kombinace nástrojů Terraform a Ansible, jelikož jsou na trhu nejrozšířenější a mají nejširší podporu. Pomocí strukturovaného jazyka bude definován nástroj OpenNebula jako poskytovatel cloudových služeb a bude nastaveno připojení na univerzitní instanci. Stejným způsobem bude uvedena konfigurace virtuálního stroje a potřebných náležitostí k uvedení infrastruktury do provozu.

Pomocí nástroje Ansible bude na zajištěném virtuálním stroji nakonfigurován operační systém a nainstalovány závislosti potřebné pro běh nasazovaného software. Ten bude na instanci spuštěn ve formě Docker kontejnerů zpřístupněných pomocí Docker registrů, do kterých bude software zabalen a zveřejněn v průběhu *deployment pipeline*.

Deployment pipeline bude sestávat z fází sestavení, testování a nasazení. Ve fázi sestavení bude ověřena validita integrovaných změn a software bude zabalen do Docker obrazu, který bude zveřejněn do Docker registrů. Testovací fáze podrobí systém implementované sadě testů, verifikujících správnou funkčnost jednotlivých částí systému. Fáze nasazení následně pomocí automatizovaných skriptů a nástroje Terraform nasadí celou infrastrukturu do privátního cloudu.

6.7 Shrnutí

V této kapitole byly představeny možnosti dekompozice navrhovaného systému, ze kterých byla vybrána architektura sestávající z webové Single-Page aplikace a mobilní aplikace, serveru pro jejich obsluhu a samostatného serveru pro zpracování dat z nositelných zařízení. Klientské aplikace budou implementovány v nástrojích React.js, respektive React Native, a serverové služby budou využívat běhové prostředí Node.js s pomocí knihovny Fastify. Všechny aplikace budou dále implementovány v jazyce TypeScript.

Jako perzistentní úložiště byl zvolen databázový systém PostgreSQL pro jeho hojné rozšíření a podporu. Sběr dat bude implementován pomocí protokolu MQTT s využitím MQTT brokera jako fronty zpráv, ze které bude server data číst.

Navržena byla i integrace autentizační služby Keycloak prostřednictvím *API gateway* do infrastruktury systému tak, aby bylo docíleno vhodné dekompozice zabezpečených služeb od procesu autentizace. V neposlední řadě

byl specifikován způsob nasazení platformy na univerzitní infrastrukturu pomocí nástroje OpenNebula, který bude zahrnovat zajištění virtuálního stroje s běhovým prostředím pro spuštění nasazovaných služeb ve formě komunikujících kontejnerů. Celý proces nasazení a správy infrastruktury bude implementován pomocí IaC nástrojů Terraform a Ansible jako součást *deployment pipeline*.

V následující kapitole bude popsán proces konfigurace jednotlivých služeb podpůrné infrastruktury a jejich vzájemného propojení pro možnost spuštění infrastruktury v lokálním vývojovém prostředí i její nasazení do cloudu.

7 Implementace infrastruktury

V této kapitole bude popsána konkrétní konfigurace jednotlivých služeb podpůrné infrastruktury. Nejdříve bude představena jednotná definice služeb, pomocí které je bude možné spustit v lokálním i produkčním prostředí. Následně bude uveden proces konkrétního nastavení klíčových služeb infrastruktury, který zajišťuje jejich vzájemnou komunikaci v souladu s návrhem v kapitole 6.

7.1 Zajištění infrastruktury

Prvním krokem k vytvoření infrastruktury bylo její zajištění v lokálním vývojovém prostředí. K tomu byl využit nástroj Docker Compose, pomocí kterého byly definovány jednotlivé služby, které chceme spustit jako Docker kontejnery uvnitř stejné sítě. Tyto služby bylo třeba nakonfigurovat pomocí proměnných prostředí tak, aby bylo možné veškerá citlivá data předávat bezpečným způsobem. Jednotlivé proměnné prostředí lze specifikovat přímo uvnitř `docker-compose.yml` souboru. Nástroj Docker Compose následně dokáže substituovat uvedené hodnoty pomocí interpolace hodnot z aktuálního běhového prostředí.

Provázání jednotlivých služeb a vystavení jejich koncových bodů pomocí portů lze provést v zásadě libovolně. Nicméně, NGINX kontejner bylo nutné vystavit na port `80`, resp. `443`, aby na něj při nasazení na doménu přicházel veškerý HTTP, reps. HTTPS provoz.

Všechny definované služby jsou ve výchozím stavu spouštěny najednou. V tomto případě bylo nutné specifikovat závislosti mezi službami. Například služba autentizační proxy může být spuštěna až ve chvíli, kdy je Keycloak instance dostupná a přijímá požadavky, čehož bylo docíleno pomocí direktiv `depends_on`. Tato definice je ve zkrácené verzi ukázána v příloze C v úryvku C.1.

Infrastrukturu lze následně zajistit spuštěním příkazu `docker-compose up -d` v adresáři se souborem `docker-compose.yml`. Po úspěšném spuštění by měl být v terminálu viditelný výpis, podobný tomu v úryvku 7.1.

Celá definice lokální infrastruktury pomocí `docker-compose.yml` je v projektu přítomna v adresáři `iac`.

```
iac git:(main)  docker-compose up -d
Starting mailhog        ... done
Starting mqttwebapp     ... done
Starting mqttbroker     ... done
Starting iac_postgres_1 ... done
Starting iac_keycloak_1 ... done
Starting iac_oauth2proxy_1 ... done
Starting iac_npm_1         ... done
```

Úryvek 7.1: Příklad úspěšného spuštění infrstruktury pomocí Docker Compose.

7.2 Konfigurace autentizačních služeb

Nejprve bylo nutné nakonfigurovat instanci Keycloak, aby ji mohly ostatní služby začít využívat. Pomocí proměnných prostředí byly nastaveny přístupové údaje pro administrátora celé platformy a připojení k databázi. Toho bylo docíleno definicí proměnných prostředí v již zmíněném konfigurační souboru Docker Compose.

7.2.1 Nastavení autentizační proxy

Konfigurace autentizační proxy vyžaduje nastavení URL koncového bodu emitenta OpenID Connect tokenu. Ten je vystaven na specifickém zdroji a dynamicky se dle domény nasazení emitenta mění pouze jeho předpona. V průběhu implementace infrastruktury bylo ovšem zjištěno, že Keycloak tento koncový bod poskytuje samostatně pro každou sféru (angl. *realm*), která je v aplikaci vytvořena, a neposkytuje možnost v procesu autentizace zvolit přihlášení do jiné sféry, přičemž každá sféra v instanci Keycloak reprezentuje jednu organizaci. Spuštění instance autentizační proxy navíc selže, pokud není tato URL předána jako proměnná prostředí. Pro připojení autentizační proxy jako OpenID Connect klientské aplikace je nutné vytvořit odpovídající OAuth 2.0 aplikaci v konkrétní sféře Keycloak, ke které chce proxy přistupovat. Toto je zásadní překážka pro implementaci víceuživatelské instance platformy, která byla navržena v předchozích kapitolách. Implementace tedy byla omezena pouze na obsluhu jediné organizace.

7.2.2 Konfigurace OpenID Connect

Jelikož bylo nutné předat do konfigurace autentizační proxy cestu ke koncovému bodu protokolu OpenID Connect pro specifickou Keycloak sféru, bylo nutné nejprve zajistit existenci dané sféry v systému včetně její konfigurace, založení uživatelských rolí a specifikace jejích klientských aplikací. Keycloak umožňuje inicializaci sfér z konfiguračního souboru a zároveň umožňuje export existujících sfér do této podoby. Nejdříve tedy byla vytvořena vzorová sféra pomocí uživatelského rozhraní, kde byly založeny uživatelské role pro zdravotnický personál a administrátora, byla vytvořena OAuth 2.0 klientská aplikace s přístupovými údaji pro autentizační proxy a byl zde vytvořen i uživatelský profil administrátora pro tuto sféru. Celá konfigurace byla následně exportována do formátu konfiguračního souboru, který byl následně namapován do souborového systému Keycloak kontejneru mechanismem Docker volume. Při startu kontejneru byla ověřena existence sféry podle jejího názvu v konfiguračním souboru a veškerá data byla uložena do databáze.

Autentizační proxy byla dále nakonfigurována pomocí proměnných prostředí prostřednictvím Docker Compose definice. Jako poskytovatel OpenID Connect připojení byl nastaven Keycloak, byl nastaven koncový bod, kde se má autentizační proxy dotazovat na informace o ostatních relevantních koncových bodech protokolu, dále množina povolených URL pro přesměrování, povolené domény ad. Všechny citlivé či dynamicky měnící se hodnoty byly interpolovány z vnějšího prostředí. Částečná ukázka konfigurace je zobrazena v příloze C v úryvku C.3.

7.2.3 Kontrola připravenosti Keycloak

Při spuštění infrastruktury ovšem docházelo k chybě. Instance Keycloak označuje kontejner za připravený dříve než stačí nastartovat aplikační server, na čemž je závislý start autentizační proxy. Docker Compose ovšem pomocí direktivy `depends_on` blokuje spuštění služby pouze do doby, než jsou prerekvizitní kontejnery označeny za připravené. Nemá ovšem možnost jak automaticky zjistit připravenost interních aplikací, jelikož se proces jejich spouštění může lišit. Bylo tedy nutné připravit vlastní ověření připravenosti Keycloak kontejneru (angl. *healthcheck*), který by Docker Compose informoval o připravenosti kontejneru až ve chvíli, kdy je připraven i interní aplikační server. Toho lze docílit zasláním HTTP požadavku na koncový bod `/health/ready` aplikačního serveru Keycloak, který začne bezchybně odpovídat až ve chvíli, kdy je server připraven. Zaslání tohoto požadavku bylo provedeno s využitím lokálních TCP nástrojů a v odpovědi byl následně vyhledán kód `200`, označující úspěšnost požadavku. Tato kontrola je spouštěna

s periodou 10 sekund. Výsledná konfigurace je uvedena v příloze C v úryvku C.2.

7.3 Konfigurace NGINX reverzní proxy

Dalším krokem byla konfigurace NGINX reverzní proxy tak, aby byly koncové body systému rozděleny na veřejné, zabezpečené a dostupné pouze administrátorovi a zároveň byly požadavky na spuštění procesu autentizace pomocí OAuth 2.0 protokolu korektně předány dále do systému.

Nejdříve bylo nutné definovat pravidla pro předání požadavků na autentizační proxy k dalšímu zpracování. Je běžnou praxí shlukovat podobné zdroje na stejnou URL cestu. Veškeré požadavky související s autentizací a autorizací uživatele byly sloučeny na cestu s předponou `/auth`. Provoz na této cestě byl pro potenciální budoucí použití přesměrován na instanci Keycloak. Tato část konfigurace je ukázána v úryvku 7.2.

```
# ...
   location /auth {
    rewrite ^/auth(/.*)$ $1 break; # Zahodíme /auth předponu
    # Nastavení potřebných hlaviček
    proxy_set_header Host $host;
    proxy_set_header X-Forwarded-Scheme $scheme;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_set_header X-Forwarded-For $remote_addr;
    proxy_set_header X-Real-IP $remote_addr;
    # Přesměrování na kontejner s vystaveným portem
    proxy_pass http://keycloak:8080;
  }
# ...
```

Úryvek 7.2: Konfigurace NGINX pro přesměrování provozu na Keycloak instanci.

Jelikož implementace autentizační proxy vystavuje své koncové body na cestách s předponou `/oauth2`, byl NGINX nastaven tak, aby byly všechny požadavky na cestě `/auth/oauth2` dále přeposlány na její kontejner. Definice pravidel je uvedena v úryvku 7.3.

Další částí konfigurace je kontrola požadavků na zabezpečené služby, jejich ověření u autentizační služby a následné přeposlání na serverovou službu. Obdobně jako byly sloučeny autentizační požadavky na cestu s předponou `/auth`, požadavky na rozhraní serverových služeb byly vystaveny na cestu s předponou `/api`. Všechny požadavky na tuto URL byly přesměrovány na

```
# ...
  location ~ ^/auth/oauth2/.+$ {
    rewrite ^/auth(/.*)$ $1 break; # Zahodíme /auth předponu
    # nastavení potřebných hlaviček
    # ...
    # tělo požadavku pro autentizační požadavky nechceme
    # přeposílat
    proxy_set_header Content-Length "";
    proxy_pass_request_body off;
    # přesměrování na kontejner s vystaveným portem
    proxy_pass http://oauth2proxy:4180;

  }
# ...
```

Úryvek 7.3: Konfigurace NGINX pro přesměrování provozu na autentizační proxy.

kontejner serveru. Zde bylo ovšem nutné brát v potaz rozdílnost lokální a nasazené infrastruktury. Ačkoli je možné vyvíjet server uvnitř kontejneru, běžnou praxí je spouštění vývojového serveru přímo na stroji vývojáře. Proto bylo žádoucí umět konfiguraci dynamicky upravovat pomocí šablony obsahující proměnné, které bylo možné následně substituovat pomocí nástroje Terraform při nasazení, či lokálně pomocí nástroje `envsubst`. Při nasazení lze provoz směrovat na spuštěný kontejner, při lokálním vývoji je nutné provoz směrovat na IP adresu stroje v lokální síti a port spuštěného procesu. Toto pravidlo je zobrazeno v úryvku 7.4.

```
# ...
  location /api {
    # nastavení potřebných hlaviček
    # ...
    # např. 192.168.0.231:4000 lokálně
    # nebo portal-backend:4000 při nasazení
    proxy_pass http://${BACKEND_SERVICE_URL}$request_uri;
  }
# ...
```

Úryvek 7.4: Konfigurace NGINX pro přesměrování provozu na server.

Požadavky na aplikační rozhraní musí být vždy autentizované. Všechny požadavky tedy bylo nutné nejdříve přesměrovat na autentizační proxy a podle odpovědi rozhodnout co s požadavkem dále, čehož bylo docíleno direktivou `auth_request`. Pokud je požadavek autentizován, bude dále přeposlán

na server. Neautentizovaný požadavek musí být buď zamítnut s chybovým kódem, pokud se nejedná o požadavek prohlížeče, nebo musí být přesměrován na koncový bod autentizační proxy, která automaticky zahájí proces autentizace. Konfigurace tohoto procesu je zobrazena v úryvku 7.5.

```
# ...
  location /api {
    # ...
    # vynucení autentizace požadavku
    auth_request /auth/oauth2/auth;
    error_page 401 = @error401;

    if ($http_accept = "application/json") {
      error_page 401 = @ajax401;
    }
  }

  location @ajax401 {
    add_header Content-Type "application/json";
    return 401 '{"error": "unauthorized"}';
  }

  location @error401 {
    # nejedná se o přesměrování v rámci NGINX
    # zpracování, nýbrž je nutné vrátit odpověď
    # přesměrovat prohlížeč uživatele - nutná adresa
    # nasazené autentizační proxy
    return 302 $forward_scheme://${AUTH_GW_URL}/
        oauth2/start?rd=http://$host$uri;
  }
# ...
```

Úryvek 7.5: Konfigurace NGINX pro autentizaci provozu na server.

Při úspěšné autentizaci požadavku jsou k dispozici údaje o jeho odesilateli, např. uživatelské jméno, email či unikátní identifikátor. Tyto údaje jsou vráceny autentizační službou prostřednictvím hlaviček odpovědi a jsou dále předávány do požadavku na server za účelem vykonání operací aplikační logiky, např. autorizace přístupu ke zdrojům. Toho bylo docíleno použitím direktiv `auth_request_set` a `proxy_set_header`. Jejich užití je zobrazeno v úryvku 7.6. Autentizační proxy ovšem nedokáže využít možností Keycloak v oblasti omezení přístupu ke zdrojům na základě definovaných rolí. Keycloak tyto informace poskytuje přes interní administrátorské API, ale autentizační proxy se spoléhá pouze na rozhraní protokolu OpenID Connect. Z tohoto důvodu byla delegována autorizace přístupu ke zdrojům na samotný server, který bude kontrolu provádět na základě předaných uživatelských informací v hlavičkách požadavku.

```
# ...
  location /api {
    # ...
    # předání dalších informací serveru
    # prostřednictvím hlaviček požadavku
    auth_request_set $user
        $upstream_http_x_auth_request_user;
    auth_request_set $email
        $upstream_http_x_auth_request_email;
    auth_request_set $username
        $upstream_http_x_auth_request_preferred_username;
    auth_request_set $forwarded_groups
        $upstream_http_x_auth_request_groups;

    proxy_set_header X-User $user;
    proxy_set_header X-Email $email;
    proxy_set_header X-Forwarded-User $username;
    proxy_set_header X-Forwarded-Groups $forwarded_groups;
  }
# ...
```

Úryvek 7.6: Konfigurace NGINX pro autentizaci provozu na server.

V neposlední řadě bylo nutné přesměrovat veškerý provoz, který neodpovídá již uvedené konfiguraci, na webovou klientskou aplikaci, jelikož si SPA směrování prostřednictvím URL řeší sama. Pokud by k tomu nedošlo, NGINX by se snažil lokalizovat zdroje ve svém adresáři a vracel by chybový kód, ačkoli by webová aplikace na dané cestě vykreslila existující stránku. Z důvodů zmíněných u přesměrování požadavků na aplikační server bylo nutné vytvořit dynamickou definici pravidel pomocí proměnné. Výsledná

konfigurace je zobrazena v úryvku 7.7.

```
# ...
  location / {
    # Vše ostatní přesměrujeme na webového klienta
    add_header       X-Served-By $host;
    # nastavení potřebných hlaviček
    # ...
    # Zachování originální URI
    proxy_set_header X-Original-URI $request_uri;
     např. 192.168.0.231:5173 lokálně
    # nebo portal-frontend:4000 při nasazení
    proxy_pass http://${FRONTEND_SERVICE_URL}$request_uri;
  }
# ...
```

Úryvek 7.7: Konfigurace NGINX pro směrování ostatnícho provozu na webový klient.

7.4 Shrnutí

V této kapitole byla vytvořena definice podpůrné infrastruktury prostřednictvím Docker Compose. Byla připravena počáteční konfigurace instance Keycloak, včetně administrátorských přístupů, definice uživatelských rolí a založení sféry jakožto organizace. Bylo ovšem zjištěno, že Keycloak v kombinaci s autentizační proxy nepodporují víceuživatelský režim autentizace. Implementace byla tedy omezena na jedinou organizaci.

Byla nastavena autentizační proxy, aby vhodným způsobem komunikovala s instancí Keycloak a prováděla autentizaci příchozích požadavků pomocí protokolu OpenID Connect.

Nakonec byla definována sada pravidel, kterými se řídí instance NGINX v roli reverzní proxy, aby předávala požadavky k ověření prostřednictvím autentizační proxy, ověřené požadavky předávala na zabezpečené servery a neověřené zamítla nebo přesměrovala na autentizační proces protokolu OpenID Connect.

Celá konfigurace byla vytvořena se zohledněním proměnných prostředí, díky čemuž je možné totožnou konfiguraci pomocí substituce proměnných využít nejen pro spuštění v lokálním prostředí, ale i pro kompletní nasazení infrastruktury do produkčního prostředí v cloudu.

V následující kapitole bude do infrastruktury začleněn webový portál, sestávající ze serverové služby a webové aplikace.

8 Implementace webového portálu

V této kapitole bude představena implementace webového portálu včetně serverové části, webového rozhraní a databázového schématu.

Nejdříve budou představeny implementované systémové entity a související relační model. Dále bude vysvětlena dekompozice serverové části do jednotlivých balíků a bude představeno aplikační rozhraní systému včetně jeho specifikace.

V neposlední řadě bude vysvětlena implementace uživatelského rozhraní, rozdělení aplikace do jednotlivých obrazovek, které budou rozlišeny podle úrovně privilegovaného přístupu.

8.1 Systémové entity

Prvním krokem při implementaci serverové části webového portálu bylo navržení vhodného datového modelu. Zásadním faktem, který bylo nutné při návrhu zohlednit, je potřeba navázání na autentizační službu. Přestože proces komunikace autentizační proxy s autentizační službou neumožňuje realizaci víceuživatelského režimu, webový portál bylo nutné pro přechod na tento režim uzpůsobit. Keycloak disponuje vlastním datovým modelem na jehož tabulky bylo nutné vlastní systémové tabulky navázat pomocí relací. Konkrétně se jedná o vazby na tabulku organizace, díky kterým bude možné provádět autorizaci přístupu ke zdrojům jako jsou zařízení či pacienti. Aby bylo možné udržovat tuto vazbu co nejtěsnější pomocí referenční integrity, byly vlastní systémové entity vytvořeny ve stejném databázovém schématu jako entity autentizační služby. Provázání systémových entit je vyobrazeno v diagramu 8.1.

Zařízení a profil zařízení Základní entitou systému je nositelné zařízení, jehož unikátním identifikátorem je jeho MAC adresa. Zařízení lze přiřadit název a profil, charakterizující jeho účel či způsob užití (např. krevní tlak, diabetes, apod.). Tyto profily bude možné do systému dynamicky přidávat i je z něj odebírat, tudíž byly reprezentovány jako samostatná entita. Pro auditní účely bylo vhodné uchovávat časovou značku vytvoření a aktualizace záznamu. Zařízení je nejdříve přidáno do systému administrátorem

jako neaktivní a později je inicializováno zdravotníkem při prvním přiřazení pacientovi. Informace o přechodu mezi těmito stavu byla reprezentována časovou značkou. Dále bylo nutné rozeznat stav zařízení ve dvoukrokovém procesu odpárování od pacienta. Toho bylo docíleno přidáním příznaku typu `bool` jako atributu entity. Každé zařízení i jejich profily vždy patří jediné organizaci, tudíž byly k obou entitám přidány relace typu *1:N* s entitou organizace.

Pacient Zařízení je v aplikaci přiřazováno pacientům, kteří jsou další systémovou entitou. Informace o pacientovi obsahují jeho křestní jméno, příjmení a datum narození. Stejně tak jako u zařízení bylo pro auditní účely vhodné začlenit do entity časové značky vytvoření a poslední úpravy. Pacient je spravován vždy jednou organizací, proto byla k entitě přidána relace typu *1:N* s entitou organizace.

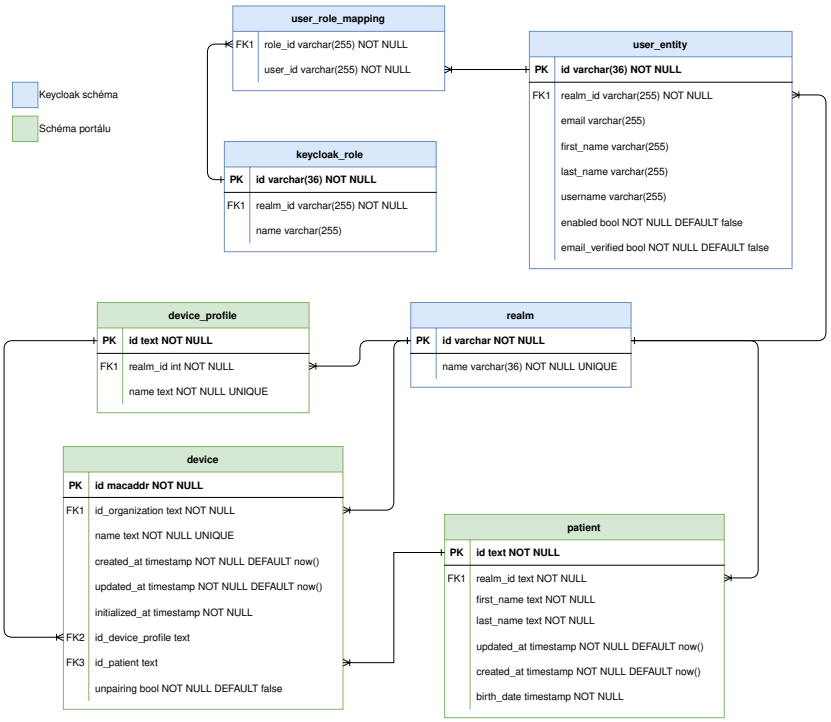
Organizace Organizace je v systému reprezentována jako sféra autentizační služby Keycloak. Keycloak disponuje vlastním datovým modelem, ve kterém jsou sféry uloženy v tabulce `realm`.

Uživatel a uživatelské role Uživatel i uživatelské role spadají do správy autentizační služby. V jejích databázovém schématu jsou reprezentovány tabulkami `user_entity` a `user_role`. Jelikož se jedná o vazbu typu *M:N*, tento vztah je dále rozložen vazební tabulkou `user_role_mapping`.

Každá entita je v aplikačním serveru implementována vlastní třídou v balíku `@glyco-hub/portal-entities`, dostupném v projektu na cestě `packages/portal/entities`. Výsledné databázové schéma systému, obsahující vlastní systémové entity navázané na entity autentizační služby je vizualizováno v relačním diagramu 8.1.

8.2 Aplikační logika

Implementace aplikační logiky serveru byla zapouzdřena do balíku `@glyco-hub/portal-use-cases`. Interně jsou implementace jednotlivých případů užití rozděleny podle entit do samostatných podadresářů. Každý podadresář definuje rozhraní případů užití v modulu `<xy>Api.ts`, které je následně implementováno uvnitř modulu `<xy>.ts`. Rozhraní případů užití specifikuje veškeré závislosti, které jsou k jeho implementaci potřebné, včetně dalších případů užití a rozhraní datové vrstvy, které je pro každý zdroj definováno v modulu `<xy>DataApi.ts` Balík aplikační logiky má přímou závis-



Obrázek 8.1: Diagram relací mezi entitami.

lost na balík systémových entit, které k implementaci využívá. Návratovou hodnotou z každého případu užití je generický objekt `UseCaseResult`, který jako data nese výstupní systémové entity nebo instanci chyby, pokud k nějaké došlo (např. neautorizovaný přístup ke zdroji). Veškeré případy užití jsou vázány na organizace, v rámci kterých jsou vykonávány. Aplikační logika vždy vykonává autorizaci přístupu k danému případu užití a při nedostatečném oprávnění uživatele je přístup zamítnut. Ukázka rozhraní případu užití je uvedena v příloze C v úryvku C.4.

8.3 Datová vrstva

Datová vrstva je implementována v samostatném balíku `@glyco-hub/portal-database`. Jednotlivé exportované funkce implementují datové rozhraní z balíku `@glyco-hub/portal-use-cases` s aplikační logikou. Jelikož jsou jako návratové typy z datového rozhraní specifikovány systémové entity, datová vrstva dále závisí na balíku systémových entit, do kterých transformuje data obdržená z databáze.

Pro komunikaci s databázovým systémem a zajištění transformace příchozích dat na objekty jazyka JavaScript je využit nástroj pro objektově-relační mapování (ORM) *DrizzleORM* [14]. Tyto datové objekty jsou po

obdržení transformovány na systémové entity a vráceny jako návratové hodnoty.

8.4 Aplikační server a jeho rozhraní

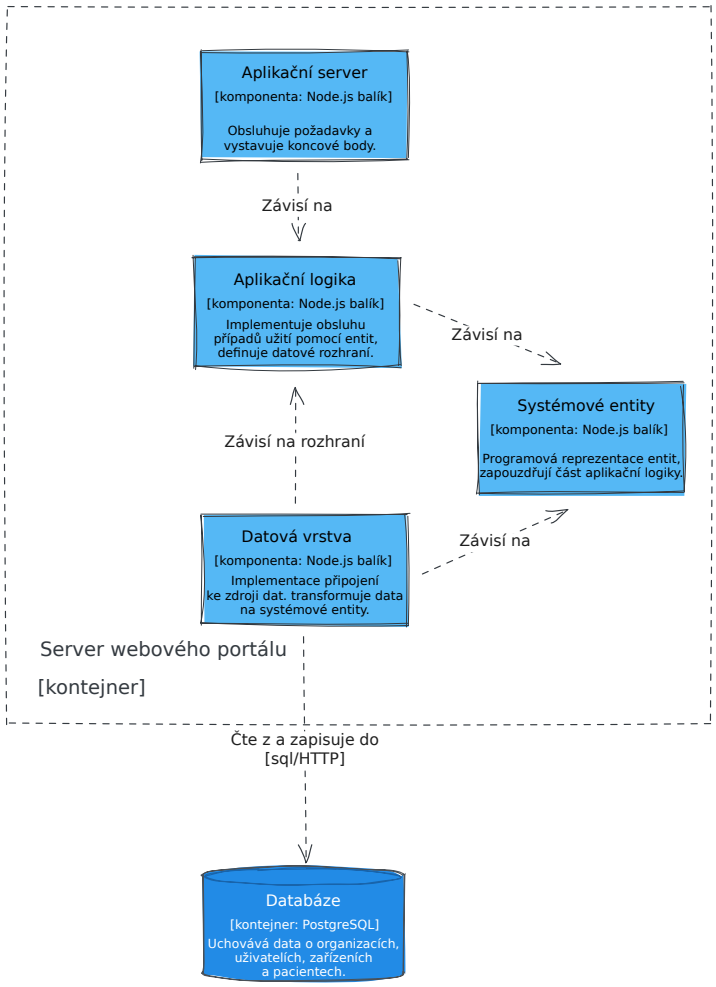
Aplikační server webového portálu je implementován pomocí knihovny Fastify jako jednoduchý HTTP server. Spuštění aplikačního serveru a zajištění všech závislostí pro běh systému je implementováno v aplikačním balíku `@glyco-hub/portal-backend` v projektovém adresáři `apps/portal/backend`.

8.4.1 Realizace aplikačního serveru

Rozhraní serveru je implementováno jako REST API, zdroje v aplikačním rozhraní reprezentují systémové entity a HTTP metody reprezentují povolené operace nad těmito entitami. Koncové body jsou vystaveny pomocí knihovny Fastify, která za pomoci komunitních zásuvných modulů zajišťuje validaci metod a dat příchozích požadavků proti definovanému schématu. Celé aplikační rozhraní je vystaveno na zdroji `/api`, aby odpovídalo konfiguraci reverzní proxy z části 7.3. Jelikož reverzní ani autentizační proxy nejsou schopny zajistit omezení přístupu ke zdrojům na základě rolí, je toho docíleno vystavením administrátorských operací na zdroj `/api/admin`. U všech příchozích požadavků na tomto zdroji je ověřena přítomnost uživatelské role `role:admin` v hlavičce `x-forwarded-groups`, kterou při autentizaci požadavku plní Keycloak a je dále předána přes autentizační a reverzní proxy na server. Koncové body aplikačního rozhraní jsou vystaveny na zdrojích konkrétní organizace (např. `/api/organizations/:id/devices`) za účelem zvýšení bezpečnosti systému a umožnění snadné autorizace přístupu uživatele k daným zdrojům.

Obsluha požadavků je realizována voláním implementací případů užití z balíku `@glyco-hub/portal-use-cases`, kterým jsou jako závislosti předávány funkce datové vrstvy z balíku `@glyco-hub/portal-database`. Vizualizace závislostí systémových balíků je uvedena v diagramu 8.2.

Aplikační server poskytuje pomocí zásuvného modulu knihovny Fastify definici rozhraní prostřednictvím OpenAPI specifikace [66] na zdroji `/api/docs`. Z této specifikace je následně generován klientský kód pro komunikaci s tímto rozhraním v balíku `@glyco-hub/backend-client`.



Obrázek 8.2: Diagram komponent serveru webového portálu.

8.4.2 Sestavení kontejneru

Cílem práce bylo nasazení celé infrastruktury jako množiny komunikujících kontejnerů. Sestavení aplikačního serveru bylo tedy nutné realizovat zabalením výstupů do vlastního Docker kontejneru.

Prvním krokem k vytvoření vlastního aplikačního kontejneru bylo zajistit přítomnost veškerých potřebných zdrojů a zdrojových souborů pro běh aplikačního serveru v kontejneru. Proto bylo nutné rozšířit skript pro sestavení serveru tak, aby kopíroval ostatní potřebné zdroje do složky s balíčkem zdrojových souborů jazyka JavaScript. To jsou konkrétně vygenerované dokumenty se specifikací rozhraní a dokumenty s verzovanými migračními skripty databáze, které po sestavení nejsou automaticky přibaleny do výstupu, jelikož nejsou v kódu přímo využívány. Pro tyto potřeby byl vytvořen sdílený balík `@glyco-hub/builder`, který rozšiřuje proces sestavení o tuto

funkcionalitu a je možné ho v budoucnu využít pro stejné účely v ostatních aplikačních modulech.

Následně byla vytvořena definice obrazu aplikačního kontejneru pomocí souboru `Dockerfile`. Jako výchozí obraz byl zvolen `node:20-10.0-alpine`, který obsahuje potřebnou verzi Node.js. Byl vystaven interní port aplikačního serveru a obsah složky `dist` byl zkopírován do kontejneru. Jako vstupní bod byl zvolen příkaz `node main.js`, který spustí aplikační server z pracovního adresáře.

8.5 Uživatelské rozhraní

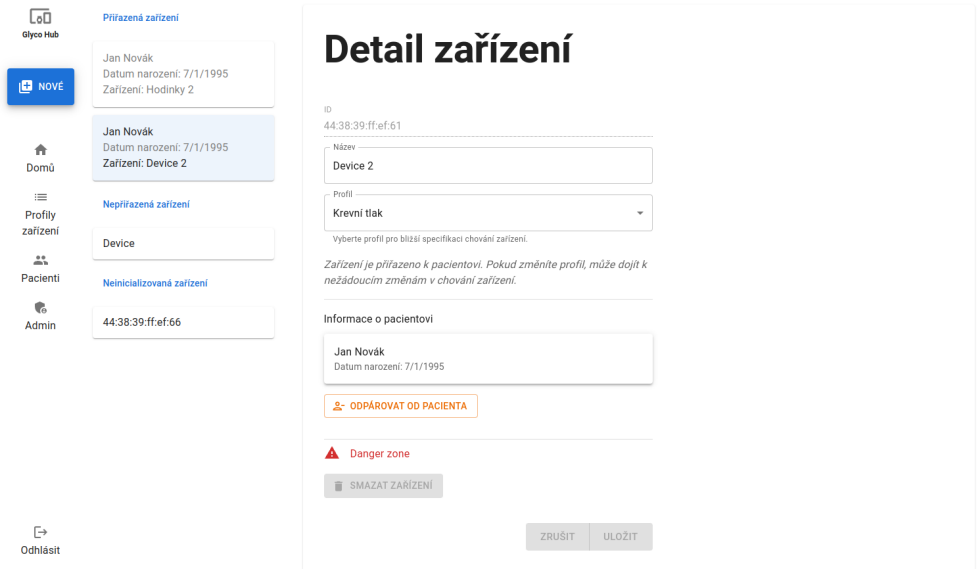
Uživatelské rozhraní webového portálu bylo implementováno jako SPA pomocí nástroje React.js. Návrh uživatelského rozhraní vycházel z návrhového systému Material Design 2 využitím knihovny Material UI. Jednotlivé obrazovky webové aplikace byly rozděleny na veřejné, zabezpečené a administrátorské podle úrovně vyžadovaných oprávnění uživatele. Směrování uvnitř aplikace je realizováno knihovnou React Router, která zajišťuje dynamické změny URL a vykreslování odpovídajících komponent v závislosti na těchto změnách. URL jednotlivých obrazovek byly navrženy dle specifikace REST.

8.5.1 Přihlašovací obrazovka

Obrazovka pro přihlášení uživatele je dostupná na zdroji `/login` a vykresluje jednoduchý formulář s tlačítkem, které přesměruje uživatele na zdroj `/auth/oauth2/start`, kde je následně prostřednictvím reverzní a autentizační proxy zahájen proces autentizace (viz konfigurace reverzní proxy v části 7.3). Po úspěšné autentizaci je uživatel přesměrován na domovskou obrazovku organizace.

Interní obrazovky byly uspořádány do kanonického rozvržení typu Detail položky. Prezentuje uživateli seznam položek přidružených jeho organizaci v levé části obrazovky. Pravá část obrazovky je vyplněna pomocným textem a je určena pro vykreslení detailu položky.

Do krajní levé části byl implementován vertikální navigační panel, obsahující tlačítka sloužící k navigaci mezi obrazovkami a volitelně také akční tlačítko, které provádí významnou akci sdruženou s aktuálně vykreslenou obrazovkou. Ukázka detailu zařízení v uživatelském rozhraní portálu je na obrázku 8.3.



Obrázek 8.3: Detail zařízení v uživatelském rozhraní webového portálu.

8.5.2 Zabezpečené obrazovky

Interní obrazovky webového rozhraní jsou dostupné pouze přihlášeným uživatelům. Při přístupu na zabezpečenou obrazovku je vždy ověřena identita uživatele prostřednictvím zdroje `/auth/oauth2/userinfo`, který přesměruje uživatele na autentizační službu, která jako odpověď vrací informace o aktuálně přihlášeném uživateli v závislosti na poskytnutém přístupovém tokenu v cookies odeslaných spolu s požadavkem. Pokud přístupový token není přítomen, autentizační služba vrací chybový kód a uživatel je přesměrován na přihlašovací obrazovku.

Na zabezpečených obrazovkách dochází k odesílání požadavků na získání zdrojů uživatelovi organizace. Pokud se uživatel pokusí přistoupit na stránku neexistující nebo nepřístupné organizace, je mu přístup zamítnut a následně je přesměrován na domovskou stránku své organizace.

Domovská obrazovka

Domovská obrazovka prezentuje uživateli seznam zařízení přidružených jeho organizaci v levé části obrazovky. Pravá část obrazovky je vyplněna pomocným textem a je určena pro vykreslení detailu položky. Po kliknutí na položku seznamu je uživatel přesměrován na detail tohoto zařízení. Seznam zařízení je rozdělen do tří částí:

- *Přiřazená zařízení* - inicializovaná zařízení, která jsou přiřazena konkrétnímu pacientovi. U položky seznamu je uvedeno jméno a datum

narození pacienta. Pokud je zařízení v procesu odpárování, je tato položka barevně odlišena.

- *Nepřiřazená zařízení* - inicializovaná zařízení, která nejsou přiřazena žádnému pacientovi a je možné provést spárování s pacientem na obrazovce detailu zařízení.
- *Neinicializovaná zařízení* - zařízení, která byla přiřazena organizaci administrátorem, nicméně zatím nebyl proveden proces jejich inicializace prostřednictvím mobilní aplikace.

Administrátorovi systému je na této obrazovce zpřístupněno tlačítko pro přidání nového zařízení. V mobilním zobrazení je prezentován pouze seznam zařízení s totožným chováním jako ve verzi pro stolní počítač. Domovská stránka organizace je přítomna na zdroji `/organizations/:id`.

Detail zařízení

Obrazovka s detailem zařízení uživateli kromě seznamu zařízení popsaného výše prezentuje informace o konkrétním zařízení. Ty jsou vykresleny prostřednictvím interaktivního formuláře, ve kterém je uživatel oprávněn upravovat název a profil zařízení. Formulář v neposlední řadě umožňuje zahájit nebo ukončit proces odpárování zařízení od pacienta. Pokud je zařízení nepřiřazeno, ve formuláři je možné provést přiřazení pacientovi. Administrátorovi systému je na této obrazovce zpřístupněno tlačítko pro přidání nového zařízení a pokud zařízení není přiřazeno pacientovi, zpřístupněno je i tlačítko pro odebrání tohoto zařízení ze systému. Tato obrazovka je přístupna na zdroji `/organizations/:id/devices/:deviceId`.

Seznam pacientů

Obrazovka se seznamem pacientů prezentuje uživateli interaktivní seznam pacientů organizace. Po kliknutí na položku seznamu je uživatel přesměrován na detail vybraného pacienta. Na této stránce je každému uživateli zpřístupněno tlačítko pro přidání nového pacienta k organizaci. V mobilním zobrazení je prezentován pouze seznam pacientů s totožným chováním jako ve verzi pro stolní počítač. Obrazovka se seznamem pacientů je dostupna na zdroji `/organizations/:id/patients`.

Detail pacienta

Obrazovka s detailem pacienta uživateli kromě seznamu pacientů popsaného výše prezentuje osobní informace o konkrétním pacientovi. Ty jsou vykres-

leny prostřednictvím interaktivního formuláře, ve kterém lze upravovat jeho jméno a datum narození. Formulář v neposlední řadě umožňuje odstranit pacienta ze systému. Pokud jsou k pacientovi přiřazena některá zařízení, jsou následně automaticky odpárována. Detail pacienta je přístupný na zdroji `/organizations/:id/patients/:patientId`.

Přidání pacienta

Obrazovka pro přidání nového pacienta organizace obsahuje interaktivní formulář, ve kterém lze vyplnit osobní informace o pacientovi. Po odeslání formuláře je v systému založen nový záznam, který je následně zobrazen v seznamu pacientů na levé straně obrazovky. Přidat nového pacienta je možné na zdroji `/organizations/:id/patients/new`.

Seznam profilů zařízení

Domovská obrazovka prezentuje uživateli seznam profilů zařízení přidružených jeho organizaci v levé části obrazovky. Pravá část obrazovky je vyplněna pomocným textem a je určena pro vykreslení detailu profilu. Po kliknutí na profil v seznamu je uživatel přesměrován na jeho detail. Administrátorovi systému je na této obrazovce zpřístupněno tlačítko pro přidání nového profilu zařízení. V mobilním zobrazení je prezentován pouze seznam profilů s totožným chováním jako ve verzi pro stolní počítač. Tato obrazovka je dostupna na zdroji `/organizations/:id/device-profiles`.

Detail profilu zařízení

Obrazovka detailu profilu zařízení uživateli kromě seznamu profilů prezentuje interaktivní formulář, ve kterém lze upravit jeho název. Administrátorovi je zpřístupněno tlačítko pro odstranění profilu ze systému, přičemž dojde k odstranění všech relací zařízení s tímto profilem. Detail profilu je přístupný na zdroji `/organizations/:id/device-profiles/:deviceProfileId`.

8.5.3 Administrátorské obrazovky

Obrazovky určené pouze administrátorovi obsahují dodatečnou kontrolu autorizace přístupu. Z odpovědi získané od koncového bodu `/auth/oauth2/userinfo` jsou přečteny uživatelské role a je ověřena přítomnost role `role:admin`. Pokud uživatel není pro přístup autorizován, následuje jeho přesměrování na domovskou stránku organizace.

Obrazovka pro přidání zařízení

Obrazovka pro přidání zařízení prezentuje administrátorovi systému interaktivní formulář, ve kterém je možné vyplnit MAC adresu zařízení, které má být přiřazeno organizaci. Odeslání formuláře končí s chybou pokud je formát MAC adresy neplatný nebo již zařízení s touto adresou v systému existuje. Po úspěšném uložení záznamu se zařízení zobrazí jako neinicializované v seznamu zařízení v levé části obrazovky. Tato obrazovka je administrátorovi dostupná na zdroji `/organizations/:id/devices/new`.

Obrazovka pro přidání profilu zařízení

Na obrazovce pro přidání profilu zařízení je zobrazen interaktivní formulář, ve kterém je možné zadat název nového profilu. Odeslání formuláře končí s chybou pokud již profil se stejným názvem pro danou organizaci existuje. Po úspěšném uložení záznamu se profil zařízení zobrazí v seznamu v levé části obrazovky. Tento formulář je administrátorovi dostupný na zdroji `/organizations/:id/device-profiles/new`.

Administrátorský rozcestník

Obrazovka s administrátorským rozcestníkem obsahuje seznam karet, odkazujících na uživatelské rozhraní externích služeb, se kterými platforma komunikuje. Konkrétně jsou přítomny odkazy do administrátorského rozhraní instance Keycloak a do uživatelského rozhraní mailového serveru, skrz který Keycloak odesílá uživatelům emaily. Tato obrazovka je administrátorovi dostupná na zdroji `/organizations/:id/admin`.

8.5.4 Sestavení kontejneru

Výstupem sestavení uživatelského rozhraní je optimalizovaný balíček sestávající z několika JavaScript souborů připojených k jednoduchému `index.html` souboru spolu s ostatními nutnými závislostmi jako jsou definice fontů a kaskádové styly. Vstupním bodem webového rozhraní je pak právě soubor `index.html`.

Tyto soubory je nutné umět servírovat prostřednictvím webového serveru, aby byly dostupné k zobrazení prohlížečem přes Internet. `Dockerfile` obraz webového rozhraní byl tedy odvozen z obrazu `nginx:alpine`, aby

bylo možné využít možností serveru NGINX. Obsah adresáře `/dist` byl následně nakopírován do interního adresáře `/usr/share/nginx/html` uvnitř kontejneru, ve kterém NGINX vyhledává zdroje jako odpověď na příchozí požadavky. Aby bylo zajištěno očekávané chování při přistupování na jednotlivé URL, bylo nutné upravit výchozí konfiguraci NGINX tak, aby vždy vracel soubor `index.html` a směrování nechával v kompetenci samotné aplikace. Tato konfigurace byla následně nakopírována do interního adresáře `/etc/nginx/conf.d/` uvnitř kontejneru, ze kterého NGIX následně načítá vlastní konfigurace. Vlastní nastavení serveru NGINX je zobrazeno v úryvku 8.1.

```
location / {
    root   /usr/share/nginx/html;
    index  index.html index.htm;
    # Potřebujeme směrovat veškerý provoz na soubor index.html,
    # jelikož se jedná o single page aplikaci
    try_files $uri $uri/ /index.html index index.html index.htm;
}
# nastavení vlastní chybové stránky
error_page  404              /error.html;
# přesměrování chyb serveru na vastní chybovou stránku
error_page   500 502 503 504  /error.html;

location = /error.html {
    root   /usr/share/nginx/html;
    # Potřebujeme směrovat veškerý provoz na soubor index.html,
    # jelikož se jedná o single page aplikaci
    try_files $uri $uri/ /index.html index index.html index.htm;
}
```

Úryvek 8.1: Nastavení webového serveru NGINX pro servírování sestavených souborů webového rozhraní.

8.6 Verifikace webového portálu

Verifikace funkcionality webového portálu byla rozdělena do dvou částí. První části byla verifikace aplikační logiky serveru, tou druhou byla verifikace uživatelského rozhraní. K obou částem aplikace bylo nutné přistupovat odlišným způsobem.

Pro verifikaci aplikačního serveru byla implementována sada jednotkových testů. Naproti tomu uživatelské rozhraní bylo ověřováno pomocí manuálních a průzkumných testů.

Další sady testů jako jsou testy integrační či *end-to-end* nebyly v rámci práce implementovány, jelikož byl vyvíjený systém zamýšlen jako prototyp a jsou očekávány jeho další zásadní změny, které by znamenaly kompletní přepracování všech složitějších testů i testů nestabilních částí systému. Pokud by se některá část systému v aktuální podobě dále využívala, je silně doporučeno v rámci rozšíření této práce implementovat její integrační a *end-to-end* testy.

8.6.1 Verifikace aplikační logiky

Pro verifikaci aplikační logiky serveru byla vytvořena sada jednotkových testů, které pokryly veškeré zásadní případy užití, pomocné systémové funkce i aplikační logika zapouzdřená v implementaci systémových entit. Sada jednotkových testů čítala celkem 143 procházejících jednotkových testů, ze kterých bylo 103 jednotkových testů implementace aplikační logiky. Celkové procentuální pokrytí implementace případů užití jednotkovými testy je vyobrazeno v tabulce 8.1.

Případy užití	Výrazy [%]	Větvení [%]	Řádky [%]
Profily zařízení	100	100	100
Zařízení	95,2	94,59	95,2
Organizace	63,15	66,66	63,15
Pacienti	100	100	100
Role	92,3	100	92,3
Uživatelé	100	100	100

Tabulka 8.1: Tabulka pokrytí implementace případů užití jednotkovými testy.

8.6.2 Verifikace uživatelského rozhraní

Verifikace uživatelského rozhraní byla prováděna primárně manuálním testováním. Po nasazení nové funkcionality byla uživatelem prozkoumána celá aplikace podle specifikace předem definovaných případů užití, které jsou vyobrazeny v diagramech v příloze A.

Po standardním manuálním průchodu byla nasazena metoda průzkumného testování, kdy uživatel procházel aplikaci s cílem nalezení chyb nevyskytujících se při očekávaných průchodech případy užití, tedy cílem bylo aplikaci rozbít.

8.7 Shrnutí

V této kapitole byla popsána implementace webového portálu, který zahrnuje serverovou část, webové rozhraní a datový model.

Implementace serverové části začala navržením datového modelu, které bylo vhodně propojeno s autentizační službou Keycloak. Byly popsány systémové entity zařízení, profilů zařízení, pacientů a organizací. Tyto entity byly propojeny v databázi a vytvořily tím relační model systému.

Aplikační logika serveru byla zapouzdřena do balíku `@glyco-hub/portal-use-cases`, kde byly implementovány jednotlivé případy užití. Tyto případy užití byly rozděleny podle entit a implementovány tak, aby specifikovaly veškeré závislosti potřebné k jejich běhu.

Datová vrstva byla implementována v balíku `@glyco-hub/portal-database` s využitím objektově-relačního mapování nástroje DrizzleORM. Datová vrstva implementuje specifikované datové rozhraní balíku aplikační logiky.

Aplikační server byl implementován pomocí knihovny Fastify tak, aby poskytoval REST API pro komunikaci s klienty. Pro obsluhu požadavků byly využity balíky aplikační logiky a datové vrstvy. Rozhraní serveru bylo definováno pomocí OpenAPI specifikace a bylo zpřístupněno na zdroji `/api/docs`.

Uživatelské rozhraní bylo implementováno jako SPA pomocí React.js a Material UI. Bylo rozděleno na veřejné, zabezpečené a administrátorské obrazovky, které umožňují uživatelům různé úrovně přístupu pro práci se systémem. Byly implementovány obrazovky s přehledem zařízení, pacientů a profilů zařízení a zároveň byly vytvořeny obrazovky prezentující detaily jednotlivých entit.

Administrátorské obrazovky zahrnují formuláře pro přidání zařízení a profilů zařízení a rozcestník k externím službám, s nimiž portál komunikuje, jako je například Keycloak.

Pro účel nasazení aplikace do cloudového prostředí byl pro obě části vytvořen vlastní obraz aplikačního kontejneru pomocí `Dockerfile`.

V následující kapitole bude tento systém nasazen do cloudového prostředí s pomocí IaC nástrojů. V rámci procesu nasazení bude vytvořena *deployment pipeline*, která celý proces integrace, ověření a nasazení automatizuje.

9 Nasazení a správa infrastruktury

V této kapitole bude vytvořena IaC konfigurace správy infrastruktury celého systému. Budou vytvořeny dedikované moduly pro samostatné logické celky systému, obsahující automatizované skripty pro instalaci veškerých systémových i aplikačních závislostí. Závěrem budou sestaveny jednotlivé fáze *deployment pipeline*, které by měly zajistit automatické ověření změn a jejich nasazení do cloudového prostředí.

9.1 Správa podpůrné infrastruktury

Pro správu a nasazení infrastruktury systému byl využit IaC nástroj Terraform. V adresáři `iac/modules` byla infrastruktura dekomponována do menších souvisejících celků. Každý modul je dekomponován do několika souborů v závislosti na ověřených postupech nástroje Terraform:

- `main.tf` - hlavní soubor modulu obsahující definice částí infrastruktury
- `variables.tf` - soubor s deklaracemi akceptovaných externích proměnných
- `outputs.tf` - soubor s výstupními proměnnými, které jsou naplněny hodnotami po spuštění modulu
- `terraform.tfvars` - soubor s definicemi hodnot externích proměnných, které Terraform při provedení skriptu načítá

9.1.1 Terraform modul podpůrné infrastruktury

Modul `one-infrastructure` obsahuje definice Terraform zdrojů, potřebných k připojení na univerzitního poskytovatele cloudových služeb, konfiguraci a spuštění virtuálního stroje, a zprovoznění kontejnerů podpůrné infrastruktury.

OpenNebula Terraform poskytovatel

Nejprve byla definována univerzitní instance OpenNebula jako poskytovatel cloudových služeb a byly nastaveny nutné přihlašovací údaje. Následně byla uvedena definice obrazu operačního systému, podle kterého byla později vytvořena instance virtuálního stroje. Tyto definice jsou uvedeny v příloze C v úryvku C.5.

Konfigurace virtuálního stroje

Následně byla vytvořena konfigurace instance virtuálního stroje, který by měl být v cloudovém prostředí zajištěn. Virtuálnímu stroji bylo přiděleno 1 sdílené virtuální CPU, a 2048MB operační paměti. Operační systém byl nastaven pro běh na architektuře `x86_64` a byl mu přidělen disk o velikosti 12GB, ze kterého bude načítán definovaný obraz operačního systému. Síťové nastavení bylo provedeno tak, aby virtuální stroj přijímal spojení na své IP adrese v lokální síti a ovladači síťového rozhraní bylo nastaveno ID virtuální sítě, která má být virtuálnímu stroji přiřazena. V neposlední řadě byl nastaven veřejný SSH klíč pro možnost pozdějšího vzdáleného přístupu do instance. Tato konfigurace je uvedena v příloze C v úryvku C.6.

Inicializace systému

Dále bylo nastaveno spuštění sady inicializačních skriptů virtuálního stroje, které vytvoří nového uživatele se zadaným jménem a přiřazeným dříve nastaveným SSH klíčem, zablokují SSH autentizaci pomocí hesla, nastaví název hostitele a aktualizují systémové balíky. Spolu s inicializačními skripty byl do systémového adresáře přenesen i konfigurační soubor Keycloak sféry. Přenos veškerých souborů do instance virtuálního stroje byl nakonfigurován tak, aby využíval zabezpečené SSH spojení.

Vygenerování konfiguračních souborů

Dalším krokem bylo vygenerování všech nutných konfiguračních souborů z jejich šablon a substituování proměnných za výstupní hodnoty zdrojů Terraform. Pro pozdější využití nástrojem Ansible byl vytvořen inventář spravovaných strojů pomocí šablony s definovanou IP adresou stroje a jménem uživatele. Následně bylo přidáno vygenerování konfigurace reverzní proxy z šablony NGINX konfigurace, která byla definována v části 7.3. Proměnné byly substituovány za IP adresu zajištěného virtuálního stroje a názvy kontejnerů v později nasazené síti Docker kontejnerů. Terraform zdroje, které generují uvedené konfigurace jsou uvedeny v úryvku 9.1.

```
resource "local_file" "ansible-inventory" {
  content = templatefile("${path.module}/../../ansible/inventory.tmpl",
    {
      vm_admin_user = var.vm_admin_user,
      host          = opennebula_virtual_machine.glyco-hub.*.ip,
  })
  filename = "${path.module}/../../ansible/inventory"
}

resource "local_file" "proxy_config" {
  content = templatefile("${path.module}/../../nginx/proxy.conf.template",
    {
      AUTH_GW_URL           =
        "${opennebula_virtual_machine.glyco-hub.ip}/auth",
      BACKEND_SERVICE_URL   = var.backend_service_url,
      FRONTEND_SERVICE_URL = var.frontend_service_url,
      SERVER_NAME           = opennebula_virtual_machine.glyco-hub.ip
  })
  filename = "${path.module}/../../nginx/proxy.conf"
}
```

Úryvek 9.1: Terraform zdroje pro vygenerování Ansible inventáře a NGINX konfigurace.

Přenos konfigurací

Definice pokračovala přenosem vygenerované konfigurace proxy na virtuální stroj zabezpečeným SSH přenosem. Vzhledem k vlastnostem nástroje Terraform, který jednotlivé kroky konfigurace provádí pouze pokud v minulých spuštěních provedeny nebyly nebo došlo ke změnám, bylo nutné zajistit, aby Terraform o změně obsahu konfigurace proxy věděl. Toho bylo docíleno direktivou `triggers`, která zajišťuje exekuci daného kroku pouze ve chvíli, kdy je hodnota spouštěče odlišná od té z předchozího běhu. Hodnota spouštěče byla vygenerována pomocí funkcí `file`, která vygnerovanou konfiguraci načte do paměti, a `sha1`, která z obsahu souboru vytvoří 20B *hash* hodnotu. Tato hodnota se změní pouze ve chvíli, kdy se libovolně změní obsah souboru.

Ansible proměnné

Předposledním krokem byla definice zdroje pro vygenerování souborů z proměnnými pro nástroj Ansible, které budou dále využívány při spouštění infrastruktury na zajištěném virtuálním stroji. Toho bylo opět docíleno pomocí šablonových souborů ve formátu YAML.

```
resource "null_resource" "ansible-provisioner" {
  # Přidáme trigger, který vždy vynutí spuštění zdroje
  triggers = {
    always_run = "${timestamp()}"
  }

  provisioner "local-exec" {
    command     = "ansible-playbook -i inventory glyco-hub-node.yml"
    working_dir = "${path.module}/../../ansible"
    environment = {
      ANSIBLE_HOST_KEY_CHECKING = "False",
      LC_ALL                    = "en_US.UTF-8"
    }
  }
  depends_on = [local_file.ansible-inventory]
}
```

Úryvek 9.2: Terraform zdroj, který spouští Ansible příručky na zadané cestě..

Spuštění Ansible příruček

Posledním krokem Terraform konfigurace bylo spuštění nástroje Ansible, který pomocí interních definic (takzvaných *příruček*), představených v následujících odstavcích, spustí podpůrnou infrastrukturu na cílovém virtuálním stroji. Tento krok nemá žádné kvantifikovatelné výstupní hodnoty, tudíž bylo nutné s pomocí direktivy `triggers` vynutit jeho opakování při každém spuštění nástroje Terraform. Proto bylo jako spouštěč nastaveno volání funkce `timestamp`, která vždy vrací unikátní hodnotu. Terraform zdroj pro spuštění Ansible příruček je uveden v úryvku 9.2.

9.1.2 Ansible příručka podpůrné infrastruktury

Příručka `glyco-hub.node.yml` obsahuje definice úloh, sloužících k instalaci programových závislostí a spuštění kontejnerů podpůrné infrastruktury.

Instalace závislostí

První sada Ansible úloh obsahuje vyčkání na dostupnost vzdáleného serveru, instalaci jazyka Python, správce závislostí Pip a nástroje Docker, včetně Docker Compose a nastavení kontejnerových registrů. Příklad jednoduché příručky je ukázán v úryvku 9.3.

```
- debug:
    msg: "Waiting for the host {{ inventory_hostname }} \
        to be available."

- name: Wait for the host to be available
  ansible.builtin.wait_for_connection:
    delay: 60
    timeout: 300

- debug:
    msg: "The host {{ inventory_hostname }} is running."
```

Úryvek 9.3: Příklad Ansible příručky obsahující úlohy pro vyčkání na připravenost stanice a informativní výpisy.

Spuštění kontejnerů

Ansible podporuje definici úlohy ve formátu Docker Compose, přičemž po spuštění úlohy jsou kontejnery spuštěny podle uvedené definice. Proto bylo možné využít definici lokální infrastruktury z části 7.1 pro jednotné nasazení do cloudového prostředí. Ansible navíc disponuje mechanismem kontroly běhu spuštěných služeb a proto byla definována úloha, která provede kontrolu korektního spuštění všech definovaných kontejnerů. Dané úlohy jsou ukázány v úryvku 9.4.

```
- docker_compose:
    project_name: glycohub
    definition:
      version: "3.8"
    # ...
- assert:
    that:
      - "output.services.oauth2proxy.oauth2proxy.state.running"
      - "output.services.postgres.postgres.state.running"
      - "output.services.keycloak.keycloak.state.running"
      - "output.services.npm.npm.state.running"
      - "output.services.mailhog.mailhog.state.running"
      - "output.services.mqttbroker.mqttbroker.state.running"
```

Úryvek 9.4: Ansible příručka ve formátu Docker Compose včetně úlohy pro kontrolu korektního spuštění služby.

9.2 Správa webového portálu

Aby bylo možné spustit aplikační kontejnery webového portálu na zajištěném virtuálním stroji, bylo nejprve nutné stáhnout jejich obrazy z kontejnerových registrů. Nahrání sestavených obrazů do kontejnerových registrů bude součástí sekce 9.3. Proto byly v této části zvoleny jako kontejnerové registry ty, které jsou součástí univerzitní instance GitLab, využívané pro správu zdrojových souborů tohoto projektu. Zároveň byly předdefinovány očekávané názvy obou obrazů, podle kterých je bude možné ze sdílených registrů stáhnout.

Jelikož se jedná o totožnou logiku stažení pro aplikační server i webové rozhraní, byl vytvořen Terraform modul `ansible-gitlab`, který pomocí specifikovaných proměnných, jako jsou přístupové údaje do kontejnerových registrů, název obrazu a URL registrů, vygeneruje soubor s proměnnými pro nástroj Ansible, který je dále využit jednotlivými Ansible příručkami pro spuštění aplikačních kontejnerů.

9.2.1 Aplikační server

Definice pravidel správy aplikačního serveru začala konfigurací Terraform modulu `one-portal-backend` pro přístup ke kontejnerovým registrům. Dále bylo definováno vygenerování Ansible proměnných pro přístup k databázovému serveru a nakonec bylo uvedeno spuštění nástroje Ansible nad příručkou `glyco-hub-portal-backend.yml`. Příručka pro spuštění aplikačního serveru nejdříve vyčká na připravenost stanice, poté načte všechny potřebné proměnné a v dedikované úloze se autentizuje se vůči kontejnerovým registrům a spustí dynamicky konfigurovaný aplikační kontejner pomocí úlohy s Docker Compose definicí.

9.2.2 Webové rozhraní

Konfigurace správy kontejneru webového rozhraní v modulu `one-portal-frontend` byla obdobná správě aplikačního serveru. Nejdříve byl využit sdílený modul pro přístup ke kontejnerových registrům k vygenerování potřebných Ansible proměnných a následně byl uveden zdroj, který spustí nástroj Ansible nad příručkou `glyco-hub-portal-frontend.yml` pro spuštění aplikačního kontejneru webového rozhraní. Ta opět vyčkala na připravenost stanice, načetla nutné proměnné a spustila dedikované úlohy pro autentizaci vůči kontejnerovým registrům a spuštění aplikačního kontejneru.

9.3 Deployment pipeline

Jedním z cílů projektu bylo vytvoření *deployment pipeline*, která by automatizovaně ověřovala integraci změn do projektu, prováděla automatické sestavení projektu a jeho verifikaci prostřednictvím automatizovaných testů, sestavila obrazy aplikačních kontejnerů webového portálu, nahrála je do kontejnerových registrů a následně automaticky spustila proces nasazení.

Fáze sestavení První fází *deployment pipeline* byla fáze sestavení. V této části bylo provedeno nainstalování potřebných závislostí a následné sestavení projektu. Výsledné artefakty byly uchovány do repozitáře artefaktů nástroje GitLab pro možnost jejich pozdějšího využití.

Verifikační fáze Fáze verifikace byla určena pro spuštění automatizovaných testů nad sestaveným projektem. Tyto úlohy byly nastaveny tak, aby byly automaticky spouštěny pouze při vytvoření požadavku na začlenění změn z vedlejší větvě do větve hlavní a při spuštění *pipeline* nad hlavní větví projektu.

Sestavení a nahrání kontejneru Aby bylo možné sestavit Docker obraz uvnitř GitLab pipeline, která je spuštěna uvnitř Docker kontejneru, bylo nutné využít mechanismu *Docker-in-Docker*. Tento mechanismus znamená, že *pieline* je spuštěna prostřednictvím Docker či Kubernetes exekutoru, který následně využívá obraz kontejneru nástroje Docker. Nicméně, při konfiguraci této fáze *deployment pipeline* bylo zjištěno, že tento mechanismus pro své fungování uvnitř GitLab *pipeline* vyžaduje nastavení privilegovaného režimu exekutoru [25], který v univerzitní instanci GitLab není povolen. Z tohoto důvodu nebylo možné vytvořit automatizovanou fázi sestavení a publikování Docker kontejnerů služeb webového portálu a bylo nutné začlenit do procesu nasazení manuální krok.

Nouzové manuální řešení Aby bylo manuální sestavení a publikování aplikačních kontejnerů co nejpohodlnější, byl vytvořen Bash skript, který celý proces automatizuje a lze ho spustit jediným příkazem v adresáři serverové či klientské části webového portálu. Skript se nejprve autentizuje vůči kontejnerovým registrům, přičemž případně vyzve uživatele k zadání přístupových údajů. Následně je sestaven obraz kontejneru podle definice v `Dockerfile`, který je posléze publikován do kontejnerových registrů. Definice tohoto skriptu je uvedena v úryvku 9.5.

```
#! /bin/bash
# Usage: ./scripts/publish.sh -t <tag>
#
getopts :t: tag

GITLAB_REGISTRY=registry.gitlab-vyuka.kiv.zcu.cz
GITLAB_PROJECT_PATH=iot-device-platform/device-onboarding-portal
GITLAB_REGISTRY_IMAGE=$GITLAB_REGISTRY/$GITLAB_PROJECT_PATH
CONTAINER_NAME=portal/backend

tagname=${OPTARG:=latest}

echo "Building application"

pnpm build

echo "Logging in to $GITLAB_REGISTRY Docker registry"
docker login $GITLAB_REGISTRY

echo "Building $CONTAINER_NAME with tag $tagname"
docker build -t $GITLAB_REGISTRY_IMAGE/$CONTAINER_NAME:$tagname .

echo "Publishing $CONTAINER_NAME to $GITLAB_REGISTRY_IMAGE with tag $tagname"
docker push $GITLAB_REGISTRY_IMAGE/$CONTAINER_NAME:$tagname
```

Úryvek 9.5: Skript pro sestavení a publikování obrazu kontejneru serverové části webového portálu.

9.3.1 Nasazení infrastruktury

Záměrem *deployment pipeline* bylo automatické spuštění nástroje Terraform pro nasazení infrastruktury projektu, jakmile by došlo k sestavení aplikačních kontejnerů webového portálu. Jelikož ovšem neexistuje centrální cloudové prostředí, do kterého by se aplikace měla nasadit (momentálně je projekt spuštěn pouze na studentské instanci virtuálního stroje) a navíc nebylo možné vytvořit automatickou fázi sestavení a publikování obrazů aplikačních kontejnerů portálu, snaha o automatizaci nasazení nakonec nedávala velký smysl.

Nasazení podpůrné infrastruktury i webového portálu bylo tedy zachováno taktéž jako manuální krok. Nasazení jednotlivých Terraform modulů je možné provést spuštěním příkazu `terraform apply -auto-approve` v adresáři cílového modulu. Nejdříve je ovšem nutné nasadit podpůrnou infrastrukturu a až poté jednotlivé aplikační kontejnery webového portálu. Výstupem z nástroje Terraform by měl být výpis podobný tomu v příloze C v úryvku C.8.

Toto řešení není optimální z hlediska praktik DevOps a CI/CD, nicméně vzhledem k vazbě cloudového prostředí na konkrétního vývojáře je pro momentální potřeby projektu dostačující. Cílem další práce by ovšem měla být celková automatizace *deployment pipeline* a zajištění centrálního cloudového prostředí, do kterého by bylo možné aplikaci nasazovat.

9.4 Shrnutí

V této kapitole byla představena konfigurace infrastruktury systému pomocí praktiky Infrastructure as Code. Nejprve byl vytvořen Terraform modul pro správu podpůrné infrastruktury, jehož konfigurace zajišťuje veškeré zdroje nutné pro spuštění virtuálního stroje a nastavení běhového prostředí. Nakonec modul spustí Ansible příručku, která nainstaluje potřebné aplikační závislosti a pomocí Docker Compose definice spustí aplikační kontejnery podpůrné infrastruktury.

Dále byly vytvořeny Terraform moduly pro správu webového rozhraní a aplikačního serveru webového portálu. Oba moduly nakonfigurují potřebné proměnné pro spuštění dedikovaných Ansible příruček, které se po spuštění autentizují u kontejnerových registrů a stáhnou obrazy obou kontejnerů, které následně spustí.

V neposlední řadě byla vytvořena část *deployment pipeline* sestávající z fází sestavení a ověření. Při snaze o vytvoření fáze sestavení obrazů kontejnerů webového portálu bylo zjištěno, že univerzitní instance GitLab neu-

možňuje využití mechanismu *Docker-in-Docker* pro sestavení obrazů uvnitř *pipeline*. Proto byl zbytek procesu nasazení zachován jako manuální. Pro sestavení obrazů byl vytvořen jednoduchý skript a nasazení infrastruktury bylo ponecháno na spuštění nástroje Terraform v příslušných Terraform modulech.

10 Zhodnocení výsledků

I přes začlenění manuálních kroků sestavení aplikačních kontejnerů a nasazení infrastruktury do cloudového prostředí se podařilo vytvořit platformu pro vzdálenou správu nositelných zařízení, která vychází z principů nativních cloudových aplikací.

10.1 Infrastruktura systému

Vytvořená *API gateway* vhodně dekomponuje proces autentizace od zbytku systému a tím odstiňuje ostatní služby od implementačních detailů. Reverzní proxy definuje sadu pravidel, pomocí kterých efektivně směruje příchozí provoz na autentizační proxy či ostatní části systému v závislosti na nutnosti autentizace jednotlivých požadavků. Zavedení reverzní proxy zároveň umožňuje její budoucí využití pro balancování zátěže nebo limitování počtu příchozích požadavků podle definovaných pravidel.

Integrovaná autentizační služba Keycloak splňuje zadané požadavky a poskytuje vlastní uživatelské rozhraní pro autentizaci uživatelů. Zároveň také poskytuje administrátorské rozhraní, ve kterém je možné spravovat jednotlivé organizace, jejich uživatele a uživatelské role. Zároveň poskytuje mechanismus pozvání uživatelů do systému prostřednictvím komunikačních kanálů.

Zvolená autentizační proxy implementuje protokol OpenID Connect pro standardizovanou komunikaci s autentizační službou. Její implementace ovšem neumožňuje realizaci víceuživatelského režimu platformy a proto bylo nutné se v rámci práce omezit na jedinou organizaci. Budoucí rozvoj tohoto systému by měl ideálně cílit na zapracování této funkcionality, například implementací vlastní náhrady za autentizační proxy, nebo další služby, která by zprostředkovala komunikaci mezi autentizační proxy a instancí Keycloak takovým způsobem, aby bylo možné se dynamicky autentizovat u různých organizací.

Veškerá podpůrná infrastruktura byla zajištěna vhodným způsobem, díky kterému je možné ji za stejných podmínek provozovat v lokálním vývojovém prostředí i v prostředí cloudovém. Toho bylo docíleno nástrojem Docker Compose s využitím vhodných konfiguračním parametrů.

Správa infrastruktury platformy byla implementována jako Infrastructure as Code (IaC), díky kterému je možné v repozitáři zdrojových kódů uchovávat jednoznačnou reprezentaci vyžadovaného stavu infrastruktury. Stav

definice lze verzovat, provádět audit jeho změn a využít jeho definici pro vytvoření vícera aplikačních prostředí, určených například pro testování či uživatelskou akceptaci nové funkcionality systému. Bohužel nebylo v rámci projektu možné využít centralizované cloudové prostředí pro nasazení aplikace, pro které by mohl být využit centrální repozitář uchovávající interní stav nasazení infrastruktury. Budoucí práce na tomto projektu by měla zacílit na vytvoření sdíleného centralizovaného prostředí, do kterého by mohlo aplikaci nasazovat více vývojářů. V tu chvíli by bylo zároveň žádoucí využít služby pro centrální uchování stavu nasazení infrastruktury, například Terraform Cloud.

10.2 Webový portál

Implementovaný aplikační server vhodně dekomponuje své dílčí funkcionality do samostatných modulů, které implementují všechny požadované případy užití. Díky této dekompozici je systém snadno udržovatelný, rozšiřitelný a testovatelný. I proto bylo možné jednoduše vytvořit sadu jednotkových testů, které verifikují správnost funkcionality systému vůči specifikaci. Implementace integračních testů, která již byla nad rámec tohoto projektu, je vhodným námětem pro budoucí práce.

Ačkoli nebylo možné realizovat víceuživatelský režim systému, aplikační server byl implementován takovým způsobem, který přechod do tohoto režimu podporuje bezešvě díky zohlednění autorizace přístupu k jednotlivým zdrojům a návrhu i implementaci vhodného datového modelu. Ten je začleněn do databázového systému PostgreSQL, do kterého je začleněn i datový model Keycloak. Tato implementace datového modelu byla zvolena pro zachování jednoduchosti systému a možnosti implementace relací mezi entitami pomocí referenční integrity. V dalším rozšíření systému lze případně provést rozpad datových modelů serveru a Keycloak do samostatných schémat nebo samostatných databázových serverů pro další dekompozici.

Produkční verze aplikačního serveru byla vhodně zabalena do odlehčeného Docker obrazu, který je možné snadno distribuovat prostřednictvím kontejnerových registrů.

Aplikační server shromažďuje logy do standardního výstupu kontejneru, které je momentálně možné číst pouze připojením na stanici virtuálního stroje jelikož začlenění monitorovací infrastruktury a agregátoru logů do platformy bylo již nad rámec tento práce. Jedním z témat budoucích prací by mělo být nasazení služby pro agregaci logů (např. Loki), služby pro interní monitorování běhu aplikace (např. Prometheus) a potenciálně služby

pro vizualizaci dat a čtení aplikačních logů z agregátoru (např. Grafana, Kibana).

Podařilo se implementovat intuitivní uživatelské rozhraní, které je možné využívat prostřednictvím prohlížeče. Rozhraní je vytvořeno za pomoci moderních technologií pro vývoj webových aplikací. Využit byl primárně nástroj React.js a knihovna komponent Material UI, jejichž využití zaručuje přívětivou uživatelskou zkušenost a responzivní návrh, díky kterému je možné aplikaci využívat nejen na osobních počítačích ale i na mobilních zařízeních a tabletech. Klientská část platformy implementuje požadované případy užití a umožňuje přístup na různé obrazovky s různými úrovněmi oprávnění. Na základě autorizace uživatele dále umožňuje přístup k dílčím funkcionalitám systému. Verifikace funkcionality uživatelského rozhraní byla provedena manuálním testováním předem specifikovaných scénářů a průzkumným testováním. Vzhledem k prototypové povaze systému je tento způsob verifikace dostačující, nicméně pokud bude aktuální rozhraní pro budoucí vývoj zachováno, je silně doporučeno implementování sady jednotkových testů uživatelského rozhraní a zároveň sady integračních testů, které by důkladně verifikovaly funkcionalitu aplikace vůči odpovědím serveru. Produkční verze webového rozhraní byla opět vhodně zabalena do Docker obrazu kontejneru s jehož pomocí ho lze dále distribuovat.

10.3 Implementace praktik DevOps

Jedním z cílů práce bylo vytvoření kompletní *deployment pipeline*, která by automatizovaně ověřovala integraci nových změn do základny zdrojového kódu, prováděla jejich verifikaci, jejich následné zabalení do aplikačních kontejnerů a nasazení do cloudového prostředí. Univerzitní instance GitLab bohužel svou konfigurací neumožňovala automatické sestavení Docker kontejneru uvnitř fáze *deployment pipeline* a proto byly vytvořeny pouze fáze sestavení a verifikace. Vytvoření aplikačních kontejnerů webového portálu zůstalo manuálním krokem pomocí skriptu uloženého v repozitáři zdrojových kódů. Nasazení infrastruktury do cloudového prostředí bylo prováděno lokálním spouštěním nástroje Terraform.

Zásadním cílem budoucí práce by mělo být zprovoznění kompletně automatické *deployment pipeline*, která by dokázala automaticky sestavit aplikační kontejnery webového portálu a zároveň automatizovaně nasadit vždy aktuální verzi infrastruktury do cloudového prostředí.

10.4 Integrace mobilní aplikace do platformy

V rámci oborového projektu, plněného v rámci předmětu KIV/OPSWI, byla implementována mobilní aplikace, která komunikovala s jednoduchým serverem, vydávající se za reálný aplikační server této platformy. Hlavní částí mobilní aplikace je proces personalizace nositelného zařízení a jeho inicializace v systému. Tato aplikace byla vytvořena nástrojem React Native a využívá ke komunikaci se serverem stejné jazykové prvky a nástroje jako uživatelské rozhraní webového portálu. V rámci rozšíření tohoto projektu by bylo přínosné integrovat existující mobilní aplikaci s rozhraním implementovaného aplikačního serveru pomocí sdíleného generovaného kódu z definice rozhraní OpenAPI.

11 Závěr

Tato diplomová práce se zaměřila na vytvoření platformy pro vzdálenou správu nositelných zařízení, která vychází z principů nativních cloudových aplikací. V kapitole 2 byl nejdříve představen systém SmartCGMS, jeho možnosti nasazení na nositelná zařízení a problematika sběru a zpracování dat v IoT.

Dále byl v kapitole 3 položen vhodný teoretický základ, který přibližuje fungování cloudových aplikací. Byl vysvětlen pojem cloud computing a byly představeny pojmy Infrastructure as a Service (IaaS), Platform as a Service (PaaS) a Software as a Service (SaaS) včetně jejich benefitů a vzájemných rozdílů. Následně byly zmíněny jednotlivé podpůrné služby, které jsou často pro implementaci cloudových aplikací využívány, a závěrem kapitoly byly představeny charakteristiky nativních cloudových aplikací včetně jejich odlišností a benefitů oproti tradičnímu vývoji software.

Kapitola 4 představila pojmy Continuous Integration (CI), Continuous Delivery (CDE) a Continuous Deployment (CD) a jejich souvislost s vývojem cloudových aplikací. Byl představen princip *deployment pipeline* a metodika DevOps, která všechny zmíněné pojmy využívá pro zefektivnění vývojového procesu software a procesu jeho nasazení a správy. V neposlední řadě byl představen princip Infrastructure as Code (IaC), který je metodikou DevOps často využíván ke správě infrastruktury systému.

Obsahem kapitoly 5 byla specifikace požadavků na implementovaný webový portál a celkovou infrastrukturu systému, v rámci které byly zdokumentovány jednotlivé případy užití pomocí podrobných procesních diagramů. V kapitole 6 byla navržena vhodná infrastruktura pro cloudovou platformu sestávající z webového HTTP serveru a uživatelského rozhraní, dedikované autentizační služby Keycloak, reverzní proxy NGINX, emailového serveru, databázového serveru PostgreSQL a MQTT brokera.

Kapitola 7 obsahovala popis konfigurace dílčích služeb podpůrné infrastruktury v lokálním prostředí nástrojem Docker Compose a konfiguračním souborem služby NGINX, na kterou plynule navázala kapitola 8, popisující implementaci samostatného webového serveru a uživatelského rozhraní. Součástí kapitoly bylo sestavení produkční verze obou částí webového portálu a jejich zabalení do vhodné podoby pro pozdější distribuci.

Kapitola 9 nakonec přiblížila proces správy celé infrastruktury projektu a její nasazení do cloudového prostředí pomocí IaC nástrojů Terraform a Ansible.

Seznam zkratek

API Application Programming Interface. 3, 12, 28, 46, 51, 60

AWS Amazon Web Services. 8

CD Continuous Deployment. 1, 13, 16, 19, 70, 76

CDE Continuous Delivery. 1, 15, 16, 76

CI Continuous Integration. 1, 13, 15, 16, 19, 70, 76

CLI Command Line Interface. 4, 19, 30

CPU Central Processing Unit. 63

DaaS Database as a Service. 30

GCP Google Cloud Platform. 8

HLA High-Level Architecture. 3

HTML Hyper-Text Markup Language. 26

HTTP Hyper-Text Transfer Protocol. 3, 12, 28, 29, 36, 40, 42, 51, 76

HTTPS Hyper-Text Transfer Protocol Secure. 40

IaaS Infrastructure as a Service. 8, 9, 14, 37, 76

IaC Infrastructure as Code. 1, 18–20, 38, 39, 61, 62, 70, 72, 76

IDF IoT Development Framework. 3

IoT Internet of Things. 1, 2, 4, 5, 36, 37, 76, 79

IP Internet Protocol. 44, 63, 112

MAC Media Access Control. 27, 48, 57

MQTT Message Queuing Telemetry Transport. 36–38, 76, 79, 112

NIST National Institute of Standards and Technology. 6

ORM Object-Relational Mapping. 50

OTA Over The Air. 3–5, 79

PaaS Platform as a Service. 8, 9, 14, 30, 37, 76

QR Quick-Response. 27

REST Representational State Transfer. 28, 51, 53, 60

SaaS Software as a Service. 8, 9, 14, 21, 30, 33, 76

SDK Software development kit. 30

SPA Single-Page Application. 26, 46, 53, 60

SRP Single responsibility principle. 10

SSH Secure Shell. 63, 64, 110, 112

TCP Transfer Control Protocol. 42

URL Universal Resource Locator. 42, 43, 46, 53, 58, 67

USB Universal Service Bus. 4

YAML YAML Ain't Markup Language. 64

ZČU Západočeská univerzita v Plzni. 37

Seznam obrázků

Seznam tabulek

Seznam úryvků kódu

Literatura

[1] AL-DEBAGY, O.; MARTINEK, P.: A Comparative Review of Microservices and Monolithic Architectures. In *2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI)*, Listopad 2018, ISSN 2471-9269, s. 149–154, doi:10.1109/CINTI.2018.8928192.

[2] ALJAHDALI, H.; ALBATLI, A.; GARRAGHAN, P.; aj.: Multi-tenancy in Cloud Computing. In *2014 IEEE 8th International Symposium on Service Oriented System Engineering*, Duben 2014, s. 344–351, doi:10.1109/SOSE.2014.50, [cit. 2024-04-21].

[3] ARTAC, M.; BOROVSSAK, T.; DI NITTO, E.; aj.: DevOps: Introducing Infrastructure-as-Code. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, Květen 2017, s. 497–498, doi:10.1109/ICSE-C.2017.162, [cit. 2024-04-21].

[4] BABIUCH, M.; FOLTÝNEK, P.; SMUTNÝ, P.: Using the ESP32 Microcontroller for Data Processing. In *2019 20th International Carpathian Control Conference (ICCC)*, Květen 2019, s. 1–6, doi:10.1109/CarpathianCC.2019.8765944, [cit. 2024-04-21].

[5] BALALAIE, A.; HEYDARNOORI, A.; JAMSHIDI, P.: Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture. *IEEE Software*, ročník 33, č. 3, Květen 2016: s. 42–52, ISSN 1937-4194, doi:10.1109/MS.2016.64, [cit. 2024-04-21].

[6] BASS, L.; WEBWE, I.; ZHU, L.: *DevOps: A Software Architect's Perspective*. Addison-Wesley Professional, Květen 2015, ISBN 9780134049885, [cit. 2024-04-21].

[7] BONIFACE, M.; NASSER, B.; PAPAY, J.; aj.: Platform-as-a-Service Architecture for Real-Time Quality of Service Management in Clouds. In *2010 Fifth International Conference on Internet and Web Applications and Services*, Květen 2010, s. 155–160, doi:10.1109/ICIW.2010.91, [cit. 2024-04-21].

[8] BRIKMAN, Y.: *Terraform: Up and Running*. O'Reilly Media, Inc., Září 2022, ISBN 9781098116743, [cit. 2024-04-21].

[9] BROWN, S.: *Software architecture for developers*. Lean Publishing, 2014, [cit. 2024-04-16].

[10] CHEN, L.: Continuous Delivery: Huge Benefits, but Challenges Too. *IEEE Software*, ročník 32, č. 2, Březen 2015: s. 50–54, ISSN 1937-4194, doi:10.1109/MS.2015.27, [cit. 2024-04-21].

[11] CHEN, R.; LI, S.; LI, Z.: From Monolith to Microservices: A Dataflow-Driven Approach. In *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*, Prosinec 2017, s. 466–475, doi:10.1109/APSEC.2017.53.

[12] CLERK INC.: Clerk: The most comprehensive User Management Platform. 2024, [Online], Dostupné z: https://clerk.com/ [cit. 2024-04-27].

[13] DE CLERQ, J.: Single Sign-On Architectures. In *Infrastructure Security*, editace G. DAVIDA; Y. FrankeFRANKEL; O. REES, Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, ISBN 978-3-540-45831-9, s. 40–58, [cit. 2024-04-21].

[14] DRIZZLE TEAM: Drizzle ORM. 2024, [Online], Dostupné z: https://orm.drizzle.team/ [cit. 2024-05-04].

[15] EBERT, C.; GALLARDO, G.; HERANTES, J.; aj.: DevOps. *IEEE Software*, ročník 33, č. 3, Květen 2016: s. 94–100, ISSN 1937-4194, doi:10.1109/MS.2016.68, [cit. 2024-04-21].

[16] EMQ TECHNOLOGIES INC.: EMQX. 2022, [Online], Dostupné z: https://www.emqx.io/ [cit. 2024-04-27].

[17] ESPRESSIF SYSTEMS (SHANGHAI) Co., Ltd.: ESP-IDF Programming Guide. [Online], Dostupné z: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/index.html [cit. 2024-03-05].

[18] ESPRESSIF SYSTEMS (SHANGHAI) Co., Ltd.: Over The Air Updates (OTA). [Online], Dostupné z: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/ota.html [cit. 2024-03-05].

[19] FARLEY, D.; HUMBLE, J.: *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Professional, 2010, ISBN 9780321670250, [cit. 2024-04-21].

[20] FIELDING, R. T.: *Architectural Styles and the Design of Network-based Software Architectures*. Dizertační práce, University of California, Irvine, 2000, [Online], Dostupné z: https://ics.uci.edu/%7Efielding/pubs/dissertation/rest_arch_style.htm [cit. 2024-05-04].

[21] FITZGERALD, B.; STOL, K.-J.: Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*, ročník 123, 2017: s. 176–189, ISSN 0164-1212, doi:https://doi.org/10.1016/j.jss.2015.06.063, Dostupné z: https://www.sciencedirect.com/science/article/pii/S0164121215001430 [cit. 2024-04-21].

[22] FOWLER, M.: Continuous Integration. Leden 2024, [Online], Dostupné z: https://martinfowler.com/articles/continuousIntegration.html [cit. 2024-02-21].

[23] GANNON, D.; BARGA, R.; SUNDARESAN, N.: Cloud-Native Applications. *IEEE Cloud Computing*, ročník 4, č. 5, Září 2017: s. 16–21, ISSN 2325-6095, doi:10.1109/MCC.2017.4250939, [cit. 2024-05-10].

[24] GARRISON, J.; NOVA, K.: *Cloud Native Infrastructure Patterns for Scalable Infrastructure and Applications in a Dynamic Environment*. O'Reilly Media, Inc., Listopad 2017, ISBN 9781491984307, [cit. 2024-04-21].

[25] GITLAB B.V.: Use Docker to build Docker images. 2024, [Online], Dostupné z: https://docs.gitlab.com/ee/ci/docker/using_docker_build.html#use-docker-in-docker [cit. 2024-05-08].

[26] GOOGLE LLC: Google Play Developer Program Policy. 2024, [Online], Dostupné z: https://support.google.com/googleplay/android-developer/answer/14144369?visit_id=638414383765054452-84557828&rd=2 [cit. 2024-03-05].

[27] GOOGLE LLC: Svelte: Deliver web apps with confidence. 2024, [Online], Dostupné z: https://angular.io/ [cit. 2024-04-21].

[28] HARDT, D.: The OAuth 2.0 Authorization Framework. RFC 6749, říjen 2012, doi:10.17487/RFC6749, [Online], Dostupné z: https://www.rfc-editor.org/info/rfc6749 [cit 2024-04-26].

[29] HOFFMAN, K.: *Beyond the Twelve-Factor App*. O'Reilly Media, Inc, 2016, ISBN 9781491944035, [cit. 2024-04-21].

[30] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION [ISO]: ISO/IEC 18004:2015 - QR Code bar code symbology specification. *Automatic identification and data capture techniques*, 2015, Dostupné z: https://www.iso.org/standard/62021.html [cit. 2024-04-21].

[31] JOHANN, S.: Kief Morris on Infrastructure as Code. *IEEE Software*, ročník 34, č. 1, Leden 2017: s. 117–120, ISSN 1937-4194, doi:10.1109/MS.2017.13, [cit. 2024-04-21].

[32] KENT, K. A.; SOUPPAYA, M.: Guide to Computer Security Log Management. *Recommendations of the National Institute of Standards and Technology*, 2006, [cit. 2024-04-21].

[33] KONG INC.: Kong: The world's most adopted open source API gateway. 2024, [Online], Dostupné z: https://konghq.com/products/kong-gateway [cit. 2024-04-27].

[34] KOUTNÝ, T.; ÚBL, M.: Parallel software architecture for the next generation of glucose monitoring. *Procedia Computer Science*, ročník 141, Leden 2018: s. 279–286, ISSN 1877-0509, doi:10.1016/J.PROCS.2018.10.197, [cit. 2024-04-21].

[35] KRAKEND S.L.: KrakenD: The API Gateway pattern at its full extent. 2024, [Online], Dostupné z: https://www.krakend.io/ [cit. 2024-04-27].

[36] KRIEF, M.: *Learning DevOps*. Packt Pulblishing, Březen 2022, ISBN 9781801818964, [cit. 2024-04-21].

[37] MARTY, R.: Cloud application logging for forensics. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, SAC '11, New York, NY, USA: Association for Computing Machinery, 2011, ISBN 9781450301138, str. 178–184, doi:10.1145/1982185.1982226, Dostupné z: https://doi.org/10.1145/1982185.1982226 [cit. 2024-04-21].

[38] MDN CONTRIBUTORS: JavaScript. 2024, [Online] Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript [cit. 2024-05-11].

[39] MELL, P. M.; GRANCE, T.: The NIST definition of cloud computing. 2011, doi:10.6028/NIST.SP.800-145, Dostupné z:

https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf [cit. 2024-04-21].

[40] META OPEN SOURCE: React: The library for web and native user interfaces. 2024, [Online], Dostupné z: https://react.dev/ [cit. 2024-04-21].

[41] META PLATFORMS INC.: React Native: Learn once, write anywhere. 2024, [Online], Dostupné z: https://reactnative.dev/ [cit. 2024-04-25].

[42] MICROSOFT: TypeScript: JavaScript with syntax for types. 2024, [Online] Dostupné z: https://www.typescriptlang.org/ [cit. 2024-05-11].

[43] MIKOWSKI, M.; POWELL, J.: *Single page web applications: JavaScript end-to-end*. Simon and Schuster, 2013, ISBN 978-1617290756, [cit. 2024-04-21].

[44] MILOJIČIĆ, D.; LLORENTE, I. M.; MONTERO, R. S.: OpenNebula: A Cloud Management Tool. *IEEE Internet Computing*, ročník 15, č. 2, Březen 2011: s. 11–14, ISSN 1941-0131, doi:10.1109/MIC.2011.44, [cit 2024-04-27].

[45] MONTESI, F.; WEBER, J.: Circuit Breakers, Discovery, and API Gateways in Microservices. *arXiv preprint arXiv:1609.05830*, Září 2016, [cit. 2024-04-21].

[46] MORRIS, K.: *Infrastructure as a Code*. O'Reilly Media, Inc., Prosinec 2020, ISBN 9781098114671, [cit. 2024-04-21].

[47] MQTT.ORG: MQTT Version 5.0. 2019, [Online], Dostupné z: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html [cit 2024-04-26].

[48] NEON INC.: Neon: Serverless Postgres. 2024, [Online], Dostupné z: https://neon.tech/ [cit. 2024-04-25].

[49] NEWMAN, S.: *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media, červenec 2021, ISBN 978-1492034025, [cit. 2024-04-21].

[50] OAUTH2 PROXY CONTRIBUTORS: Oauth2 Proxy. 2024, [Online] Dostupné z: https://oauth2-proxy.github.io/oauth2-proxy/ [cit. 2024-05-10].

[51] OKTA INC.: Auth0: Secure access for everyone but not just anyone. 2024, [Online], Dostupné z: https://auth0.com/ [cit. 2024-04-27].

[52] OPEN JS FOUNDATION: Node.js: Run JavaScript Everywhere. 2024, [Online] Dostupné z: https://nodejs.org/ [cit. 2024-05-11].

[53] OPEN JS FOUNDATION; THE FASTIFY TEAM: Fastify: Fast and low overhead web framework, for Node.js. 2024, [Online] Dostupné z: https://fastify.dev/ [cit. 2024-05-11].

[54] ORY CORP: Ory Hydra. 2024, [Online], Dostupné z: https://www.ory.sh/docs/ecosystem/projects#ory-hydra [cit. 2024-04-27].

[55] ORY CORP: Ory Oathkeeper. 2024, [Online], Dostupné z: https://www.ory.sh/docs/ecosystem/projects#ory-oathkeeper [cit. 2024-04-27].

[56] OTTA, M.: Towards a health software supporting platform for wearable devices. *Procedia Computer Science*, ročník 210, Leden 2022: s. 112–115, ISSN 1877-0509, doi:10.1016/J.PROCS.2022.10.126, [cit. 2024-04-21].

[57] PLAGERAS, A. P.; PSANNIS, K. E.; STERGIOU, C.; aj.: Efficient IoT-based sensor BIG Data collection–processing and analysis in smart buildings. *Future Generation Computer Systems*, ročník 82, 2018: s. 349–357, ISSN 0167-739X, doi:https://doi.org/10.1016/j.future.2017.09.082, Dostupné z: https://www.sciencedirect.com/science/article/pii/S0167739X17314127 [cit. 2024-04-21].

[58] PLATFORMIO: Your Gateway to Embedded Software Development Excellence. 2024, [Online], Dostupné z: https://docs.platformio.org/en/latest [cit. 2024-03-05].

[59] PULUMI INC.: Pulumi: Infrastructure as Code in Any Programming Language. 2024, [Online] Dostupné z: https://www.pulumi.com/ [cit. 2024-05-11].

[60] RED HAT INC.: Ansible community documentation. 2024, [Online] Dostupné z: https://docs.ansible.com/ [cit. 2024-05-11].

[61] ROSE, K.; Eldridge, S.; Chapin, L.: The internet of things: An overview. *The internet society (ISOC)*, ročník 80, 2015: s. 1–50, [cit. 2024-04-21].

[62] SAKIMURA, N.; BRADLEY, J.; JONES, M.; aj.: OpenID Connect Core 1.0. Prosinec 2023, [Online], Dostupné z: https://openid.net/specs/openid-connect-core-1_0.html [cit 2024-04-26].

[63] SHAHIN, M.; ALI BABAR, M.; ZHU, L.: Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. *IEEE Access*, ročník 5, 2017: s. 3909–3943, ISSN 2169-3536, doi:10.1109/ACCESS.2017.2685629, [cit. 2024-04-21].

[64] SVELTE CONTRIBUTORS: Svelte: Cybernetically enhanced web apps. 2024, [Online], Dostupné z: https://svelte.dev/ [cit. 2024-04-21].

[65] THE KUBERNETES AUTHORS, THE LINUX FOUNDATION: Kubernetes. 2024, [Online], Dostupné z: https://kubernetes.io/ [cit. 2024-03-05].

[66] THE LINUX FOUNDATION: OpenAPI Specification. 2021, [Online], Dostupné z: https://spec.openapis.org/oas/latest.html [cit. 2024-05-04].

[67] THE LINUX FOUNDATION: Keycloak: Open Source Identity and Access Management. 2023, [Online], Dostupné z: https://www.keycloak.org/ [cit. 2024-04-27].

[68] THE LINUX FOUNDATION: OpenTofu: The open source infrastructure as code tool. 2024, [Online] Dostupné z: https://opentofu.org/ [cit. 2024-05-11].

[69] THÖNES, J.: Microservices. *IEEE Software*, ročník 32, č. 1, Leden 2015: s. 116–116, ISSN 1937-4194, doi:10.1109/MS.2015.11, [cit. 2024-04-21].

[70] TURNBULL, J.: *Monitoring with Prometheus*. Turnbull Press, 2018, [cit. 2024-04-21].

[71] VINOSKI, S.: Advanced Message Queuing Protocol. *IEEE Internet Computing*, ročník 10, č. 6, 2006: s. 87–89, doi:10.1109/MIC.2006.116, [cit. 2024-04-21].

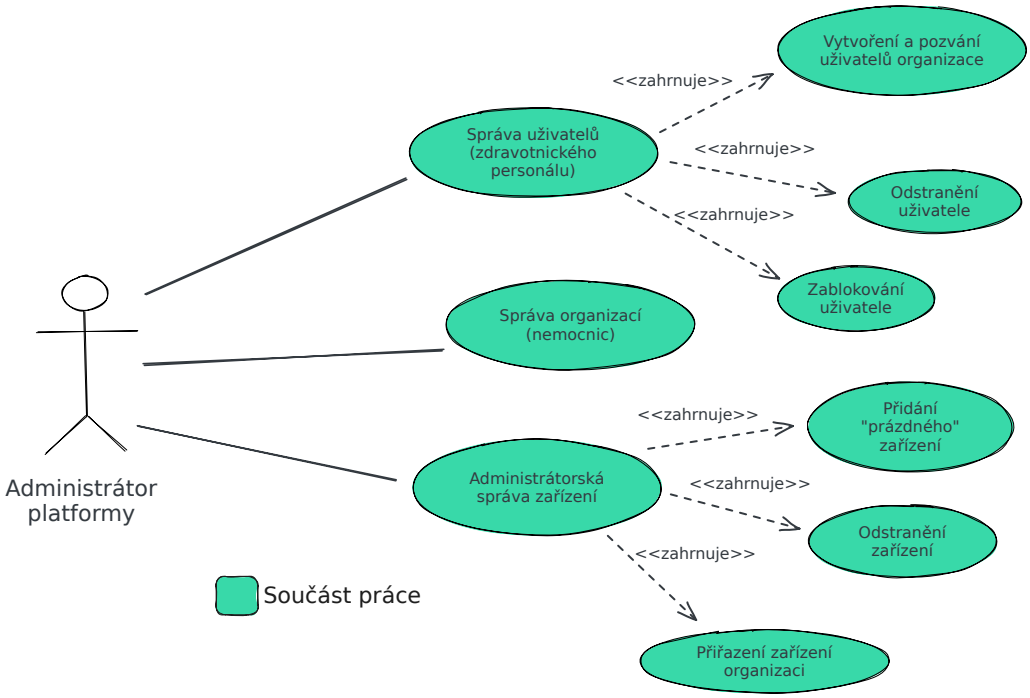
[72] WATERS, B.: Software as a service: A look at the customer benefits. *Journal of Digital Asset Management*, ročník 1, č. 1, 2005: s. 32–39, ISSN 1743-6559, doi:10.1057/palgrave.dam.3640007, [cit. 2024-04-21].

[73] WEBER, I.; NEPAL, S.; ZHU, L.: Developing Dependable and Secure Cloud Applications. *IEEE Internet Computing*, ročník 20, č. 3, Květen 2016: s. 74–79, ISSN 1941-0131, doi:10.1109/MIC.2016.67, [cit. 2024-04-21].

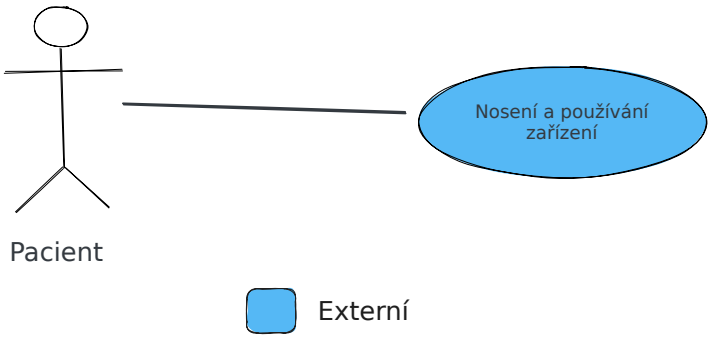
[74] WIGGINS, A.: The Twelve-Factor App. 2017, [Online], Dostupné z: https://12factor.net/ [cit. 2024-03-05].

[75] WORLD WIDE WEB CONSORTIUM: HTML. Duben 2024, [Online], Dostupné z: https://html.spec.whatwg.org/multipage/ [cit. 2024-04-21].

A Případy užití



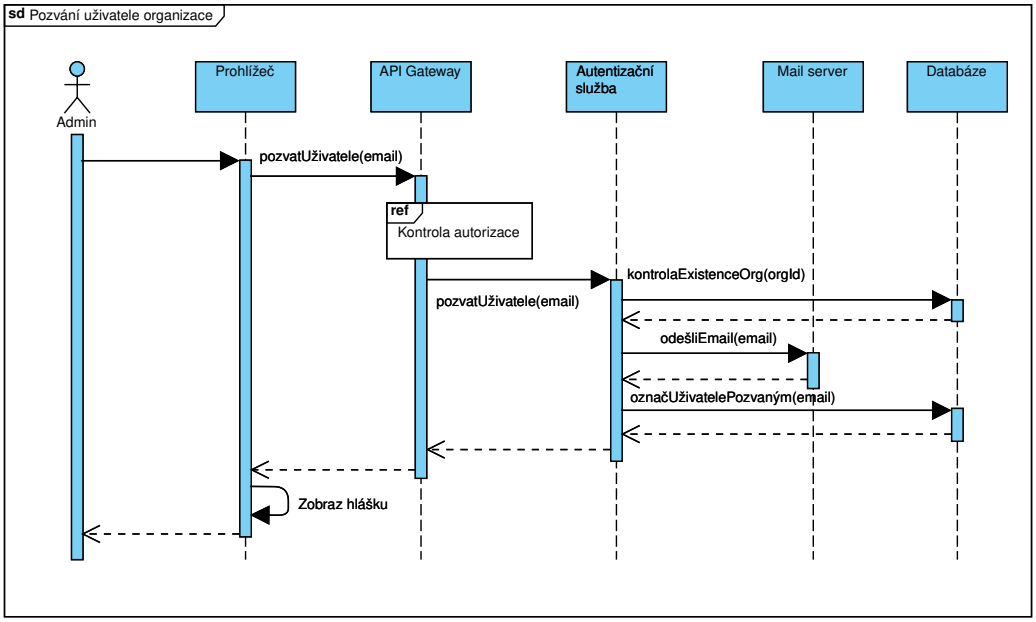
Obrázek A.1: Administrátorské případy užití.



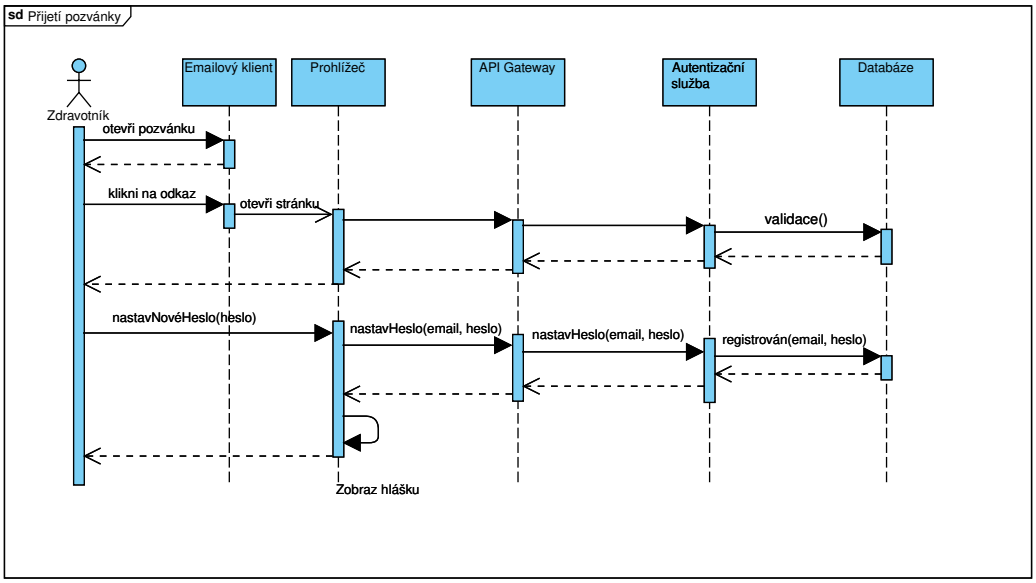
Obrázek A.2: Případy užití pacientů.

Obrázek A.3: Případy užití zdravotnického personálu.

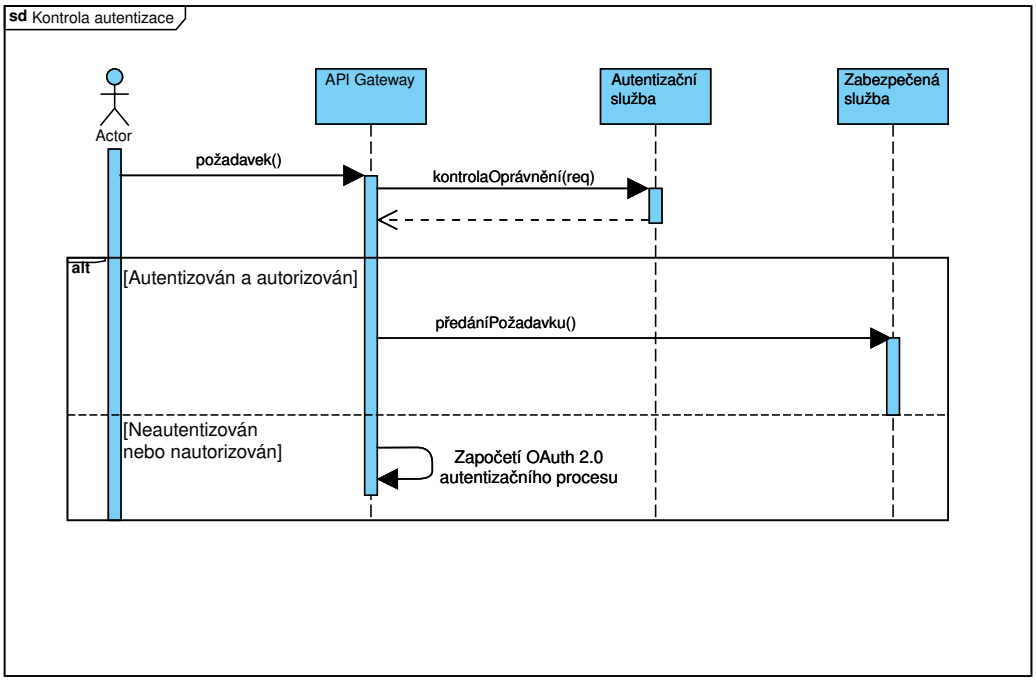
B Sekvenční diagramy



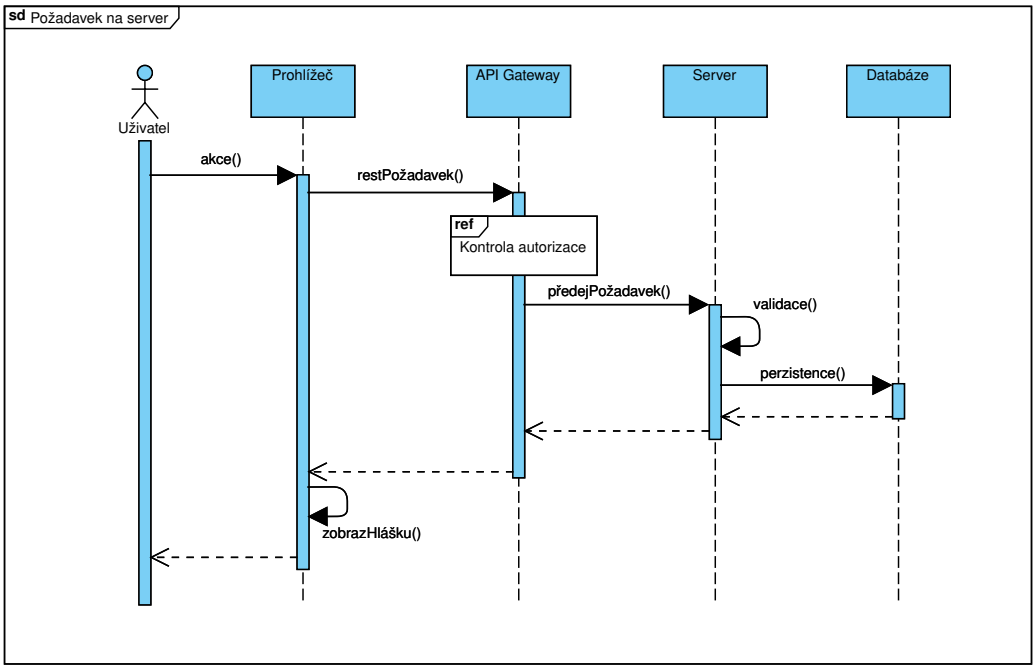
Obrázek B.1: Sekvenční diagram pozvání uživatele do systému.



Obrázek B.2: Sekvenční diagram akceptace pozvánky do systému.



Obrázek B.3: Sekvenční diagram kontroly autentizace a autorizace.



Obrázek B.4: Sekvenční diagram arbitrárního požadavku na server.

C Úryvky kódu

```yaml
version: "3.8"
    services:
      oauth2proxy:
        image: quay.io/oauth2-proxy/oauth2-proxy:latest
          # ...
        depends_on:
          keycloak:
            # keycloak musí být spuštěn a přijímat spojení
            condition: service_healthy
      keycloak:
        image: quay.io/keycloak/keycloak:23.0.6
            # ...
      # nginx reverse proxy s webovým UI
      npm:
        # ...
      postgres:
        image: postgres:latest
        # ...
      mailhog:
        image: mailhog/mailhog:latest
        # ...
      mqttbroker:
        image: emqx/emqx:latest
        # ...
      mqttwebapp:
        image: emqx/mqttx-web:latest
        # ...
```

Úryvek C.1: Úryvek z definice podpůrných služeb infrastruktury pomocí Docker Compose.

```yaml
services:
  oauth2proxy:
    image: quay.io/oauth2-proxy/oauth2-proxy:latest
      # ...
    depends_on:
      keycloak:
        # keycloak musí být spuštěn a přijímat spojení
        condition: service_healthy
  keycloak:
    image: quay.io/keycloak/keycloak:23.0.6
        # ...
    healthcheck:
      test:
          [
              "CMD-SHELL",
              "exec 3<>/dev/tcp/127.0.0.1/8080;
              echo -e \"GET /health/ready HTTP/1.1\r\n
              host: http://localhost\r\nConnection: close
              \r\n\r\n\">&3;grep \"HTTP/1.1 200 OK\" <&3",
            ]
      interval: 10s
      timeout: 5s
      retries: 5
  # ...
```

Úryvek C.2: Vlastní ověření stavu Keycloak kontejneru v Docker Compose.

```yaml
services:
  oauth2proxy:
    image: quay.io/oauth2-proxy/oauth2-proxy:latest
    environment:
      OAUTH2_PROXY_HTTP_ADDRESS: "0.0.0.0:4180"
      OAUTH2_PROXY_PROVIDER: keycloak-oidc
      OAUTH2_PROXY_CLIENT_ID: ${OAUTH2_PROXY_CLIENT_ID}
      OAUTH2_PROXY_CLIENT_SECRET:
        ${OAUTH2_PROXY_CLIENT_SECRET}
      OAUTH2_PROXY_ALLOWED_AUDIENCE: ${OAUTH2_PROXY_CLIENT_ID}
      OAUTH2_PROXY_OIDC_ISSUER_URL:
        http://${AUTH_SERVER_HOSTNAME}/realms/${REALM_NAME}
      OAUTH2_PROXY_REDIRECT_URL:
        ${AUTH_GW_URL}/oauth2/callback
      OAUTH2_PROXY_SCOPE: "openid email roles"
      # ...
    depends_on:
      keycloak:
        condition: service_healthy

  keycloak:
    image: quay.io/keycloak/keycloak:23.0.6
    environment:
      - KEYCLOAK_ADMIN=${KEYCLOAK_ADMIN_USERNAME}
      - KEYCLOAK_ADMIN_PASSWORD=${KEYCLOAK_ADMIN_PASSWORD}
      - KC_DB=postgres
      - KC_DB_URL=jdbc:postgresql://postgres:5432/keycloak
      - KC_DB_USER=${KEYCLOAK_DB_USER}
      - KC_DB_PASSWORD=${KEYCLOAK_DB_PASSWORD}
      - REALM_NAME=${REALM_NAME}
        # ...
    # ...
    healthcheck:
        # ...
    volumes:
      - ./keycloak/realm-config.json:
          /opt/keycloak/data/import/realm-config.json
    # ...
```

Úryvek C.3: Úryvek z konfigurace autentizačních služeb.

```
import { DeviceProfile } from "@glyco-hub/portal-entities"

import type {
  AssertHasUserOrganizationPermissionsUseCase,
  AssertHasUserOrganizationPermissionsUseCaseDependencies,
} from "../users/usersUseCasesApi"
import type { Logger, UseCaseResult } from "../utils/types"
import type {
  QueryOrganizationDeviceProfiles,
  UpdateDeviceProfile,
} from "./deviceProfilesDataApi"

type FindAllDeviceProfilesUseCaseArguments = {
  organizationId: string
  userId: string
}

export type FindAllDeviceProfilesUseCaseDependencies = {
  queryOrganizationDeviceProfiles:
    QueryOrganizationDeviceProfiles
  logger?: Logger
  assertHasUserOrganizationPermissions:
    AssertHasUserOrganizationPermissionsUseCase
} & AssertHasUserOrganizationPermissionsUseCaseDependencies

export type FindAllDeviceProfilesUseCase = (
  args: FindAllDeviceProfilesUseCaseArguments,
  deps: FindAllDeviceProfilesUseCaseDependencies,
) => Promise<UseCaseResult<Array<DeviceProfile>>>
```

Úryvek C.4: Rozhraní případu užití získání všech profilů zařízení.

```
terraform {
  required_providers {
    opennebula = {
      source  = "OpenNebula/opennebula"
      version = "~> 1.2"
    }
  }
}
provider "opennebula" {
  endpoint = var.one_endpoint
  username = var.one_username
  password = var.one_password
}

resource "opennebula _image" "os-image" {
  name          = var.vm_image_name
  datastore_id = var.vm_imagedatastore_id
  persistent    = false
  path          = var.vm_image_url
  permissions   = "600"
}
# ...
```

Úryvek C.5: Definice připojení k univerzitní instanci OpenNebula jakožto poskytovateli cloudových služeb nástrojem Terraform.

```
resource "opennebula_virtual_machine" "glyco-hub" {
  name        = "glyco-hub"
  description = "Glyco Hub VM"
  cpu         = 1
  cpumodel {
    model = "host-passthrough"
  }

  vcpu        = 1
  memory      = 2048
  permissions = "600"
  group       = "users"

  context = {
    NETWORK        = "YES"
    HOSTNAME       = "$NAME"
    SSH_PUBLIC_KEY = "${var.vm_ssh_pubkey}"
  }
  os {
    arch = "x86_64"
    boot = "disk0"
  }
  disk {
    image_id = opennebula_image.os-image.id
    target   = "vda"
    size     = 12000 # 12GB
  }

  graphics {
    listen = "0.0.0.0"
    type   = "vnc"
  }

  nic {
    network_id = var.vm_network_id
  }
}
# ...
```

Úryvek C.6: Konfigurace instance virtuálního stroje pomocí nástroje Terraform.

```
resource "opennebula_virtual_machine" "glyco-hub" {
    # ...
    connection {
        type = "ssh"
        user = "root"
        host = self.ip
        private_key = file("~/.ssh/id_ecdsa")
    }

    provisioner "file" {
        source      = "${path.module}/../../init-scripts/"
        destination = "/tmp"
    }

    # konfigurační soubor Keycloak lze zkopírovat ihned
    provisioner "file" {
        source      = "${path.module}/../../keycloak"
        destination = "/etc"
    }

    provisioner "remote-exec" {
        inline = [
          "export INIT_USER=${var.vm_admin_user}",
          "export INIT_PUBKEY='${var.vm_ssh_pubkey}'",
          "export INIT_LOG=${var.vm_node_init_log}",
          "export INIT_HOSTNAME=${self.name}",
          "touch ${var.vm_node_init_log}",
          "sh /tmp/init-start.sh",
          "sh /tmp/init-node.sh",
          "sh /tmp/init-users.sh",
          "sh /tmp/init-finish.sh"
        ]
    }
    # ...
}
```

Úryvek C.7: Zkopírování inicializačních skriptů do virtuálního stroje z repositáře a jejich spuštění pomocí nástroje Terraform.

```
null_resource.ansible-provisioner (local-exec):
    PLAY [Glyco Hub Backend node setup]
    ****************************************

null_resource.ansible-provisioner (local-exec):
    TASK [Gathering Facts]
    ****************************************************
null_resource.ansible-provisioner (local-exec): ok: [147.228.173.38]

null_resource.ansible-provisioner (local-exec):
    TASK [backend : Login to Gitlab container registry]
    ***************************
null_resource.ansible-provisioner (local-exec): changed: [147.228.173.38]

null_resource.ansible-provisioner (local-exec):
    TASK [backend : docker_compose]
    ********************************************
null_resource.ansible-provisioner: Still creating... [1m10s elapsed]
null_resource.ansible-provisioner (local-exec): ok: [147.228.173.38]

null_resource.ansible-provisioner (local-exec):
    TASK [backend : assert]
    *****************************************************
null_resource.ansible-provisioner (local-exec): ok: [147.228.173.38] => {
null_resource.ansible-provisioner (local-exec):     "changed": false,
null_resource.ansible-provisioner (local-exec):     "msg": "All assertions passed"
null_resource.ansible-provisioner (local-exec): }

null_resource.ansible-provisioner (local-exec):
    PLAY RECAP
    *********************************************************************
null_resource.ansible-provisioner (local-exec):
    147.228.173.38: ok=7    changed=1    unreachable=0
        failed=0    skipped=0    rescued=0    ignored=0

null_resource.ansible-provisioner:
    Creation complete after 1m10s [id=8874284257709439678]

Apply complete! Resources: 1 added, 0 changed, 1 destroyed.
```

Úryvek C.8: Výsledek úspěšného nasazení aplikačního serveru webového portálu nástrojem Terraform.

D Struktura odevzdaného archivu

- `Text_prace` – všechny „zdrojové“ soubory, tj. .tex, .docx, .png apod. a výsledný PDF soubor
  - `A21N0059P_text.pdf` - PDF soubor s textem práce
  - `A21N0059P_text` - adresář se zdrojovými soubory LaTeX a obrázky
- `Poster` - oba soubory posteru
- `Aplikace__knihovny` - adresář obsahující repozitář vytvořených zdrojových kódů
  - `apps/portal` - adresář se samostatnými aplikacemi webového portálu
    - `backend` - adresář s repozitářem zdrojových kódů aplikačního serveru portálu
    - `frontend` - adresář s repozitářem zdrojových kódů uživatelského rozhraní portálu
  - `iac` - adresář s definicí skriptů, konfigurací a deklarací pro správu infrastruktury
    - `ansible` - adresář s definicemi Ansible příruček pro spuštění infrastruktury na cílovém stroji
    - `init-scripts` - adresář s inicializačními skripty pro nastavení systému na cílovém stroji
    - `keycloak` - adresář s konfiguračními soubory služby Keycloak
    - `modules`
      - `ansible-gitlab` - Terraform modul pro generování Ansible proměnných pro autentizaci vůči GitLab registrům
      - `one-infrastructure` - Terraform modul pro zajištění podpůrné infrastruktury systému a spuštění Ansible příruček
      - `one-portal-backend` - Terraform modul pro zajištění kontejneru aplikačního serveru portálu

            - `one-portal-frontend` - Terraform modul pro zajištění kontejneru uživatelského rozhraní portálu

- `packages` - adresář se sdílenými systémovými balíky
    * `builder` - adresář sdíleného modulu pro sestavení aplikačních serverů
    * `clients/backend` - adresář sdíleného modulu s generovaným kódem pro komunikaci s rozhraním aplikačního serveru
    * `configs` - adresář se sdílenými moduly s konfigurací pomocných nástrojů *ESLint*, *Prettier* a nastavením jazyka Typescript
    * `logger` - adresář se zdrojovým kódem sdíleného modulu pro logování
    * `portal`
        - `database` - adresář se zdrojovými kódy modulu datové vrstvy portálu
        - `entities` - adresář se zdrojovými kódy modulu systémových entit portálu
        - `useCases` - adresář se zdrojovými kódy aplikační logiky portálu
- `utils` - adresář sdíleného modulu s užitečnými funkcemi
- `package.json` - soubor s definicemi *NPM* závislostí repozitáře a pomocnými skripty
- `pnpm-lock.yaml` - soubor správce závislostí *PNPM* uzamykající jejich verze
- `pnpm-workspace.json` - definice pracovního prostoru *PNPM*
- `turbo.json` - konfigurace nástroje *Turborepo* pro správu repozitáře
- `README.md` - detailní vývojové infromace o projektu

E Uživatelská příručka

Obsahem této uživatelské příručky bude uvedení postupu použití nasazeného webového portálu a spuštění kompletní infrastruktury v lokálním prostředí ze zdrojových souborů projektu.

Uživatelský manuál i celý projekt předpokládají využití v UNIX-like operačním systému. Některé skripty (například vlastní BASH skripty) nemusí na operačním systému Windows fungovat.

Prerekvizitami pro spuštění a vývoj systému jsou nainstalované nástroje Docker, Docker Compose, Node.js verze `>=20.5.0`, npm a pnpm.

E.1 Spuštění v lokálním prostředí

Pro spuštění systému v lokálním prostředí je nutné provést spuštění tří částí platformy:

- podpůrné infrastruktury
- aplikačního serveru portálu
- uživatelského rozhraní portálu

E.1.1 Spuštění podpůrné infrastruktury

Pro spuštění lokální infrastruktury je nutné mít nainstalovány nástroje Docker a Docker Compose. Po jejich instalaci je nutné vyplnit sadu proměnných prostředí v souboru `iac/.env` podle vzoru v souboru `iac/.env.example` ovšem s platnými hodnotami. Druhým krokem je vyplnění proměnných uvnitř skriptu `iac/nginx/envsubst.sh`.

Následně stačí změnit pracovní adresář na adresář `iac` a spustit skript `envsubst.sh` příkazem `./nginx/envsubst.sh`. V7stupem bude vygenerovaný soubor `proxy.local.conf`, který slouží jako lokální konfigurace reverzní proxy. Posledním krokem pro spuštění infrastruktury je spuštění příkazu `docker-compose up -d`, který nastartuje veškeré kontejnery podpůrné infrastruktury. Spuštění všech služeb je možné pomocí příkazu `docker ps`, který by měl vypsat tabulku s běžícími kontejnery.

E.1.2 Souštění aplikačního serveru

Pro spuštění webového portálu je nutné nejprve nainstalovat potřebné závislosti všech aplikačních modulů a balíků. Proto je třeba v kořenovém adresáži projektu spustit příkaz `pnpm i`. Následně je nutné sestavit všechyn sdílené balíky v repozitáři. Toho lze docílit spuštěním příkazu `pnpm build:packages` v kořenovém adresáři projektu. Před spuštěním aplikačního serveru je nutné inicializovat databázi. Proto je nutné přejít do adresáře `pakcages/portal/database` a spustit příkaz `pnpm dev:migrate`, který provede spuštění databázových migrací nad běžící instancí databázového serveru. Následně je možné přejít do adresáře `apps/portal/backend` a spustit aplikační server příkazem `pnpm dev`. Aplikační srever by následně měl být dostupný na adrese
`http://localhost:4000`, pokud nebyl nastaven jinak.

E.1.3 Spuštění uživatelského rozhraní

Spuštění uživatelského rozhraní je velmi přímočaré, stačí pouze přejít do adresíáře `apps/portal/frontend` a spustit příkaz `pnpm dev`. Uživatelské rozhraní by následně mělo být dostupné na adrese `http://localhost`. Port zde záměrně není uveden, jelikož provoz z portu `:80` je směrován spuštěnou reverzní proxy na port, na kterém běží uživatelské rozhraní.

E.2 Použití spuštěné instance systému

Po spuštění systému je možné se do aplikace přihlásit jako administrátor. Pro přihlášení v roli zdravotníka je nutné nejdříve založit uživatelský účet v grafickém rozhraní Keycloak.

Proto stačí přejít na adresu `http://localhost:8081`, kde je třeba vybrat „Administration console“. Uživatel bude následně přesměrován na autentizační obrazovku, kde je nutné vyplnit administrátorské přihlašovací údaje. Po zadání je uživatel přesměrován do administrátorského grafického rozhraní Keycloak. Zde je následně možné v levé části vybrat sféru `glyco-hub` a přejít na záložku „Users“. Zde je třeba kliknout na tlačítko „Add user“, vyplnit email a uživatelské jméno zdravotníka. Dále je nutné buď nastavit permanentní či jednorázové heslo, kterým se bude uživatel autentizovat.

E.2.1 Navigace v užiatelském rozhraní

Pro přehled funkcionalit jednotlivých obrazovek uživatelského rozhraní a nutných přístupových práv se obraťte na část 8.5, kde jsou všechny části uživatelského rozhraní detailně rozebrány.

E.3 Manuální nasazení do cloudu

Prerekvizitami pro manuální nasazení aplikace do cloudu je mít nainstalovaný jazyk `python` verze 3 a správce závislostí `pipenv`. V neposlední řadě je nutné mít inicializované virtuální Python prostředí v adresáři `iac` následujícím příkazem:

```
python3 -m venv .
```

Dále je nutné nainstalovat potřebné závislosti. Nejprve je třeba nainstalovat nástroj Terraform dle oficiálních instrukcí na webových stránkách nástroje `https://developer.hashicorp.com/terraform/downloads`.

Dalším krokem je vygenerování potřebného SSH klíče, který bude následně použit pro zabezpečený přenos souborů na zajištěnou stanici a vzdálení přístup na ni. Toho lze docílit spuštěním příkazu

```
ssh-keygen -t ecdsa -b 521 -N ''
```

Vygenerovaný soubor je vhodné pojmenovat podle názvu uvedeného v konfiguračním souboru `main.tf` v modulu `one-infrastructure`.

Následně je zapotřebí vstoupit do virtuálního terminálu nástroje `pipenv` příkazem

```
pipenv shell
```

Ve virtuálním prostředí `pipenv` je následně třeba nainstalovat nástroj Ansible a jeho závislosti příkazem

```
python3 -m pip install ansible
```

E.3.1 Nasazení podpůrné infrastruktury

Prvním krokem při nasazení systému je zajištění podpůrné infrastruktury. Pro nasazení podpůrné infrastruktury do vlastní instance OpenNebula stačí přejít do Terraform modulu `one-infrastructure` a nastavit potřebné proměnné uvnitř souboru `terraform.tfvars`. Pokud se jedná o první nasazení systému, je třeba inicializovat modul příkazem

```
terraform init
```

Následně je možné aplikovat konfiguraci spuštěním příkazu

```
terraform apply -auto-approve
```

E.3.2 Nasazení webového portálu

Proces nasazení obou služeb webového portálu totožný. Nejprve je nutné sestavit produkční verze služeb uvnitř modulů `apps/portal/frontend` a `apps/portal/backend` příkazem

```
pnpm build
```

Následně je možné buď využít již existující verzi obrazů aplikačních kontejnerů nebo vytvořit nové. Pokud si uživatel přeje použít existující verzi obrazů kontejnerů, lze přeskočit na další krok. Pokud si uživatel přeje vytvořit novou verzi obrazů aplikačních kontejnerů, lze toho docílit spuštěním skriptů

```
./scripts/publish.sh
```

V rámci vykonání skriptu bude uživatel nucen se autentizovat u kontejnerových registrů univerzitní instance GitLab. Toho lze docílit zadáním uživatelského jména a osobního přístupového tokenu jako hesla, který je možné vytvořit v nastavení instance GitLab. Po úspěšné autentizaci dojde k publikování vytvořených obrazů.

Následně je možné zopakovat příkazy z předchozí části, tedy přejít do Terraform modulů `one-portal-frontend` a `one-portal-backend` a jedná-li se o prvotní nasazení systému, inicializovat moduly příkazem

```
terraform init
```

Jakmile jsou moduly inicializovány, je možné služby webového portálu nasadit příkazem

```
terraform apply -auto-approve
```

E.3.3 Vzdálený přístup na spuštěnou stanici

Pro vzdálený přístup na spuštěnou stanici je třeba využít dříve vygenerovaného SSH klíče. Dále je nutné znát jméno systémového uživatele nastavené v proměnných modulu `one-infrastructure` a IP adresu spuštěné stanice. Pokud tato adresa není známa, je možné ji zjistit následujícího příkazu v modulu `one-infrastructure`:

```
terraform output
```

Tento příkaz zobrazí veškeré výstupy z daného Terraform modulu, v tomto případě výslednou IP adresu vytvořené stanice.

Následně je možné se na stanici připojit pomocí SSH příkazem

```
ssh <jmeno_uzivatele>@<ip_adresa_stanice>
```

E.4 Adresy, které je dobré znát

- `http://<ip_adresa_stanice/localhost>` - vstupní bod uživatelského rozhraní
- `http://<ip_adresa_stanice/localhost>:4000` - aplikační rozhraní serveru
- `http://<ip_adresa_stanice/localhost>:8081` - vstupní bod do administrátorského rozhraní Keycloak
- `http://<ip_adresa_stanice/localhost>:8025` - administrátorské rozhraní mailového serveru
- `http://<ip_adresa_stanice/localhost>:3000` - uživatelské rozhraní MQTT brokera