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

Diplomová práce

Rozšíření aplikace pro správu uživatelů

Plzeň 2022 Jakub Hlaváč

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2021/2022

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Jakub HLAVÁČ**
Osobní číslo:	**A20N0050P**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Informační systémy**
Téma práce:	**Rozšíření aplikace pro správu uživatelů**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se s aplikací aimtec.cloud pro správu uživatelů a přístupových práv do multitenantního systému.
2. Seznamte se s technologiemi Amazon Web Services používanými v aplikaci aimtec.cloud.
3. Analyzujte business procesy současného řešení správy uživatelů aimtec.cloud.
4. Navrhněte rozšíření stávající aplikace aimtec.cloud o návrhy webové aplikace (včetně návrhu UX) a rozšíření backendu.
5. Navržená rozšíření implementujte, v případě backendu použijte technologii AWS Lambda v jazyce Java nebo Python.
6. Implementovanou funkcionalitu otestujte jednotkovými a integračními testy.
7. Vyhodnoťte dosažené přínosy.

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. Jiří Dobrý**
Aimtec a.s.
Konzultant diplomové práce: **Doc. Ing. Roman Mouček, Ph.D.**
Katedra informatiky a výpočetní techniky

Datum zadání diplomové práce: **10. září 2021**
Termín odevzdání diplomové práce: **19. května 2022**

L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

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

V Plzni dne 11. října 2021

Prohlášení

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

V Plzni dne 17. května 2022

Jakub Hlaváč

Poděkování

Tímto bych chtěl poděkovat vedoucímu mé práce Ing. Jiřímu Dobrému za podporu při řešení problémů, připomínky a za čas, který strávil vedením této diplomové práce. Programátorům ve firmě Aimtec, za kterými jsme mohl přijít v případech, kdy jsem si nevěděl rady s implementací. A velké díky si také zaslouží má rodina, která mě při psaní práce podporovala jak finančně tak psychicky.

Abstract

The main purpose of this thesis is to design and implement the necessary extension for the management of users and their access rights into a multitenant environment within aimtec.cloud. The enlargement of aimtec.cloud Portal is needed to simplify the current process of management of users, applications and access rights in a multitenant environment. First part contains an introduction of the current implementation of aimtec.cloud portal with used processes. The following part is focusing on a deeper understanding and analysation of the Amazon Web Services. The final user interface wireframe consists of customer's requirements. This UI draft is then consulted with an external specialist. The main part of this thesis is to implement the solution which consists of individual Java applications deployed into the AWS Lambda function and libraries for working with Amazon DynamoDB and Amazon Cognito. The implementation of the user interface is done into existing Angular application aimtec.cloud Portal. After the solution is tested by unit and integration tests, there follows the deployment into a production environment. After the deployment of the final solution, the process of user management and access rights is simplified. The process is now easily transferable among employees and there is no need for a specially trained programmer. At the same time the consistency of stored data is perfectly ensured.

Abstrakt

Cílem práce je navrhnout a implementovat rozšíření nutné pro správu uživatelů a přístupových práv do multitenantního prostředí v rámci aimtec.cloud. Rozšíření aplikace aimtec.cloud Portál je potřeba pro zjednodušení procesu správy uživatelů, aplikací a přístupových práv v multitenantním prostředí. Za účelem vypracování této práce bylo provedeno seznámení s aktuální implementací aimtec.cloud Portálu a používaných procesů. Následovalo hlubší porozumění používaných služeb Amazon Web Services. Po provedení analýzy byl udělán návrh konečného řešení, kdy byly zapracovány požadavky zákazníka a vytvořen wireframe uživatelského rozhraní. Návrh UI byl následně konzultován s externím specialistou. Hlavní částí práce je implementace navrženého řešení, kdy byly v jazyce Java implementovány jednotlivé aplikace nasazované do AWS Lambda funkcí a knihovny pro práci s Amazon DynamoDB a Amazon Cognito. Implementace uživatelského rozhraní byla provedena do již existující Angular aplikace aimtec.cloud Portál. Řešení bylo otestováno jednotkovými a integračními testy a následovalo jeho nasazení do produkčního prostředí. Po nasazení výsledného řešení došlo k zjednodušení procesu správy uživatelů a přístupových práv, kdy je proces snadno převoditelný mezi zaměstnanci a není již potřeba speciálně proškoleného programátora. Zároveň je zajištěna konzistentnost ukládaných dat.

Obsah

1 Úvod

Moderní webové aplikace jsou v dnešní době hojně nasazovány do cloudového prostředí, které pro ně znamená outsourcování kompletní infrastruktury, na které jsou následně provozovány. Odpadá tedy nutnost mít vlastní rozsáhlé IT oddělení se specializovanými odborníky na jednotlivé části dané infrastruktury, která může být u dnešních velice sofistikovaných IT systémů značně komplexní. Zároveň přináší menší jednorázovou zátěž pro firmu, která se pro něj rozhodne, jelikož odpadají velké náklady na vybudování kompletní infrastruktury. Firma, která se tedy rozhodne pro využití cloudových služeb, následně platí pravidelné poplatky za licence, které ji opravňují využívat daný software.

Aimtec.cloud je cloudové řešení od firmy Aimtec, která následuje současné trendy v IT technologiích a svým zákazníkům začala nabízet své IT řešení v cloudové podobě. Tato diplomová práce se zabývá analýzou, návrhem a implementací efektivního a jednoduchého řešení pro správu aplikací, které Aimtec nabízí v cloudové podobě, a řízení přístupových práv jednotlivých zákazníků. Jelikož je webové rozhraní aimtec.cloud navrženo pro multitenantní použití, je potřeba zvažovat i bezpečnostní stránku celého řešení. V rané fázi nasazení systému bude využíván jen pověřenými administrátory ze strany Aimtecu, nicméně systém má být připraven i na variantu, kdy každý zákazník bude mít svého administrátora, který se bude starat o své uživatele a bude je moci spravovat.

Celá práce je rozčleněna do několika kapitol, kdy v prvním z nich je provedena analýza a seznámení se se stávající podobou portálu aimtec.cloud. Následuje kapitola věnující se použitým technologiím Amazon Web Services, na kterých je postavena kompletní funkční infrastruktura cloudového řešení od Aimtecu. Ve třetí kapitole jsou analyzovány business procesy správy uživatelů a aplikací, které chce firma Aimtec zjednodušit a co nejvíce zautomatizovat, aby došlo k eliminaci co největšího počtu ručních kroků. Čtvrtá kapitola práce je věnována návrhu rozšíření všech potřebných služeb stávající aplikace, které jsou využívány pro backendovou i frontendovou část. Po dokončení implementace nových částí systému jsou všechny vzniklé části důkladně otestovány integračními i jednotkovými testy. Závěr práce tvoří diskuze a zhodnocení přínosů implementovaného řešení z pohledu zjednodušení procesu a snížení nákladů na správu uživatelů a aplikací pro cloudové řešení firmy Aimtec.

2 Portál aimtec.cloud

Tato kapitola se věnuje popisu webové aplikace společnosti Aimtec a.s.. Tato aplikace je využívána pro přístup jejích zákazníků k aktuálně nabízeným cloudovým službám. V rámci této diplomové práce se do této aplikace bude provádět implementace kompletního rozšíření pro správu uživatelů a běžících aplikací. Toto rozšíření poté bude přístupné pro vybrané divize firmy, které se budou starat o evidenci svých zákazníků samostatně.

2.1 Aimtec

Firma Aimtec a.s. vznikla jako čistě česká společnost v roce 1996 a od svého založení se specializuje na digitalizaci výroby a logistiky pro své zákazníky. Za dobu své působnosti nasbírala firma řadu zkušeností a může se představovat jako firma s globálním dosahem, jelikož svá řešení nabízí po celém světě. Těmto zákazníkům pomáhá implementovat technologické inovace. Firma reflektuje nejmodernější trendy v oblasti digitalizace a do svých produktů zahrnuje koncepce Industry 4.0 a systémy zákazníků se snaží přesouvat z on premise řešení do cloudu pro pohodlnější správu a vyšší dostupnost. [2] Aimtec a.s. má v současnosti celkem pět hlavních divizí. Implementované řešení pro správu uživatelů v cloudu budou primárně používat následující z nich:

2.1.1 DCI

Největší divize firmy, která se zabývá kompletním vývojem informačního systému DCIx, které nabízí zákazníkům komplexní řešení pro jejich řízení logistiky a výroby [16]. Divize je rozdělena do několika týmů po 5-10 programátorech, kteří spravují a rozvíjí jednotlivé části aplikace. Tyto části reflektují dodavatelsko-odberatelský řetězec a každý tým se specializuje na jeho jednu konkrétní část. O nastavení aplikace u zákazníků se starají následně konzultanti, kteří tvoří spojovací článek mezi programátory a zákazníky.

2.1.2 INT

Oddělení firmy zabývající se systémovou integrací a elektronickou výměnou dat (EDI). Oddělení se stará o produkt ClouEDI, který je provozován

v cloudové infrastruktuře a nabízí zákazníkům jednotné rozhraní pro komunikaci mezi celopodnikovými informačními systémy. Stará se o transformaci a výměnu zpráv mezi jednotlivými IS zákazníků. [4]

2.1.3 SUP

Aimtec Support je oddělení firmy, které nabízí podporu ve formátu 24/7 pro všechny produkty firmy Aimtec a.s., či podporu zákazníků a jejich interních uživatelů při práci s HW a SW [3]. Toto oddělení nabízí několik produktů, které jsou vhodné pro zákazníky z výrobního prostředí:

- Product Support - jedná se o komplexní balíček služeb a nástrojů pro podporu podniků
- Cutomer Support - podpora pro dodávané služby, projekty nebo produkty
- End User Support - podpora pro interní uživatele v oblasti HW i SW

2.2 Cloud vs On-Premise

V této sekci se podíváme na jednotlivé možnosti nasazování aplikací a zaměříme se na případné problémy a výhody, které má bezesporu každé řešení.

2.2.1 Cloud

Jedná se nový trend dodávání kompletních výpočetních služeb od úložišť a serverů až po softwarové vybavení, analytické nástroje a inteligentní funkce přes internet. Principem je tedy propůjčování výpočetního výkonu/úložiště serverů uživatelům. Je to vlastně jakýsi počítač někoho jiného, na kterém je puštěn vybraný software. Model dodávky tohoto řešení je cenově přívětivý k uživatelům, jelikož platíme právě jenom za to, co aktuálně využíváme. Tento přístup pomáhá silně snižovat provozní náklady a efektivně provozovat infrastrukturu s ohledy na neočekávané změny v potřebách výkonu, kdy je zajištěna škálovatelnost našeho řešení ze strany dodavatele. Druhým zúčtovacím modelem v cloudu je předplatné, kdy je předem jasně daná částka pro danou službu a její rozsah v časovém období. Díky pandemii COVID-19 se o cloudová řešení začala zajímat ještě větší skupina firem, než v předchozích letech, což vyústilo v 20% růst trhu s cloudovými službami [23]. Jednotlivé modely služeb nasazení cloudu jsou následující:

- IaaS (Infrastruktura jako služba) - je to nejzákladnější kategorie cloud computingu. V této kategorii si lze pronajmou IT infrastrukturu ve formě virtuálních počítačů či serverů, úložiště a siťových prostředků. Jedná se o základní stavební prvek cloudové infrastruktury, kdy máme vařešeny i základní otázky ohledně siťové bezpečnosti, jelikož jsou nakoupené servery za firewallem daného poskytovatele.

- PaaS (Platforma jako služba) - jedná se o úplné prostředí, které je připraveno pro vývoj a nasazení aplikací v cloudu. Poskytuje kompletní prostředky pro dodávání řešení od těch nejjednodušších až po propracované informační systémy. PaaS je nastaven tak, aby podporoval kompletní životní cyklus informačního systému od jeho sestavování přes testování a nasazení až po správu a následnou aktualizaci.

- Serverless (Architektura bez serveru) - v tomto segmentu dochází k překryvu s PaaS, kdy je cílen především na vytváření aplikačních funkcí bez potřeby starat se o servery či infrastrukturu. Jejich kompletní správu přebírá poskytovatel cloudu a stará se i o jejich škálovatelnost. Tyto architektury jsou řízené událostmi a prostředky pro jejich běh jsou využívány pouze v případě jejich použití.

- SaaS (Software jako služba) - jedná se o metodu doručování softwarových řešení pomocí internetu, které je zpoplatněno ve formě předplatného. V této skupině se o vše stará cloudový poskytovatel, který je zodpovědný za kompletní infrastrukturu, údržbu, opravy a upgrady aplikace. Koncoví uživatele se k nim následně připojují přes internet převážně pomocí webových prohlížečů.

Cloud je dále možno také rozdělovat podle modelu nasazení, kdy je pro vybudování cloudové infrastruktury možno využít pouze vlastních zařízení, pronajmout si hotovou službu od poskytovatele nebo obě možnosti kombinovat do jedné. V závislosti na zvoleném přístupu jsou cloudy děleny do tří typů:

- Soukromý cloud - jedná se o cloudovou infrastrukturu vlastněnou jednou společností. Je tvořena na bázi vlastní hardwarové infrastruktury či pronajatém zařízení, o které se stará ICT oddělení společnosti. Při zvolení tohoto typu cloudu nedochází ke sdílení fyzických ani virtuálních strojů. Výhodami může být vyšší autonomie a snadný přístup k výpočetním zdrojům. Nevýhodou jsou vyšší náklady na vytvoření, obtížné škálování či složitější návrh.

- Veřejný cloud - cloudová infrastruktura, kterou společnost využívá, je ve vlastnictví poskytovatele cloudových služeb a klientská společnost si ji jenom pronajímá. Z ekonomického hlediska je na tento typ přístupu brán za službu. Jelikož poskytovatel virtuální zdroje přiděluje klientské společnosti v rozsahu, v jakém je aktuálně potřebuje, a ta na nich provozuje své aplikace. Výhodou je snadné používání, kdy stačí mít přístup k internetu a snížení nákladů na údržbu IT. Připojení k internetu je bohužel také nevýhodou, protože výkon a dostupnost cloudu závisí právě na jeho stabilitě. Pro mnoho organizací je zde také nevýhodou závislost na vnější organizaci, které cloudová infrastruktura patří.

- Hybridní cloud - jedná se o využívání vlastních výpočetních zdrojů i pronajatých kapacit od poskytovatele cloudových služeb. Hybridní infrastruktura se využívá převážně při potřebě mít dostupné vlastní úložné systémy, ale zpracování dat je možno provádět v cloudu. Díky kombinaci dvou řešení, které jsou často provozovány na odlišných technologiích, je zde vysoké riziko nehod.

Výhodou využívání cloudového řešení může být především snížení počátečních nákladů na vybudování infrastruktury pro provoz informačního systému. Na pomezí výhody a nevýhody se následně nachází otázka bezpečnosti dat. Zde je nutno specifikovat, s jakými daty daná společnost manipuluje a zda je může do cloudu ukládat. Nicméně poskytovatelé cloudových služeb jsou průkopnící v bezpečnosti a ta je u nich na prvním místě, tudíž používají bezpečnostní opatření, která si většina organizací nemůže dovolit a slabým článkem bývá vždy uživatel. Hlavní nevýhodou je závislost na internetovém připojení, kdy je pro práci s aplikacemi v cloudu naprosto nezbytné. Infrastrukturu spravuje poskytovatel a organizace nad ní nemá kompletní kontrolu, tento fakt je pro některé společnosti překážkou, a proto se cloudu vyhýbají.

2.2.2 On-Premise

Jedná se o přístup v tvorbě informačních systémů, který má stále své zastánce. On-premise řešení představuje software či hardware, který je stále provozován uvnitř organizace. V případě hardwaru se jedná o fyzické počítače, které vlastní daná firma. S tímto faktem je tedy potřeba myslet na nutnost udržovat dostatečně velké IT oddělení. V dnešní době je na trhu nepřeberně mnoho HW řešení a již střední firmy jich kombinují několik najednou, tudíž není možno, aby IT oddělení o několika málo zaměstnancích

zvládalo dostatečně dobře všechna tato řešení spravovat. Oproti řešení v cloudu je toto řešení pro nově vzniklou firmu daleko finančně náročnější, kdy je potřeba koupit kompletní HW pro provoz potřebné infrastruktury. Při návrhu a nákupu HW je také nutné myslet na možnosti rozšiřování a je tedy doporučováno navrhovat a nakupovat potřebné komponenty s minimálně 10% výkonnostní rezervou pro možnost omezeného škálování. Následně je potřeba nakoupené servery zabezpečit, kdy se kromě nebezpečí ze strany uživatelů firma musí potýkat i se zajištěním bezpečnostních pravidel v přístupu k hardwaru a omezení případných vnějších vlivů. Za tuto cenu je organizaci odměnou 100% kontrola nad řešením.

Z pohledu software se u on-premise řešení jedná o aplikace, které jsou instalovány a provozovány na počítačích dané organizace. Zde je opět nutno zabývat se otázkou údržby serverů a stanic po softwarové stránce, což s sebou nese zvýšené náklady na specialisty v IT oddělení firmy. Zároveň je zde také vyšší cena jednorázového nákupu licencí pro veškerý potřebný software, kdy jsou na trhu aplikace, které mají svoji cenovou politiku založenou ve většině případů na výkonu hardwaru, na kterém následně běží (např. dle CPU jader, RAM, velikosti disků) případně se lze setkat s přístupy, kdy jsou aplikace naceněny podle počtu uživatelů, kteří je následně mohou využívat. Jelikož se ale licence platí jen jednou, může zde být jednorázová platba ve výsledku nižší než dlouhodobější pravidelné platby za cloudové řešení. Jelikož jsou aplikace instalovány lokálně, není uživatel závislý na připojení k internetu, takže nad bezpečností celého systému má větší kontrolu.

2.3 Portál

Portál aimtec.cloud je multitenantní webová aplikace společnosti Aimtec a.s. umožňující jednotný přístup zákazníků ke cloudovým aplikacím, které firma nabízí. Pro každého zákazníka, který využívá cloudových služeb se tedy jedná o vstupní bod, přes který by měl zákazník přistupovat do svých aplikací. Webová aplikace je kompletně provozována na infrastruktuře Amazon Web Services a ke svému fungování využívá několika jejích služeb, které jsou navzájem propojeny. V aktuální podobě slouží pouze jako rozcestník zákazníka, kterému se na hlavní stránce po přihlášení zobrazí seznam jeho běžících aplikací, přes který se na jednotlivé své aplikace může pohodlně dostat z jednoho centrálního místa. Na tomto dashboardu jsou také zobrazeny podstatné informace ohledně dostupnosti aplikací, seznamu plánovaných odstávek systému a aktuální stav řešení vyskytnuvších se problémů na cloudové infrastruktuře, které mají vliv na dostupnost běžících aplikací. Přístup do

aplikací přes portál je doporučovaným způsobem, jelikož umožňuje zákazníkovi využívat výhod jednotného přihlašovacího systému (Single Sign On, zkráceně SSO), kdy přihlášení do aimtec.cloud slouží zároveň pro přihlášení do daných aplikací a není tudíž nutné zadávat opakovaně přihlašovací údaje pro každou aplikaci zvlášť. Celá aplikace je překládána do dvou jazykových mutací, kdy je podporována angličtina a čeština.

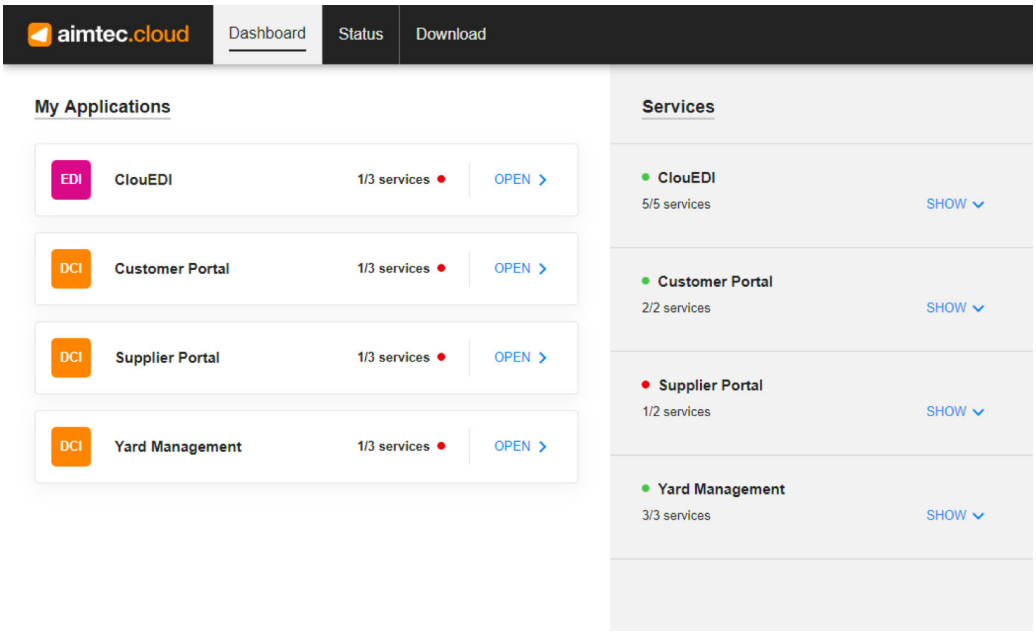
Portál je tedy určen převážně pro zákazníky Aimtecu, kteří využívají některou z cloudových služeb, kterou aktuálně nabízí (EDI, WMS, MES, YMS, Zákaznický portál a Dodavatelský portál [30]). Umožňuje jednotný přístup k těmto aplikacím a jednoduchý přehled o jejich běhu. Zároveň slouží i zaměstnancům Aimtecu, kteří se přes něj přihlašují do aplikací, které pro zákazníka spravují a dělají v nich servisní zásahy. V těchto případech je portál užitečný především pro servisní tým programátorů, kteří potřebují přístup do všech běžících aplikací a takovýto rozcestník je pro ně velice užitečný, jelikož nepotřebují hledat v externích systémech url adresy aplikací a přihlašovací údaje.

2.3.1 Dashboard

Při prvním navštívení stánky je uživateli zobrazena uvítací obrazovka, přes kterou se může přihlásit do cloudového prostředí. Po přihlášení je uživateli zobrazena vstupní obrazovka 2.1, která je rozdělena vertikálně na tři bloky, ve kterých jsou uživateli zobrazeny veškeré potřebné informace. V navigačním menu je možno se pomocí tlačítka *Status* přesměrovat na adresu stránky, která zobrazuje aktuální stav infrastruktury, kterou zákaznické cloudové aplikace využívají a je pod správou Aimtecu. V pravém rohu je možno vedle jména aktuálně přihlášeného uživatele přepnout jazyk, ve kterém chceme celý portál zobrazovat. Na výběr je angličtina a čeština, kdy je výchozí jazyk při navštívení stránky získán z výchozího nastavení prohlížeče, přes který uživatel na portál přistupuje. Po kliknutí na uživatelské jméno přihlášeného uživatele je zobrazeno vysouvací menu, ve kterém se může uživatel odhlásit z portálu a je mu zobrazena uvítací obrazovka.

První blok zleva zobrazuje přihlášenému uživateli jemu dostupné aplikace, které má v cloudovém prostředí nakonfigurovány a ke kterým má nastavena přístupová práva. U každé aplikace je ikona zobrazující o jaký typ se jedná:

- EDI - systém pro elektronickou výměnu dat, spravovaný a vyvíjený integračním oddělením firmy Aimtec a.s.
- DCI - jedná se o informační systém DCIx, který je kompletně vyvíjen



Obrázek 2.1: Portál aimtec.cloud [30]

firmou Aimtec a.s.

Za ikonou je zobrazen název dané aplikace, který je následován opět grafickým indikátorem v podobě barevné tečky. Tato barevná tečka indikuje, zda je aplikace dostupná (zelená tečka) či nikoliv (šedá tečka). Za tímto indikátorem je svislou čarou oddělena část, přes kterou se uživatel může přesměrovat na url dané aplikace. Toto přesměrování je provedeno v nově otevřené záložce prohlížeče, aby uživatel nemusel na seznam aplikací přistupovat opakovaně, pokud chce navštívit vícero aplikací zároveň.

Prostřední blok stránky je věnován jednotlivým službám, které v každé zákaznické aplikaci běží. Opět je zde indikátorem v podobě tečky zobrazena dostupnost jednotlivých částí aplikace, kdy zelená tečka indikuje úplnou dostupnost a tečka žlutá značí nedostupnost některé ze služeb aplikace. Počet správně bežících a nedostupných služeb je zobrazen na řádce pod názvem aplikace, kdy první číslo značí počet běžících a druhé číslo počet všech. Pro zobrazení detailního seznamu zde slouží tlačítko na konci každého záznamu aplikace, po jehož zmáčknuti dojde k zobrazení seznamu jednotlivých služeb dané aplikace. U každého záznamu je opět graficky znázorněn indikátor stavu a je díky němu snadné určit dostupnost a správnou funkčnost.

Pravý blok stránky zobrazuje informace o plánovaných odstávkách na cloudové infrastruktuře, aktuální problémy infrastruktury a jejich momentální stav řešení. Zároveň jsou zprávy o plánovaných odstávkách používány také pro zobrazování důležitých zpráv zákazníkům.

3 Infrastruktura aimtec.cloud Portálu

Cloudové služby, které firma Aimtec a.s. nabízí, jsou primárně provozovány na infrastruktuře Amazon Web Services. Společnost Amazon.com Inc byla vybrána jako silný partner na poli cloudových řešení, jelikož poskytuje služby ve třech modelech: SaaS, PaaS a IaaS. U těchto služeb také garantuje velmi vysokou dostupnost a škálovatelnost. [33]

3.1 Amazon Web Services

Amazon Web Services je platforma na které jsou nabízeny cloudové služby firmy Amazon.com Inc již od roku 2006. Jedná se tedy o nejdéle fungující platformu, která cloudové služby nabízí. Tato nabídka se neustále rozrůstá a vyznačuje se především svoji nízkou cenou a vysokou kvalitou. V případě cloud computingu od AWS se jedná o garantovanou základní měsíční dostupnost služeb od 99,99% (SLA[1]) pro každý jeden region [7]. Této vysoké dostupnosti je dosaženo díky rozdělení infrastruktury na takzvané **regiony**, které jsou následně rozděleny do několika **zón dostupnosti**. K březnu roku 2022 má AWS po celém světě celkem 26 geografických regionů a dalších 8 plánovaných [20]. Region je navržený tak, aby byl izolován od ostatních regionů a v případě problémů byl zasažen jen dotčený a nedošlo k ovlivnění funkčnosti ostatních. Tímto designem je dosaženo nejlepší možné stability. Jednotlivé regiony jsou následně rozděleny do několika zón dostupnosti, kdy opět každá zóna představuje izolovanou část infrastruktury. K březnu roku 2022 je pro 26 regionů definováno celkem 84 zón dostupnosti s plánem jejich rozšíření o 24 nových [20]. Jelikož se jedná o izolované části infrastruktury, kdy výpadek jedné neovlivní ostatní, je možno tohoto faktu využít pro dosažení lepší dostupnosti aplikací jejich replikací do vícero zón dostupnosti v daném regionu. Amazon Web Services se také zaměřují na bezpečnost dat a považují se za jednoho z nejbezpečnějších poskytovatelů cloudových služeb, kdy garantují šifrování dat na úrovni fyzické vrstvy při jejich přeposílání mezi jednotlivými datacentry. Mezi nejsilnější stránky poskytovatele zajisté patří

[1]Service-Level Agreement - jedná se o smlouvu mezi poskytovatelem a uživatelem služby, která definuje parametry a kvalitu poskytovaných služeb spolu se sankcemi. Zdroj: `https://www.bluepartners.cz/slovnik-it-pojmu/sla/`

oblast strojového učení a umělé inteligence, která je následována serverless architekturou v podobě AWS Lambda funkcí.

3.1.1 Srovnání AWS s ostatními cloudovými poskytovateli

Na poli poskytování cloudových služeb se vyskytují další velcí hráči. Jedná se hlavně o Microsoft s jejich produktem Microsoft Azure a Google, který nabízí možnosti cloud computingu pod názvem Google Cloud Platform. Následně se na poli cloudu vyskytuje také Oracle Corporation se svým produktem Oracle cloud.

Microsoft Azure je cloudová platfroma nabízející více jak 200 produktů a služeb. Svých více než 200 datacenter má rozmístěno po celém světe a podobně jako AWS je má rozděleno do několika oblastí, které jsou navzájem propojeny. Díky tomu garantuje dostupnost většiny svých služeb 99.99% [32]. Nejsilnější stránkou Azure je především poskytování cloudového řešení produktů firmy Microsoft jako je například SQL Server, Office a Sharepoint. Zároveň poskytuje k již stávajícím řešením ve formě on-premise podporu v podobě nástrojů pro jejich zálohování či hostování kolektivního vývoje projektů pro Visual Studio.

Google Cloud Platfrom je služba od technologického gigantu Google. Stejně jako předchozí zmíněné cloud computing služby je infrastruktura rozdělena do 29 regionů, které jsou následně rozděleny na zóny, kterých je 88. Tuto službu je tedy možno využívat ve více jak dvě stě zemích světa [21]. Podle dostupných SLA podmínek se garance dostupnosti jednotlivých služeb pohybuje od 99.95% do 99.99% [22]. Zajímavostí je zde fakt, že služba běží na stejné infrastruktuře jako nejznámější produkty Google Search engine a YouTube. Google Cloud Platfrom lze rozhodně zařadit jako lídra v oblasti vývoje umělé inteligence díky komplexní podpoře knihovny TensorFlow. Zároveň lze mezi produkty najít nástroje pro práci s přirozeným jazykem a překlady, jelikož tyto produkty jsou používány i samotnou firmou v jejích rozšířených nástrojích Google Search a Google Translate.

Pokud jde tedy o rozhodnutí, jakého dodavatele vybrat pro cloudové řešení služeb, je zde těžké vybírat, jelikož všichni velcí hráči na tomto poli nabízejí kvalitní řešení s podobnými nabídkami. Nicméně lze říci, že AWS je jasným lídrem co do počtu zákazníků a počtu služeb, které nabízí. Na druhou stranu Microsoft Azure je nyní nejziskovější cloud computing platforma, která díky přenosu svých on-premise řešení jako jsou SQL Server, Sharepoint či Office získává zákazníky z řad společností, kde jsou tyto produkty využívány již delší dobu a je tlačeno na jejich přechod do cloud podoby. Google

Cloud má aktuálně nejmenší zastoupení na trhu, nicméně nabízí své služby za nižší ceny než konkurence a je hlavním hráčem na poli kontejnerizace aplikací, jelikož Google vyvinul Kubernetes[2] standard, který je nyní velice rozšířený a dal by se považovat za nový standard při nasazování nových informačních systémů. Výběr poskytovatele cloudové platformy může být ovlivněn subjektivními preferencemi zákazníků a závisí na nabízených možnostech jednotlivých řešení. Sic jsou jejich nabízené produkty a služby velice podobné, každý s nich má oblast, ve které nad ostatními vyniká.

Tabulkové srovnání

Tabulka 3.1: Srovnání největších poskytovatelů cloudových služeb. [12][8]

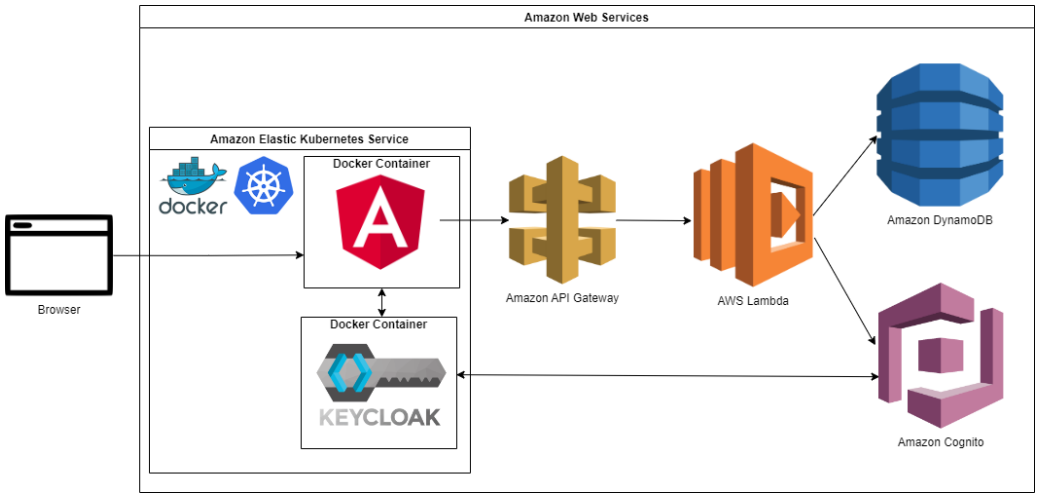
	Amazon Web Services	Microsoft Azure	Google Cloud Platform
Počet let na trhu	16 (2006)	12 (2010)	14 (2008)
Podíl na trhu (Q4 2021)	33%	22%	9%
Roční příem (Q4 2021)	17.8 bilionů USD	18.3 bilionů USD	5.54 bilionů USD
Meziroční růst	40%	26%	45%
Zkušební doba	12 měsíců	12 měsíců	12 měsíců
Zóny dostupnosti	26 regionů (84 zón dostupnosti)	60 regionů	29 regionů (88 zón dostupnosti)
Výpočetní služby	Elastic Compute Cloud (ECs)	Virtual Machines	Compute Engine
Podporované OS	Linux (AWS Linux, CentOS, Debian Server a další), macOS, Raspberry Pi OS, Windows Server	Windows Server, Linux (CentOS, CoreOS, Debian a další)	Windows Server, Linux (CentOS, Debian, Ubuntu, SUSE, RHEL a AWS Linux)
NoSQL databáze	AWS DynamoDB	Azure DocumentDB	Cloud Bigtable
SQL Databáze	Amazon Aurora, MySQL, PostgreSQL, MariaDB, Oracle, SQL Server	SQL Server, PostgreSQL, MariaDB	MySQL, PostgreSQL, SQL Server
Serverless výpočetní služby	AWS Lambda (Java, Go, Node.js, C#, Python, Ruby)	Azure Functions (C#, JavaScript, F#, Java, Python, TypeScript2)	Google Cloud Functions (Node.js, Python, Go, Java, .NET, Ruby, PHP)

3.2 Používané AWS služby

K běhu aplikace aimtec.cloud Portál je tedy využíváno několika AWS služeb od konfigurace API rozhraní až po databázovou vrstvu pro ukládání dat viz. obrázek 3.1. O API definice pro komunikaci s backendovou službou se stará služba API Gateway, toto rozhraní je napojeno na backend službu v podobě několika na sobě nezávislých AWS Lambda funkcích. Tyto implementované funkce jsou následně propojeny s databázovou vrstvou v podobě Cognito (služba pro správu uživatelů a jejich přístupů viz. kapitola 3.2.4) a DynamoDB (dokumentová NoSQL databáze viz. kapitola 3.2.3), kde jsou

[2]Kubernetes - jedná se o open-source systém, který je využíván k automatizování nasazování a škálování aplikací. Zdroj: `https://kubernetes.io/`

ukládány informace o jednotlivých uživatelích. Jednotlivé služby jsou popsány v následujících kapitolách.



Obrázek 3.1: Infrastruktura aimtec.cloud Portálu

3.2.1 Amazon API Gateway

Amazon API Gateway je služba poskytující jednoduché nástroje pro tvorbu, publikování, údržbu a především zabezpečení komunikace mezi klientem a serverem, která probíhá pomocí REST nebo WebSocket spojení. Pomocí těchto API lze publikovat endpointy jednotlivých aplikací, které běží na infrastruktuře AWS. Jedná se tedy o pomyslný základní pilíř pro architekturu mikroslužeb či pro bezserverovou architekturu, kdy potřebujeme dané požadavky směrovat na jednotlivé služby či funkce. Cenová politika této služby je přínosná pro uživatele, jelikož je zde cena služby odvozena od jejího zatížení, kdy je za měrnou jednotku považováno volání daného API endpointu. Při vytváření API definic je možné zvolit z několika přístupů, kdy každý má svoje výhody a nevýhody, tyto přístupy jsou:

- REST API - sofistikovanější přístup definování API, kdy je možno každé definici dodat pravidla pro validaci a transformaci vstupních dat, která budou následně poskytnuta backendu, či explicitně nastavit výstupní objekty, které bude API volání vracet. Jedná se tedy o komplexnější definování API endpointu, které je postaveno na architektuře modelů pro dotaz a odpověd, kde komunikace mezi klientem a službou probíhá synchronně. Datové modely jsou definovány ve formátu JSON. Pro zajištění bezpečnosti lze na jednotlivých definicích zvolit způsob autorizace jednotlivých volání, kdy je možno využít ostatních služeb

AWS jako je na příklad Amazon Cognito či AWS IAM role a profily. Cena volání jednotlivých API enpointů je díky dostupnosti těchto možností vyšší než cena volání HTTP API. Cena jednoho volání se zde pohybuje od 3,5 USD za milion volání pro prvních 333 milionů. Cena je následně snižována podle úrovní, až na cenu 1.51 USD při více jak 20 miliard volání měsíčně [5].

- HTTP API - taktéž umožňuje specifikaci endpointů ve formě REST API, kdy je možno využít nejnovějšího standardu OpenAPI 3.0, jedná se o definování vstupní brány, která bude příchozí požadavky přeposílat na předem definované endpointy backendu ve formě AWS Lambdy či veřejně dostupných endpointů. Jedná se o zjednodušenou formu již zmíněné verze REST API, kdy lze tuto variantu použít v případě, že na vstupní požadavky neklademe potřebu validace a není nutno do API zakomponovávat datové modely, které mají dané definované cesty akceptovat či vracet. Zároveň zde přicházíme o možnost definovat zabezpečení na úrovni AWS IAM role či používání Amazon Cognita. Zachovány jsou zde možnosti autorizace a autentikace pomocí JWT tokenů (která je dostupná i pro možnost REST API). Díky odlehčení ve validaci a transformaci dat lze očekávat snížení nákladů a odezvy jednotlivých requestů. Oproti definování API ve formátu REST API se zde cena pohybuje od 1 USD za milion volání pro prvních 300 milionů. Nad 300 milionů volání je následně cena snížena na 0.9 USD [5].

- WebSocket API - jedná se o kolekci WebSocketových cest, které jsou integrovány s ostatními AWS službami či HTTP endpointy. Tato spojení jsou obousměrná a umožňují bohatší interakce klient/služba, protože služby mohou klientům posílat data, aniž by bylo potřeba explicitního požadavku od klientů. Toto odesílání zpráv mezi klientem a službou lze provádět pomocí WebSocketů nezávisle na sobě.

Definice API probíhá v podobě definování url zdrojů, nad kterými se poté definují jednotlivé metody, které lze volat. Metodami jsou myšleny základní HTTP metody GET, POST, PUT a DELETE, které jsou používány pro odesílání dat z klienta na server či získávání dat ze serveru pro konzumaci klientem. API Gateway podporuje i metodu ANY, která přijímá jakýkoliv typ požadavku, který přijde na daný url zdroj [31]. Výhodou správy API v rámci této služby je také možnost dané API verzovat a v případě problémů na to aktivně reagovat vrácením posledních nasazených změn do stavu, kdy byla API Gateway funkční. Verzování API je možno využít pro udržování několika najednou běžících verzí pomocí mechanismu, který se zde nazývá **stage**.

Tyto **stage** zpřístupňují nasazenou verzi API definici s předem definovaným prefixem. Níže jsou zobrazeny příklady URL pro stage *prod*, *test*, *v1* a *v2*:

```
GET /prod/solutions
GET /test/solutions
GET /v1/solutions
GET /v2/solutions
```

Této možnosti lze využít v případě provádění nekompatibilních změn rozhraní. Pro nové rozhraní vytvoříme stage s novým prefixem. Díky tomu zachováme starou verzi API na stále stejné adrese a pro nově nasazenou verzi API vznikde adresa odlišná, tyto dvě verze API se následně dají provozovat simultánně.

JSON

JavaScript Object Notation je formát dat používaný především pro výměnu dat. Tento formát ukládání dat je snadno čitelný pro stroj i člověka. Zároveň je díky striktním pravidlům snadno data ve formátu JSON strojově generovat. Jelikož se jedná o textový a na jazyce zcela nezávislý formát, je vhodný pro použití napříč všemi programovacími jazyky. JSON podporuje následující datové typy:

- řetězec - text obalený apostrofy
- číslo - podpora float a integer
- logický datový typ - true/false
- pole - definováno pomocí hranatých závorek
- objekt - definován pomocí složených závorek

Data se tedy zapisují jako páry název/hodnota, kdy mezi názvem a hodnotou je znak *: dvojtečka*. Tyto páry jsou od sebe oddělovány znakem *, čárka*. Objekt je následně neuspořádaná množina párů název/hodnota, kdy lze objekty řetězit do pole pomocí hranatých závorek, kdy jsou jednotlivé objekty odděleny znakem *, čárka*.[25]

```
{
    "solutions": [
        { "id":"AIM_CLE", "port": 2508 },
        { "id":"AIM_CLO", "port": 8700 },
        { "id":"AIM_EDI", "port": 6445 }
    ]
}
```

Listing 3.1: Definice pole s objekty pomocí JSON

3.2.2 AWS Lambda

Jedná se o bezserverové běhové prostředí daného programovacího jazyka v cloudu. Je to tedy takový kontejner pro funkci, který běží jen v případě, že byla vyvolána událost, kterou má obsloužit. Když žádná taková událost nenastane v dalších patnácti minutách od předchozí, funkce se vypne. Za spuštění AWS Lambdy se platí podle doby, po kterou běží a vykonává obsluhu události, výše účtované ceny za jednotku času (násobky 100 milisekund) je poté závislá také na objemu operační paměti, která je dané funkci nastavena. Dává nám tedy možnosti jak spouštět daný kód bez nutnosti udržovat běžící servery nebo infrastrukturu. AWS Lambda se postará o dostatečné naškálování potřebných zdrojů, které jsou na jednotlivých funkcích nastaveny, aby se provedlo obsloužení události. Tudíž není možno se na daný server, kde jsou funkce spouštěny, připojit a provádět změny na operačních systémech. Jelikož po nastartování lambda funkce je její běhové prostředí aktivní až po dobu patnácti minut, jejich nejlepší využití je v případech, kdy je potřeba za velmi krátkou dobu zpracovat větší objem dat, a tyto akce jsou vyvolávány eventy s delšími časovými rozestupy, a tudíž se nejedná o konstantní zátěž.

Lambda funkce nabízejí podporu pro několik programovacích jazyků (kompilovaných i interpretovaných):

- Java - podpora Java 11 (Correto) či Java 8
- .NET 6 - podpora C# a PowerShell
- .NET Core 3.1 - podpora C# a PowerShell
- Go 1.x
- Node.js 14.x - případně starší verze 12.x
- Python 3.9 - případně starší verze 3.6, 3.7 a 3.8
- Ruby 2.7

U interpretovaných jazyků je možnost úpravy kódu přímo v AWS konzoli přes GUI rozhraní, kde je možné kód debugovat. Pro snadnější práci s knihovnami je zde podpora pro jejich sdílení, kdy je možno knihovnu nahrát do takzvané *vrstvy*, kterou je poté možno přidávat k jednotlivým funkcím, a je vždy přidána do běhového prostředí při jejich spuštění. Je to velice snadný způsob sdílení stejného kódu pro vícero funkcí. V grafickém rozhraní webové konzole je možno u každé funkce jednoduše zjistit, jaké události jsou na danou funkci napojeny či případně dodefinovávat nové, při kterých se má funkce spouštět. Nejčastější způsob spuštění lambda funkce

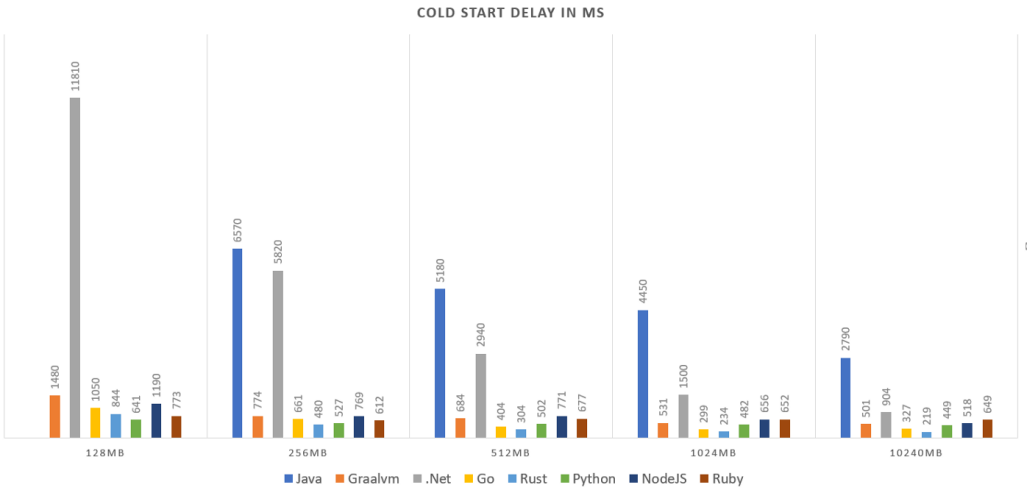
je napojení eventu na Api Gateway, která se postará o předání požadavku a zahájí inicializaci lambda funkce. Dále je zde možnost jednotlivé funkce verzovat pomocí interního systému, kdy je pro každou verzi uchovávána příslušná verze kódu a jednotlivých nastavení a následně je možno tyto verze přiřazovat k jednotlivým *aliasům* funkce, které je možno využít k rozdílným nastavením například proměnných prostředí či možnosti spustit naráz několik různých verzí dané funkce, pokud došlo k nekompatibilní změně na rozhraní a je nutno podporovat starší verze. Na každý takovýto vytvořený alias jde navázat jinou verzi funkce.

Výkonnost

Výkonnost lambda funkcí je závislá jak na nastavení přiřazené paměti RAM, jelikož podle velikosti je každé funkci vypočítávána interním algoritmem doba času CPU, který dostane, tak na vybraném programovacím jazyce. Dále záleží, zda se jedná o takzvaný *cold start*, což je kompletní nastartování dané funkce, či o *warm start*, kdy je daná funkce již nastartována a jenom obslouží další příchozí požadavek. Na měření výkonnosti můžeme tedy nahlížet z vícero pohledů a jejich případných kombinací, zároveň je vždy také potřeba brát v úvahu, jak bude nasazená funkce využívána (jaké dotazy bude obsluhovat a v jakém měřítku budou danou funkci volat).

Pokud je lambda funkce využívána nárazově, kdy je mezi jednotlivými příchozími událostmi časové okno v délce pěti minut a výše, je velice pravděpodobné, že bude často docházet ke studeným startům. Tyto starty jsou velice nepřívětivé pro kompilované jazyky jako je Java či C#, kdy je možno při velice nízkém nastavení paměti pozorovat startovací časy až 6 sekund. Kdežto u interpretovaných jazyků jako je Python, NodeJS či Go lze i při nastavení malé paměti pozorovat startovací časy v řádech milisekund. Zvýšením přiřazené paměti lambda funkci u jazyků Java či C# je možno docílit zkrácení studeného startu za cenu vyšších nákladů za jednotlivá volání funkce. Zde lze pozorovat, že při zvyšující se velikosti paměti RAM dochází k snížení časů studeného startu o sekundy, nicméně se tyto jazyky nedostanou na časy, kdy by celý studený start trval v řádech milisekund viz. obrázek 3.2. Na obrázku chybí záznam pro Javu s nastavením paměti na 128Mb, jelikož při takto nízké hodnotě nedojde k jejímu nastartování.

Takzvaný teplý start funkce je brán v potaz, když obsluhované akce přicházejí v intervalu, za který se daná funkce nestihne uspat, a tudíž je stále nastartované prostředí, ve kterém běží. U těchto startů lze pozorovat velice podobné časy vykonávání při nastavení podobné minimální velikosti paměti RAM lambda funkce. Z podporovaných jazyků zde nejhůře vychází Node.js,



Obrázek 3.2: Studený start lambda funkce v závislosti na programovacím jazyce a velikosti RAM. Dostupné z URL [10].

u kterého nedochází k výraznému zlepšení ani po několika tisících vyvolaných událostí.

Dále je možno optimalizovat jednotlivé funkce snižováním případných závislostí na knihovnách třetích stran, kdy se potvrzuje pravidlo, že čím menší je balíček spouštěných funkcí, tím kratší je následně doba, kterou potřebuje funkce ke startu. Balíčky s velkým množstvím závislostí mohou být pomalejší až 7x [15]. Sama AWS konzole upozorňuje uživatele při nahrávání obslužného kódu funkce, že pokud je větší než 10MB, je doporučeno jeho nahrání na úložiště S3 a z funkce se na umístění odkazovat.

3.2.3 Amazon DynamoDB

DynamoDB je dokumentová NoSQL databáze vyvíjená samotným Amazonem, která rozšiřuje jejich nabídku cloudových služeb v AWS. NoSQL databáze jsou aktuálně rychle se rozšiřující verzí tradičních relačních databází (RDBMS). Jejich masivní rozšíření je podpořeno díky snadnému vertikálnímu či horizontálnímu škálování, které je u tradičních relačních databázi často finančně náročnější. Oproti tomu NoSQL databáze jako je DynamoDB umožňují snadné vertikální i horizontální škálování. V prostředí AWS jsou tyto databáze uchovávány na SSD discích a škálovány pomocí počtu instancí. Zároveň jsou data pro zajištění vysoké dostupnosti automaticky replikována do vícero zón dostupnosti v rámci jednoho regionu [18]. Stále je ale nutno myslet na každý specifický případ použití a nelze globálně říci, že NoSQL databáze jsou nyní lepší než klasické RDBMS. Vždy záleží na povaze dat,

která jsou v systémech uchovávána, a zároveň také na dané aplikaci, která data využívá pro svůj chod. NoSQL databáze jsou vhodné pro ukládání dat, ke kterým se přistupuje ve velké části podle primárního klíče a nejsou potřeba složité dotazy s kombinací filtrování nad jednotlivými atributy. Jsou designovány tak, aby umožňovaly eliminaci co nejvíce komplexních spojování dat mezi tabulkami. To umožňuje ukládat data z tabulek relačních databází v denormalizovaném databázovém modelu do kolekcí nebo přímo do tabulek bez schématu, jelikož většina dnešních NoSQL databází ukládá data ve formě JSONu. Je nicméně nutné stále myslet na omezení v podobě maximální velikosti jednoho záznamu, který nesmí překročit 400KB. Tento limit je oproti jiným NoSQL databázím několikanásobně menší (největší dosažené velikosti uloženého záznamu jsou pro MongoDB až 16MB a pro Cassandra až 16GB), ale má svůj důvod. Tato velikost by měla být dostačující pro většinu případů užití a zároveň by měla co nejvíce eliminovat případné anti vzory a vést ke správné tvorbě modelu [17]. Dotazy do DynamoDB jsou poté prováděny pomocí veřejného API, které umožňuje veškeré CRUD (Create, Read, Update a Delete) operace se záznamy a dotazování se nad nimi pomocí operací *Query* nebo *Scan*. Doporučeným přístupem pro načítání dat je použití *Querry*, kdy je nicméně potřeba vytvořit dotaz z primárním klíčem. Pokud bychom tedy chtěli dostat všechny záznamy z tabulky, musíme iterovat přes všechny primární klíče, tento postup zjednodušuje operace *Scan*, která načte všechny záznamy v tabulce. Jelikož je cenová politika DynamoDB postavena také na takzvaných *read a write capacity units*, které jsou konzumovány při každém volání API DynamoDB a jejich potřebný počet pro provedení operace se odvíjí od velikosti přenášených dat, může být operace *Scan* velice drahá, jelikož dojde ke spotřebování velkého množství *read units*, a zároveň časově náročná. Tudíž je doporučována jen v nutných případech a nad menšími tabulkami. Všechna ukládaná data jsou zpřístupněna jen autentikovaným uživatelům, kterým jsou práva přidělována pomocí AWS Identity and Access Management, a zároveň jsou použity nejmodernější kryptografické techniky pro jejich šifrování a zobrazování. Databáze byla zvolena také z důvodu serverless přístupu k celému systému, kdy je v Lambda funkcích velmi neefektivní používání jakýchkoliv ORM frameworků pro práci s databázovou vrstvou. Tyto frameworky potřebují navazovat spojení s databází, které si většinou udržují pro delší dobu a tudíž je nutno myslet na nutnost tyto databázové konekce vždy uzavírat před vypnutím lambdy, jelikož by mohlo velice snadno dojít k vyčerpání maximálního počtu konekcí k databázi, kde je jejich maximální počet u každé RDS provozované v cloudu AWS pevně definován. Tento problém s využitím DynamoDB odpadá, jelikož celá komunikace probíhá pomocí volání jednoduchého API, které je

schopno zpracovávat i velké objemy dat v jednotkách milisekund. Datové typy jednotlivých atributů, které DynamoDB podporuje, lze rozdělit do tří hlavních kategorií [29]:

- skalární hodnoty - jedná se především jednoduché hodnoty, které jsou ukládány do typů **string**, **number**, **binary**, **boolean** nebo hodnota **null**
- komplexní typy - jsou to flexibilnější typy atributů, které jsou schopny uchovávat potřebné elementy. Jedná se o nejflexibilnější řešení jak ukládat libovolně vnořené atributy pomocí **list** a **map** datových struktur.
- sety - jsou to výkonné složené typy, které představují více jedinečných hodnot. V DynamoDB jsou sety velice podobné datovým strukturám *Set* ve vyšších programovacích jazycích. Použití tohoto typu atributu je vhodné především v případech, kdy je potřeba sledovat jedinečnost v určité doméně. Jejich jedinou podmínkou je, že každý prvek v setu musí mít stejného datového typu **string set**, **number set** nebo **binary set**.

3.2.4 Amazon Cognito

Jedná se o AWS službu, která poskytuje jednoduchou správu uživatelů a jejich přístupů. Díky využívání této služby je programátorovi usnadněn vývoj, jelikož se nemusí starat o koncové ukládání uživatelských dat, a není tedy nutné pro tyto potřeby připravovat vlastní řešení a nastavovat příslušnou infrastrukturu. Služba nabízí komplexní API pro klíčové prvky správy uživatelů jako je autentizace[3], autorizace[4] a spravování jednotlivých uživatelských úložišť, které jsou potřebné pro připojované aplikace. Díky standardizovanému rozhraní je možno zkombinovat tuto službu s přihlašováním pomocí sociálních sítí jako je například Facebook, Google nebo Apple či jiných poskytovatelů identit, kteří podporují některé ze standardizovaných rozhraní. Podporovány jsou následující: OpenID Connect, OAuth 2.0 a SAML 2.0.

Při využití kompletního přihlašování pomocí Cognito je integrace s aplikací jednoduchá díky široké škále podporovaných programovacích jazyků, pro které jsou vydávány vývojářské sady (SDK). Mezi podporovanými platformami se nachází Android i iOS a udržována je také knihovna pro JavaScript. Tyto nástroje umožňují vývojáři jednoduše volat API rozhraní Cognito a implementovat pomocí nich kompletní proces registrace a přihlašování jednotlivých uživatelů. Jako výhodu tohoto řešení lze brát i existenci

[3] Autentizace - jedná se o proces ověření proklamované identity

[4] Autorizace - dává uživateli oprávnění pro provádění určité činnosti

výchozího nastavení přihlašovací obrazovky pro tyto dvě akce, kterou lze uživatelsky přizpůsobit pro své potřeby pomocí grafického rozhraní AWS konzole.[6]

Základní stavební části služby jsou:

- user pools (skupiny uživatelů) - jedná se o hlavní úložiště celé služby, kde jsou ukládána data o uživatelských profilech. Při každé nové registraci je příslušná skupina uživatelů aktualizována o nového uživatele a následné přihlašování do aplikací je ověřováno vůči uloženým informacím v systému. Pod jedním AWS účtem lze mít až 1000 oddělených skupin uživatelů pro vícero aplikací.

- identity pools (skupiny identit) - slouží k vydávání dočasných přihlášení k dalším službám AWS ověřeným uživatelům. Jedná se o SSO pro zbylé AWS služby, aby uživatelé nemuseli zadávat opakovaně své přihlašovací údaje. Jedná se o část Cognita, která se stará o autorizaci.

Mezi hlavní nevýhodu Cognita lze ale jistě zařadit jeho horší nastavitelnost v případě, že je potřeba dodělávat dodatečné změny po vytvoření uživatelské skupiny. V této fázi již není možno uživatelům přidávat žádné nové atributy či měnit možnosti přihlašování. Zároveň je zde ve velkých aplikacích problém vytvářet multitenantní prostředí, kdy by bylo potřeba pro každého zákazníka vytvořit oddělenou skupinu uživatelů, tato skutečnost je v infrastruktuře aimtec.cloud vyřešena pomocí Keycloak viz. 3.2.5. Přiřazení uživatelů ke skupinám v Cognitu umožňuje určovat přístupy, kam se smí zákazník přihlašovat. Pokud ale vyvíjíme aplikace čistě na cloudové infrastruktuře AWS, je zde několik pozitiv, které by měl programátor zvážit. Cognito podporuje mnoho integrací s Lambda funkcemi, má velice dobré knihovny pro práci s mobilními zařízeními a při využívání v podnikovém prostředí může sloužit jako centrální bod přidělování dočasných přihlašovacích údajů pro další služby v AWS. Jeho výhodou je také jednoduchá integrace do API Gateway, kdy lze ve webovém rozhraní AWS konzole nadefinovat autorizaci requestů pomocí Cognita.

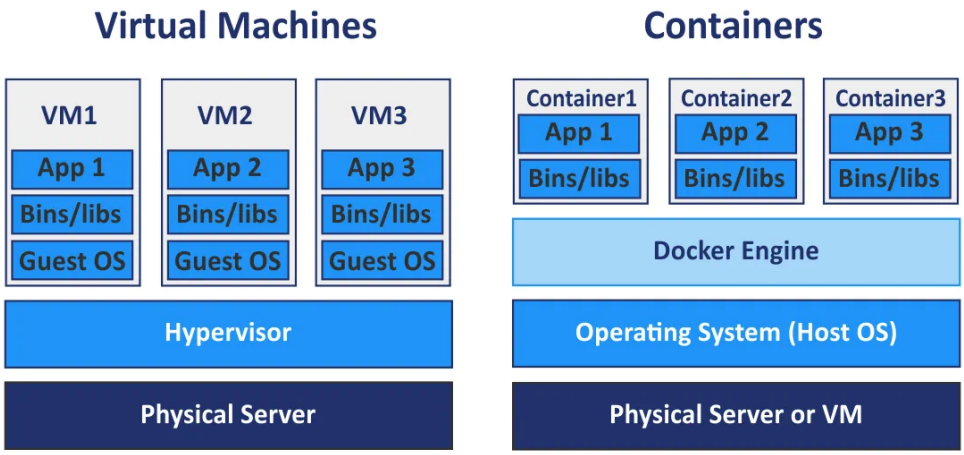
3.2.5 Amazon EKS

Amazon Elastic Kubernetes Service je služba umožňující provozovat Kubernetes v cloudové infrastruktuře AWS bez nutnosti instalovat a konfigurovat vlastní Kubernetes nody. Umožňuje tedy provozovat open source systém Kubernetes, který automatizuje nasazování aplikací, jejich škálování a poskytuje rozhraní pro jednoduchou správu díky jejich kontejnerizaci pomocí

Dockeru. Amazon EKS běží na nejnovějších stabilních verzích Kubernetes a podporuje aplikace, které běží na jakýchkoliv jiných standardních Kubernetes prostředích, je tedy velice snadné své aktuální řešení přenést do cloudu bez jakékoliv změny v kódu či nastavení sestavování.

Docker

Jedná se o open source projekt dostupný pod licencí Apache 2.0[5]. Díky Dockeru lze vytvářet kontejnery, které obsahují námi vyvíjenou aplikaci v izolovaném prostředí od zbytku systému a ostatních kontejnerů s dalšími aplikacemi. Tato kontejnerizace aplikací nám umožňuje snadné automatické nasazování aplikace a její bezproblémové přenášení mezi jednotlivými systémy, které Docker kontejnery podporují. Díky izolaci dané aplikace se dá říci, že se jedná o virtualizaci, kdy každý jeden kontejner obsahuje vše potřebné pro běh aplikace uvnitř. Rozdíl kontejnerizace oproti standardnímu použití virtuálních strojů je ten, že nám odpadají starosti o operační systémy jednotlivých virtuálních strojů, které je potřeba udržovat a aktualizovat viz obrázek 3.3.



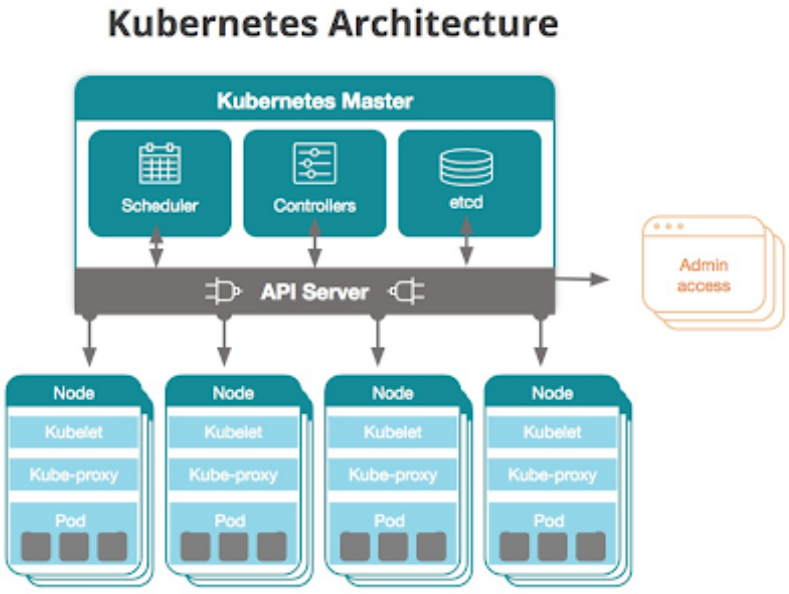
Obrázek 3.3: Rozdíl kontejnerizace oproti využití virtuálních strojů. Dostupné z URL [27].

Kubernetes

Ke správě kontejnerů, které mohou běžet až na několika desítkách serverů najednou, se využívají takzvané **orchestrátory**. Tyto orchestrátory jsou

[5]Apache 2.0 - svobodná softwarová licence, jejímž autorem je Apache Software Foundaton. Zdroj: `https://www.apache.org/licenses/LICENSE-2.0.html`

software, který se automaticky podle zavedených definic stará o zavádění, škálování a především údržbu kontejnerizovaných aplikací. Díky těmto nástrojům je možné automaticky mazat kontejnery, které již nejsou potřeba, jelikož se snížila zátěž a aplikace již nemusí být spuštěna v několika instancích. Je možno je využít i pro nasazování nových verzí, kdy je díky nim dosaženo kontinuální dodávky služeb. Toto lze dosáhnout jednoduchým nastavením, že při nasazení novější verze či při potřebě restartu aplikace je starý kontejner smazán až v případě, že nově vytvořený kompletně naběhne a je dostupný pro uživatele. Nejznámějším z takovýchto orchestrátorů je open source projekt Kubernetes. Vývoj tohoto nástroje začal pod Googlem, nicméně byl v roce 2014 zpřístupněn veřejnosti a nyní je volně dostupný pod záštitou Cloud Native Computing Foundation. Od té doby se stal hojně používaným nástrojem především pro programátory, kteří vyvíjejí aplikace s architekturou mikroslužeb.



Obrázek 3.4: Kubernetes architektura. Dostupné z URL [35].

3.3 Keycloak

Keycloak je otevřený softwarový produkt, který umožňuje do aplikací implementovat jednotný způsob přihlašování pomocí správy identit. V rámci aimtec.cloud infrastruktury je Keycloak využíván primárně pro aplikace DCIx.

Do aimtec.cloud Portálu byl Keycloak implementován převážně kvůli jednotnému přihlašování, kdy po přihlášení do portálu není po uživateli vyžadováno opětovné zadávání hesla při přístupu do aplikace DCIx, která má podporu pro SSO. Je využíván také kvůli zajištění multitenantnosti aplikace DCIx, jelikož možnosti Cognita na toto nejsou ideální. Díky snadnému napojení na Cognito přes GUI rozhraní Keycloak je zde ověřováno, zda je přihlášený uživatel v Cognitu zařazen do příslušné skupiny uživatelů, kteří mají mít k aplikaci přístup, či je uživatel již přihlášen. Tohoto lze docílit nastavením příslušné instance Cognito uživatelské skupiny a identifikátorem dané skupiny, do které musí být příslušný uživatel přiřazen. Na Keycloaku je poté řešena multitenantnost jednotlivých aplikací.

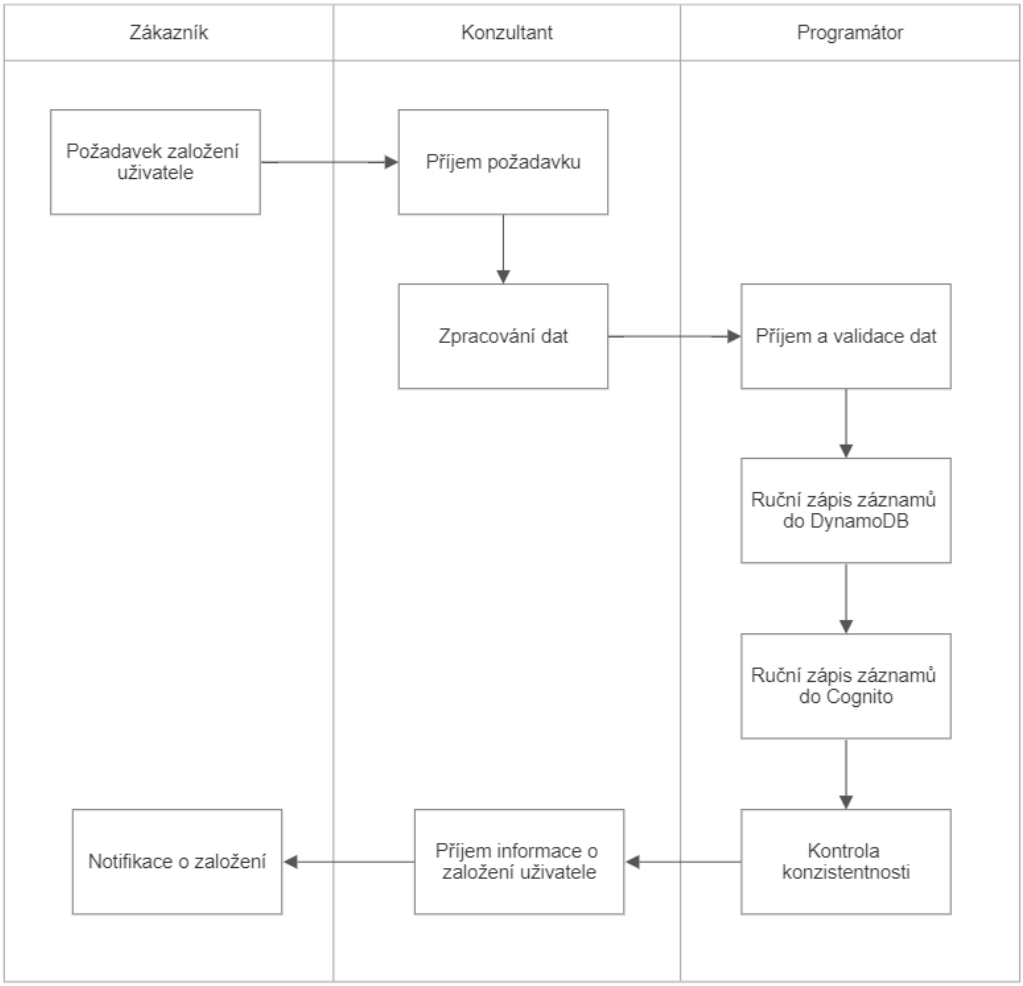
4 Business procesy správy uživatelů a aplikací

Celý proces správy běžících aplikací v cloudu a jejich uživatelů je v současné době velmi složitý a zdlouhavý. Požadavek na změnu je nyní potřeba komunikovat přes několik zaměstnanců napříč odděleními ve firmě od supportního oddělení, přes konzultanty až k programátorům, kteří tyto změny aktuálně provádějí. To vede k celkovému zpomalení procesu. S přihlédnutím na proces iterativního vývoje ve firmě je zároveň velmi obtížné předem nenaplánované požadavky zpracovávat v průběhu již běžící iterace, kdy má každý programátor dané úkoly, které vyplňují jeho celou kapacitu. Pro vyřizování těchto požadavků je tedy aktuálně zvolen vždy příslušný programátor, který si v iteraci drží vyhrazený čas, ve kterém zásahy provádí. Zároveň je zde také veliká režijní zátěž v podobě přeposílání mailů či komunikace prostřednictvím jiných oficiálních kanálů firmy mezi konzultanty a programátorem, který je danou iteraci určen k zpracovávání požadavků, jelikož k administraci není připraveno žádné uživatelsky přívětivé rozhraní a pro jejich provádění je potřeba odborně zaškolených programátorů. Změny dat využívaných v administraci jsou prováděny pomocí přímých zápisů do systémů, a je tedy potřeba dbát na dodržování konzistentního stavu mezi nimi. Proto je nežádoucí, aby v tomto stavu způsobu administrace byly kompetence správy uživatelů a aplikací delegovány na vícero osob v aktuálním řetězci. Tyto osoby by bylo potřeba vždy vyškolit a zároveň dohlížet na správné zadávání dat, které nenaruší konzistentnost v systémech. Jak již bylo naznačeno, data pro kompletní multitenantní systém jsou uložena ve dvou oddělených systémech, kdy každý uchovává potřebnou část dat. Jedná se o AWS Cognito a AWS DynamoDB, kdy jsou záznamy v těchto jednotlivých systémech navzájem propojeny pomocí primárního klíče, kterým je přihlašovací email uživatele.

4.1 Založení uživatele

Správa uživatelů probíhá v oddělených systémech Amazon Cognito a DynamoDB, kde jsou uživatelé propojeni pomocí unikátního emailu. Tento jednoznačný identifikátor uživatele je potřeba do obou systémů zadávat ve stejném formátu, kterým je jeho kompletní převedení na malá písmena a

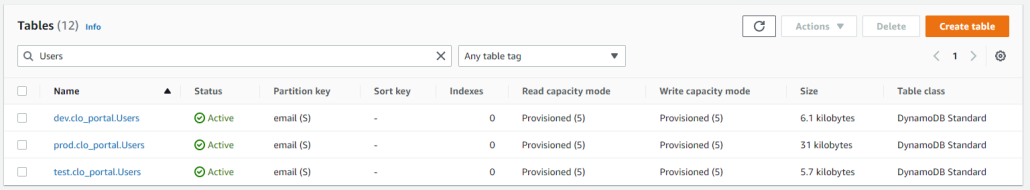
splnění základního patternu pro emaily. Seznam daných uživatelů, kteří se mají v systému založit, dostává programátor od konzultanta. Tento seznam je následně potřeba ručně zadat do systémů pomocí jejich grafického uživatelského rozhraní. K provedení tohoto kroku je potřeba, aby měl daný programátor od IT oddělení firmy nastavená příslušná práva v AWS konzoli. Po přihlášení do konzole je potřeba, aby programátor zanesl nové uživatele do obou systémů zároveň. Na následujícím obrázku 4.1 je zobrazen diagram představující proces založení nového uživatele.



Obrázek 4.1: Proces založení nového uživatele v aimtec.cloud.

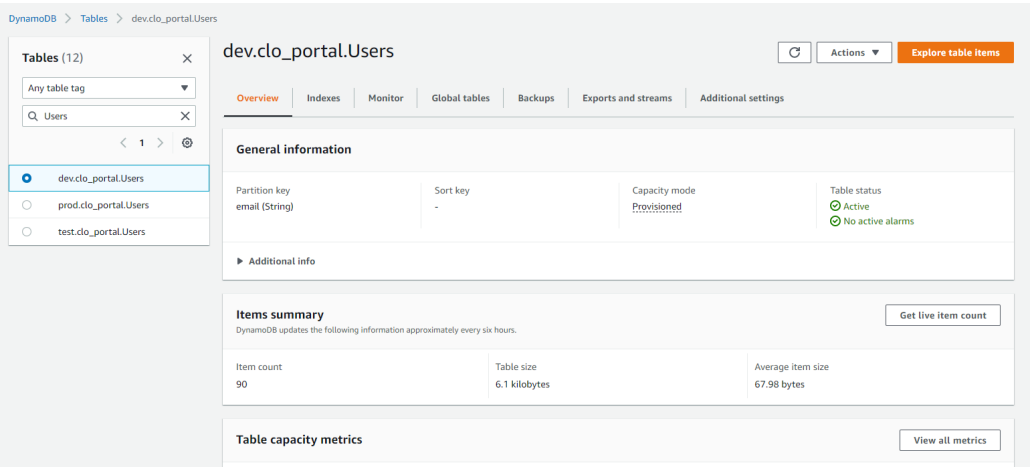
V databázi DynamoDB jsou k jednotlivým uživatelům uchovávány dodatečné informace, které jsou používány v již existujícím grafickém uživatelském rozhraní aimtec.cloud Portálu. Správa záznamů v tomto systému probíhá pomocí webové AWS konzole, která má přívětivé uživatelské rozhraní. Po zvolení služby DynamoDB je zobrazen seznam všech tabulek, které

jsou založeny. Pro práci s tabulkami je potřeba znalostí jmenných konvencí, které jsou zde zavedeny. Jelikož DynamoDB nepodporuje tvorbu schémat, jsou zde tabulky pro jednotlivá prostředí vytvořena s daným prefixem *dev*, *prod* a *test* po dané prostředí viz. obrázek 4.2 a suffix názvu reprezentuje název ukládaných záznamů v množném čísle. Pro provádění změn či zaklá-



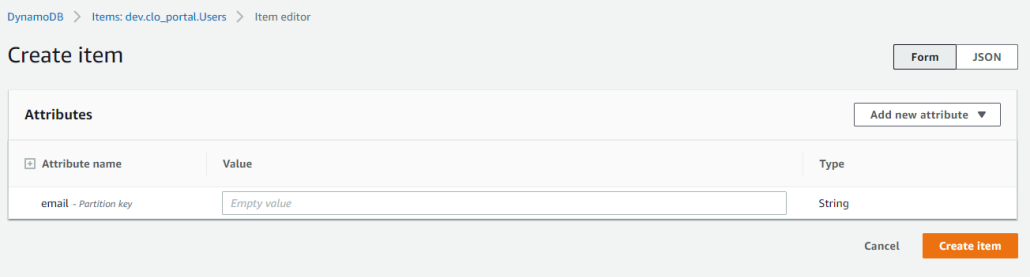
Obrázek 4.2: Grafické rozhraní DynamoDB s přehledem tabulek.

dání nových záznamů je tedy potřeba vědět, do jakého z daných prostředí se mají data zanést. Zakládání a změny uživatelů se provádějí v tabulce **Users**. Do správy této tabulky se uživatel dostane proklikem z názvu tabulky příslušného prostředí, kdy je mu zobrazeno základní nastavení jako je počet a celková velikost aktuálních záznamu či název a typ primárního klíče viz. obrázek 4.3. Z tohoto přehledu se lze následně dostat na obrazovku s výčtem



Obrázek 4.3: Grafické rozhraní DynamoDB s nastavením vybrané tabulky.

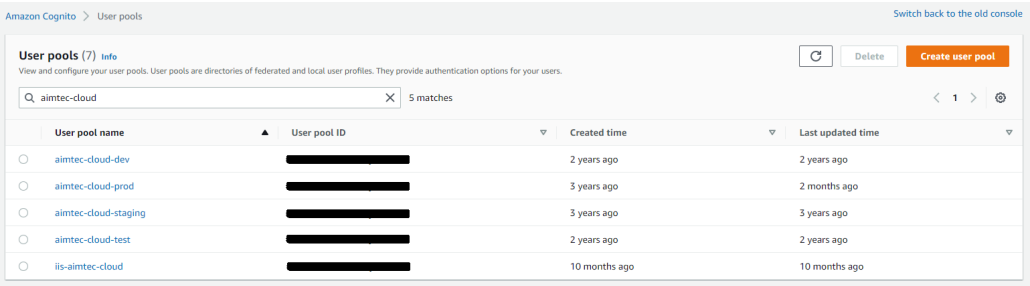
jednotlivých záznamu, ze které je poté potřeba provádět jednotlivé úpravy na uživatelích, či zakládání nových záznamů. Vytváření nového záznamu s sebou nese jisté úskalí, kdy je v grafickém rozhraní nového vytvářeného záznamu zobrazen jen primární klíč tabulky viz. obrázek 4.4. V zakládacím okně již ale nejsou bohužel zobrazeny případné další atributy, které se v tabulce na jednotlivých záznamech vyskytují. Proto je nutné, aby uživatel,



Obrázek 4.4: Grafické rozhraní DynamoDB pro zakládání nového záznamu.

provádějící vytváření nového záznamu, věděl, jaké další atributy se správnými datovými typy je potřeba k záznamu přidat, aby celý systém, který čerpá data z této tabulky, nezhavaroval. Jedním z takovýchto atributů je pole stringů *companies*, kde jsou uchovávány velkými písmeny třípísmenné zkratky zákazníků, ke kterým daný uživatel patří. Tento proces je zdlouhavý také kvůli nemožnosti založení více záznamů najednou a při požadavku založení několika nových uživatelů musí programátor vytvářet každého zvlášť pomocí otevření dialogového okna pro založení záznamu. Poté, co jsou provedeny všechny potřebné úkony v AWS DynamoDB, je potřeba data vložit i do Amazon Cognito.

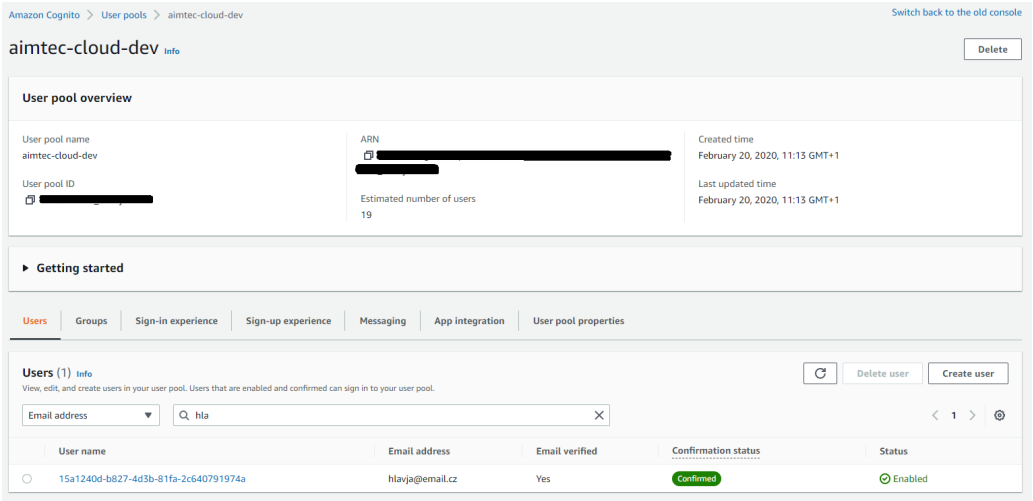
Druhým systémem využívaným pro správu uživatelů a jejich přístupů je Amazon Cognito. Tento systém je využíván primárně pro řízení přístupů daného uživatele k jednotlivým aplikacím, které jsou na Cognito napojeny pomocí Keycloaku. Správa v Cognito opět probíhá pomocí grafického uživatelského rozhraní této služby v AWS konzoli. Po přihlášení do konzole a zobrazení této služby je uživateli zobrazeno rozhraní obsahující veškeré uživatelské skupiny, které může přihlášený uživatel spravovat, viz. obrázek 4.5. Názvy těchto skupin jsou tvořeny podle definovaných jmenných konvencí



Obrázek 4.5: Grafické rozhraní Amazon Cognito s přehledem uživatelských skupin.

IT oddělení, které se o Cognito jako službu stará. Správa záznamů uvnitř

těchto skupin je již na DCI oddělení. Suffix názvu jednotlivých skupin odpovídá prostředí, které je na záznamy dané skupiny připojeno. Po rozkliknutí příslušné uživatelské skupiny, jsou programátorovi zobrazeny informace o nastavení dané skupiny a v nové verzi AWS konzole, která byla pro pořizování těchto záznamů obrazovek použita, jsou na této stránce zobrazeny v dolní části i záložky, kde lze zobrazit již existující uživatele viz. obrázek 4.6. Nového uživatele programátor vytváří pomocí tlačítka **Create user** na



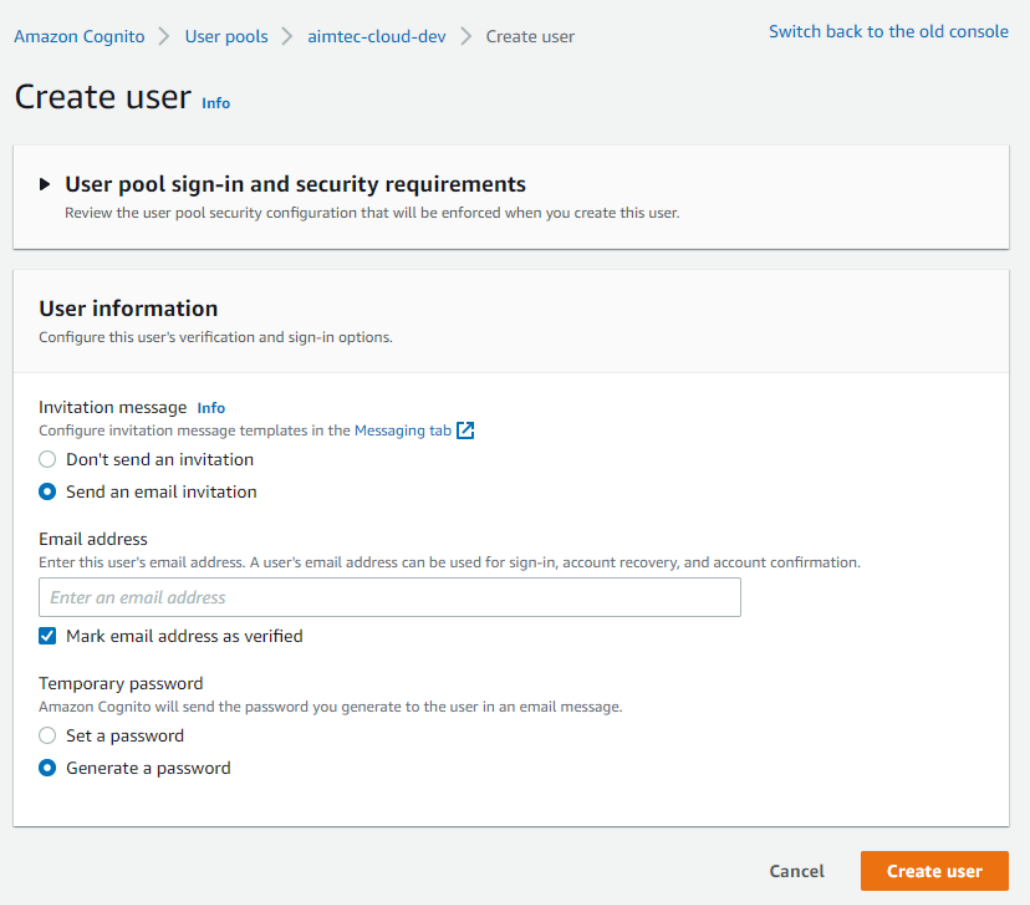
Obrázek 4.6: Grafické rozhraní uživatelské skupiny v Amazon Cognito.

záložce s uživateli. Toto tlačítko přesměruje programátora na novou stránku, kde je připraven formulář, přes který se uživatel zakládá viz. obrázek 4.7. V tomto formuláři je nutno přenastavit výchozí parametry formuláře do stavu, který je vidět na obrázku. Jedná se především o zaslání uvítacího emailu novému uživateli a také o nastavení automaticky vygenerovaného prvního hesla, které je v uvítacím emailu odesíláno a uživatel si jej při prvním přihlášení mění.

Po úspěšném založení uživatele se stejným identifikačním emailem do Amazon Cognita i AWS DynamoDB je proces založení nového uživatele úspěšný. V případě nezdaru založení záznamu do jednoho z dříve zmíněných systémů nastává nekonzistentní stav a je potřeba data opravit. Tato oprava se může provést odmazáním nadbytečného záznamu, či vyřešením chyby při zakládání.

4.2 Založení aplikace

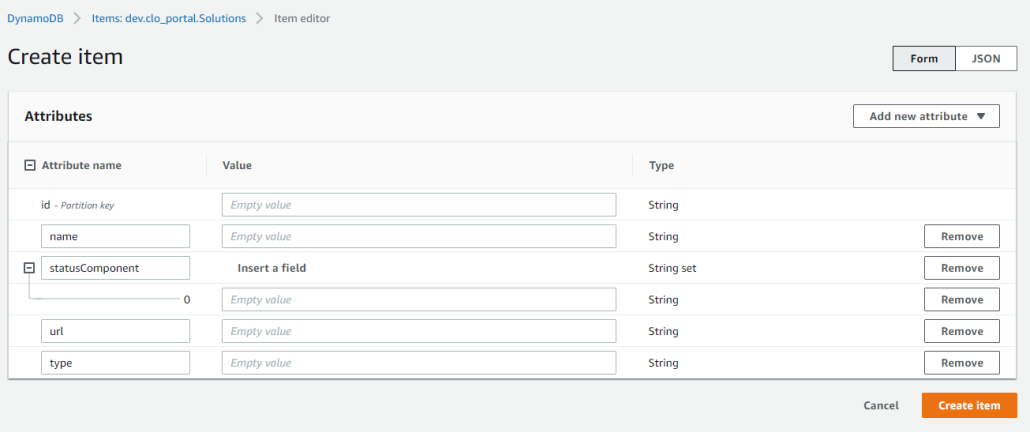
Správa aplikací a přístupů k nim opět probíhá v obou již výše zmíněných systémech. Jedná se o obdobný proces jako při zakládání uživatele. Založení



Obrázek 4.7: Grafické rozhraní vytváření uživatele v Amazon Cognito.

aplikace je zde inicializováno zprávou od IT oddělení firmy, které programátorovi sdělí název aplikace, prostředí v jakém běží a url adresu, kde je dostupná.

Následně programátor provede založení záznamu aplikace do DynamoDB, kdy vybere tabulku **Solutions** se správným prefixem daného prostředí. Do nového záznamu aplikace je opět nutno vyplnit několik atributů, ale stejně jako u zakládání uživatele je zde zobrazen pouze primární klíč záznamu. K tomuto klíči je potřeba přidat další atributy viz. obrázek 4.8, které je potřeba dohledat z dokumentace. Id aplikace je tvořeno definovanou jmennou konvencí, kdy se skládá z velkých písmen např. AIM_CPJ. První tři velká písmena reprezentují zkratku zákazníka, následuje oddělovač ve formě podtržítka a zbylé tři písmena reprezentují identifikátor aplikace. První písmeno tohoto identifikátoru je statický znak C, následovaný znakem reprezentujícím typ prostředí (P - produkční, T - testovací, D - vývojové) a posledním znakem je udávaný obor aplikace (Y - Yard, W - WMS, J - JIT, P - Portal).

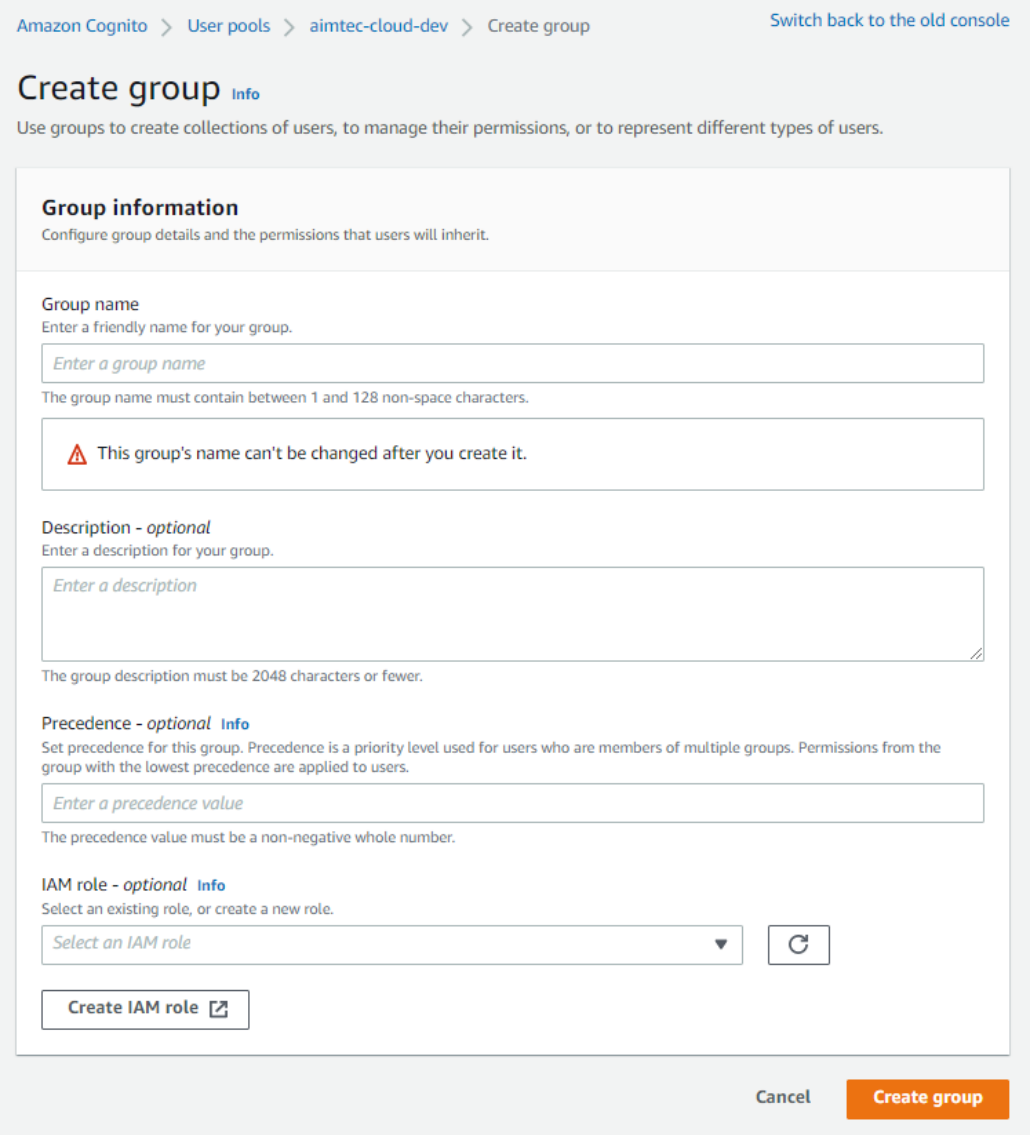


Obrázek 4.8: Grafické rozhraní vytváření záznamu aplikace v DynamoDB.

Do atributy *type* se udává zkratka provozované aplikace, která je aktuálně jen jedna a to DCI. Po založení aplikace v této tabulce je potřeba založit i záznam v Amazon Cognito, který bude sloužit k přidělování oprávnění k přístupu do dané aplikace. Toto oprávnění je uživateli nastavováno pomocí přiřazení do skupiny. Vytvoření této skupiny se opět provádí v dané uživatelské skupině pro příslušné prostředí. Programátor obsluhující zákaznické požadavky pracuje převážně s produkčním prostředím. Po rozkliknutí prostředí se programátorovi opět zobrazí přehled nastavení, kde je možno v dolní části obrazovky přepnou na záložku *Groups*, ve které se zobrazí tlačítko **Create group**. Toto tlačítko přesměruje programátora na novou stránku, kde je připraven formulář, přes který se zakládá nová skupina, viz. obrázek 4.9. Formulář obsahuje několik vstupních polí, nicméně programátor nevyplňuje všechna. Vyplnit je nutno *Group name*, do kterého je třeba doplnit stejnou hodnotu, jako nese atribut id v DynamoDB záznamu, toto id je doplněno o suffix *_access*. Do *Description* pole je vyplněn název aplikace, který odpovídá atributu *name* záznamu v DynamoDB. Po úspěšném uložení nové skupiny v Cognitu je proces tvorby záznamu pro novou aplikaci hotov. Při selhání zakládání záznamu v jednom ze systémů dochází opět k nekonzistentnímu stavu a je potřeba záznamy odmazat či vyřešit problém se zakládáním.

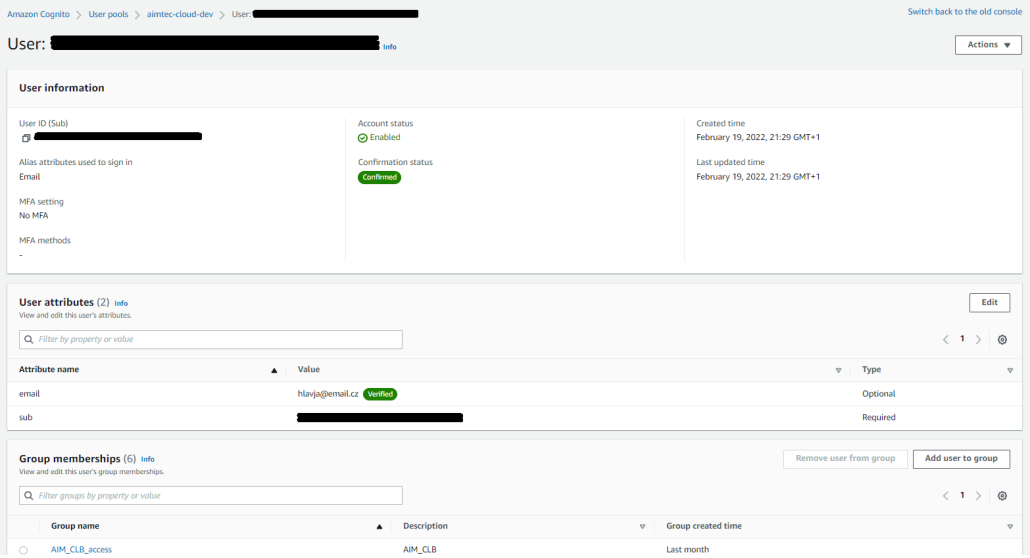
4.3 Správa přístupů

Správa přístupů k aplikacím jednotlivých uživatelů je opět prováděna simultánně v obou systémech pomocí AWS konzole jako tomu je při zakládání záznamů. Přístupy do aplikací jsou uživatelům přidělovány v Amazon Co-

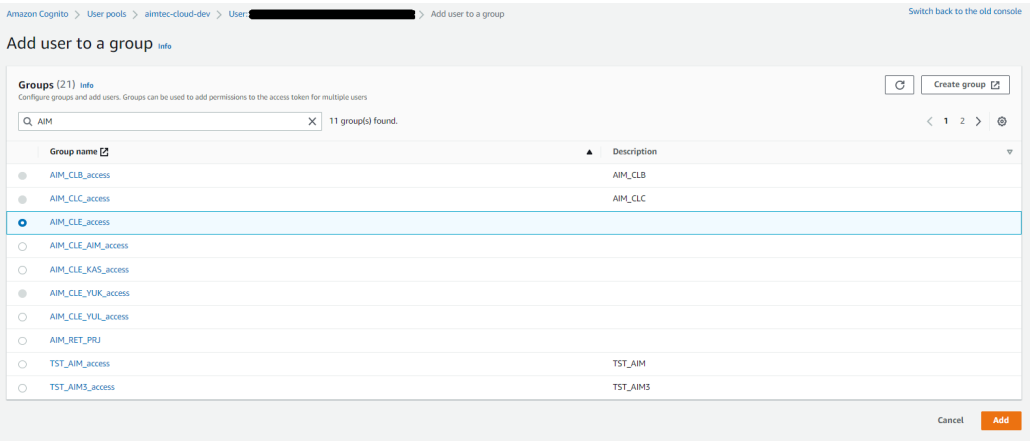


Obrázek 4.9: Grafické rozhraní vytváření nové skupiny v Amazon Cognito.

gnito, kdy je opět po otevření správné skupiny uživatelů nutno otevřít detail uživatele, kterému chceme přístupy upravovat. Detail uživatele zobrazuje na hlavní obrazovce jednoznačné identifikátory a zároveň i seznam skupin, do kterých je uživatel přiřazen, viz. obrázek 4.10. Pro přiřazení uživatele do skupiny je nutno zmáčknout **Add user to group**, které programátora přesměruje na obrazovku se založenými skupinami, ze kterých lze jednu vybrat a do té uživatele přiřadit vit. obrázek 4.11. Předností této obrazovky je možnost existující skupiny filtrovat a najít rychle záznam požadované. Nevýhodou této obrazovky je absence hromadného přiřazování skupin, je potřeba přiřazovat každou skupinu zvlášť. To je velice neefektivní např. v případě, kdy



Obrázek 4.10: Grafické rozhraní uživatelského profilu v Amazon Cognito.



Obrázek 4.11: Grafické rozhraní přiřazení uživatele do skupiny v Amazon Cognito.

chceme přidělit přístupy konzultantovi do více aplikací. Po úspěšném přiřazení uživatele do skupiny je zajištěna jeho možnost přihlášení do aplikace, ale pokud nedojde k dokončení procesu i v AWS DynamoDB, tato nová aplikace se uživateli nebude zobrazovat na jeho přehledu v aimtec.cloud Portálu.

Pro zajištění správného zobrazování proklikávatelného odkazu na dashboardu aimtec.cloud Portálu je nutno provést přiřazení i v DynamoDB. Zde je seznam aplikací, ke kterým má uživatel přístup uchováván jako atribut na záznamu uživatele s názvem *solutions*, který je typu pole stringů, aby v něm bylo možno uchovávat vícero hodnot. Po nastavení tohoto atributu (přidání či odebrání aplikace) je proces přiřazení aplikace uživateli považo-

ván za dokončený. Nekonzistentní stav v rámci přiřazení nevede k selhání systému, ale řešení následných problémů je obtížné a je při nich potřeba komunikace se zákazníkem, zda aplikaci jen nevidí na dashboardu či se do ní nemůže přihlásit.

5 Projekt vývoje softwaru

Implementační část diplomové práce byla v Aimtecu vedena jako standardní vývojový projekt a byly na ni kladeny stejné požadavky jako na ostatní zákaznické implementační projekty. Musely být tedy splněny všechny standardní náležitosti a procesy, které se ve vývoji v Aimtecu používají. Tato skutečnost měla zajistit dodání kompletního a funkčního řešení a usnadnit vývoj produktu, který postupně prošel fázemi prototypu, minimálního životaschopného produktu a následně plnou funkcionalitu. V průběhu celého projektu bylo využito několika nástrojů pro snadnější řízení projektu jako je například systém pro plánování dílčích úkolů JIRA či systém pro verzování zdrojových kódů Git. Pro řízení vývoje byla použita metodika SCRUM.

5.1 Scrum

Jedná se metodiku vývoje, která představuje sadu doporučených postupů, nástrojů, pravidel a praktik jež se zabývají celým životním cyklem vývoje softwaru od jeho návrhu až po implementaci a otestování. Tato metodika je jednou z tzv. agilních metodik vývoje softwaru, které jsou založeny na Manifestu Agilního vývoje softwaru (Manifesto for Agile Software Development). Tento manifest vznikl na základě předchozích zkušeností programátorů, kteří dospěli k následujícím hodnotám:

- upřednostnění jednotlivců a interakce před procesy a nástroji,
- upřednostnění fungujícího softwaru před vyčerpávající dokumentací,
- upřednostnění spolupráce se zákazníkem před vyjednáváním o smlouvě,
- upřednostnění reagování na změny před dodržováním plánu [13].

Tato metodika je ve firmě zavedena z důvodu možnosti rychlého a efektivního reagování na požadované změny ze strany zákazníka a časté dodávky produktu v jednotlivých fázích. Jedná se tedy o iterativní a inkrementální způsob řízení vývoje softwaru. Jedna z klíčových vlastností je vysoká adaptace na požadované změny produktu ze strany zákazníka. V tomto přístupu k vývoji softwaru hraje také velkou roli transparentnost, kdy by všichni měli mít jasný přehled o tom, co a proč se dělá a v jakém je to stavu. Metodika má několik hlavních rolí, které jsou na začátku každého projektu definovány

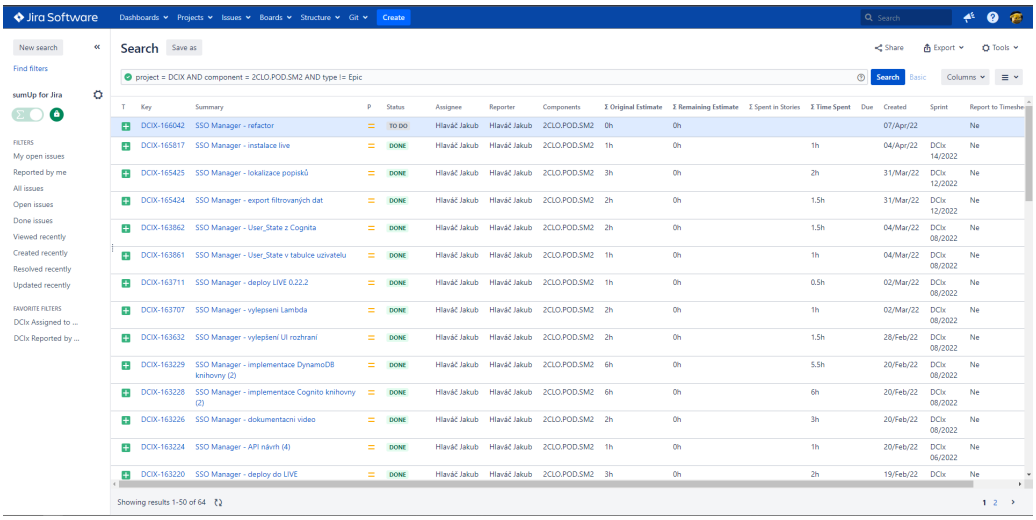
a přiřazeny jednotlivým lidem. Pro úspěšný vznik produktu jsou definovány tyto tři základní role:

- vývojový tým - jejich cílem je dokončit všechny aktivity, které jsou na daný sprint naplánovány
- vlastník produktu (Product Owner) - jedná se o zástupce zákazníka, který by měl vědět, co je potřeba v další fázi implementovat, a je v podstatě komunikačním článkem mezi vývojovým týmem a zákazníkem
- scrum master - osoba dohlížející na dodržování metodiky; pomáhá vývojovému týmu odstraňovat překážky, které brání k dokončení úkolů

Vývoj v metodice Scrum probíhá ve sprintech (obdoba iterací), které mají doporučenou délku jeden měsíc [14]. Ve firmě Aimtec je tento sprint dlouhý dva týdny a tento režim byl zachován i pro implementační část diplomové práce. Dle manifestu jsme na začátku každého sprintu (pondělí dopoledne) provedli plánování, kdy byly z produktového backlogu (seznam ohodnocených úkolů, které zbývá vykonat) vybírány aktivity s největší prioritou a odpovídající kapacitou, která byla dostupná v daný sprint. Tyto aktivity se následně přesunuly z produktového backlogu do sprint backlogu, kde došlo k jejich zafixování a následnému zpracovávání. V průběhu iterace jsou svolávány krátké denní schůzky, tzv. daily scrum, kde docházelo k předávání poznatků mezi členy týmu ohledně rozdělané, hotové či nedokončené práce. Na konci každé iterace je prováděna retrospektiva (pátek dopoledne), což je opět schůzka, na které je hlavním cílem identifikovat problémy a zjistit, zda jsou jejich příčiny systematicky řešitelné.

5.2 Jira

Jira je nástroj pro evidenci změnových požadavků a chyb. Tento nástroj umožňuje uživatelům sledovat jednotlivé úkoly, jejich současný stav, přiřazovat úkoly specifickému řešiteli či zobrazit, kdo na něm aktuálně pracuje. Aktuálně nabízí také podporu při řízení projektů, kdy poskytuje nástroje pro usnadnění spolupráce mezi vícero lidmi zároveň a přehledové statistiky. Nejzákladnější funkcionalitou, která je v Aimtecu využívána, je možnost zakládání úkolů a jejich evidence viz. obrázek 5.1. Tato evidence slouží k monitorování rychlosti práce na úkolech, kdy je každému úkolu na začátku iterace přiřazen odhad pracnosti a následně je k aktivitě vykazován strávený čas programátory při řešení. Zpětně lze tyto metriky použít pro lepší odhady



Obrázek 5.1: Obrazovka úkolů v Jira.

nových úkolů, které se podobají již dokončeným, u kterých byl v minulosti proveden nepřesný odhad pracnosti. Jira je přizpůsobená již zmíněné agilní metodice Scrum, kdy jsou jednotlivé úkoly plánovány do sprintů a přidělovány na konkrétní programátory. V Aimtecu se pracuje převážně se třemi hlavními typy úkolů:

- Epic - jedná se o souhrnný požadavek nějaké funkcionality, který je následně implementován v jednotlivých úkolech
- Story - přesně definovaný úkol
- Bug - nalezená chyba aplikace

Zároveň je u jednotlivých úkolů veden jejich stav rozpracovanosti, kdy jsou používány následující stavy:

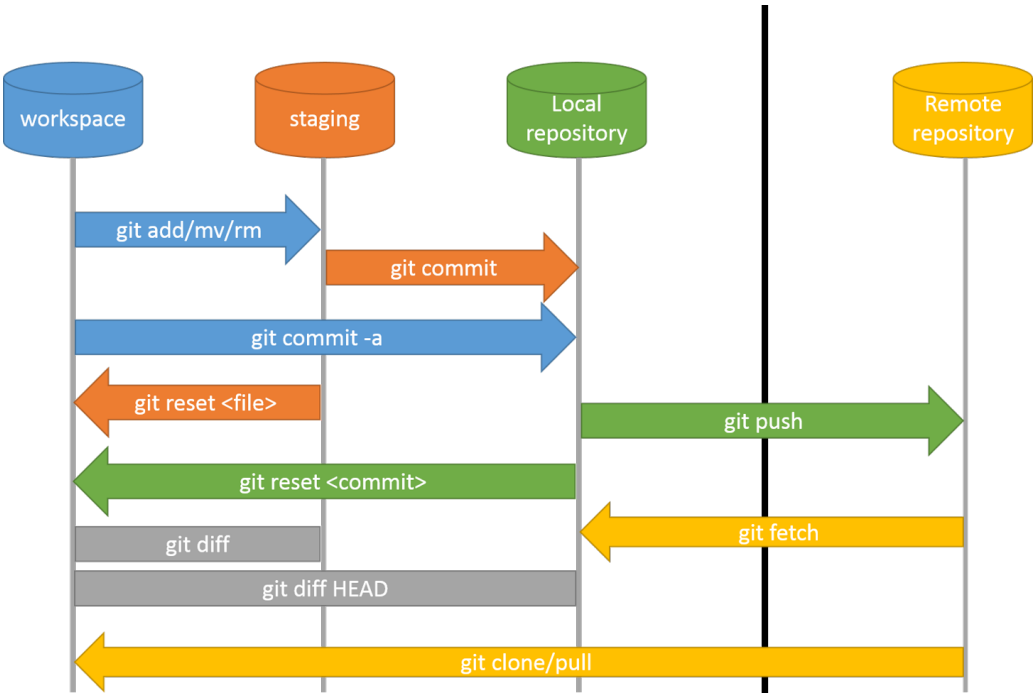
- TO DO - na úkolu se ještě nezačalo pracovat
- IN PROGRESS - úkol je rozpracovaný
- DONE - programátor ukončil práce na úkolu
- TAKEN OVER - úkol si převzal jeho zadavatel
- CANCELLED - pokud není úkol třeba dělat

5.3 Git

K ukládání a jednoduchému sdílení výkonného zdrojového kódu bylo využito nástroje Git. Tento distribuovaný systém pro správu verzí kódu je hojně rozšířeným řešením, jelikož každému vývojáři poskytuje lokální kopii celé historie vývoje. Od verzovacích nástrojů CVS nebo SVN je Git odlišný především v systému ukládání změn do repositáře. Tento krok je v Gitu rozdělen do dvou operací, kdy nejdříve operací *commit* dojde k zapsání změn do lokálního repositáře a následně jsou změny uloženy do společného úložiště operací *push*. Většina operací je tedy v Gitu prováděna nad lokální kopií. Projekt, který využívá nástroje Git, je rozdělen do tří hlavních sekcí [1]:

- adresář systému Git (adresář .git)
- pracovní adresář
- oblast připravených změn

Pro uložení změn do vzdáleného repositáře je nejprve nutné provést operaci *commit* do lokální databáze verzí a následně provést operaci *push*, která zapíše změny do vzdáleného repositáře viz. git příkazy na obrázku 5.2.



Obrázek 5.2: Příkazy verzovacího systému Git. Dostupné z [19]

5.4 IntelliJ Idea

Při implementaci backendové části v jazykce Java i frontendové čísti v jazyce Typescript jsem využil integrované vývojové prostředí IntelliJ Idea, které se v Aimtecu používá. Toto prostředí je navrženo tak, aby vývojářům usnadňovalo proces vývoje produktu od jeho ladění až po testování. IDE poskytuje intuitivní doplňování kódu, statickou analýzu a možnosti refaktorování v případech často se opakujících bloků výkonného kódu. Zároveň podporuje napojení na nejpoužívanější verzovací systémy a integrovaný klient pro prohlížení historie je velice intuitivní. Vývojové prostředí je multiplatformní a je možno ho nainstalovat na operační systémy Windows, macOS a distribuce Linuxu. Hlavním podporovaným jazykem je Java, nicméně do produktu je možno doinstalovat mnoho doplňků, které přidávají podporu například pro jazyky Python či TypeScript.

6 Návrh rozšíření

Diplomová práce stojí již na funkčním řešení, které je nasazeno v produkčním prostředí a zákazníci jej používají. Toto řešení ale postrádá jakékoliv prvky administrace uživatelů, aplikací a jednotlivých přístupů k aplikacím pomocí webového rozhraní. Aktuálně se tato administrace provádí nastavenými procesy, které byly popsány v kapitole 4. Z tohoto důvodu je potřeba navrhnout a implementovat rozšíření do aimtec.cloud Portálu, které bude administrátorům poskytovat dostatečný komfort při správě uživatelů a aplikací ve webovém rozhraní tohoto produktu. Tato kapitola se zabývá definicí požadavků na správu uživatelů a aplikací. Zároveň je zde také proveden návrh konečného rozšíření, které bude implementováno. Cílem navrženého řešení je přesun manuálních kroků, které jsou prováděny přes uživatelské rozhraní jednotlivých služeb, k plnohodnotnému řešení v uživatelsky komfortní webové aplikaci. Tohoto řešení je postupně dosaženo pomocí minimálního životaschopného produktu ve formě jednoúčelových command line nástrojů pro import do jednotlivých systémů až po zakomponování těchto nástrojů do aimtec.cloud Portálu.

Proces návrhu řešení, které bude následně implementováno, je důležitou částí každého vývoje softwaru a je definován ve všech moderních metodikách vývoje. V tomto bodě se definují základní stavební prvky řešení, kdy cílem je eliminovat kritické chyby. Vývoj softwaru bez předchozí analýzy, specifikace požadavků a návrhu řešení často tyto kritické chyby obsahuje, nedostatečně pokrývá uživatelské případy užití a jedná se často o nekvalitní řešení, které je poté těžké udržovat a rozšiřovat. Z těchto důvodů je zapotřebí přípravu projektu nepodcenit a absolvovat se zákazníkem dostatečný počet schůzek, kde dojde k důkladné specifikaci jeho požadavků.

6.1 Požadavky na rozšíření

Proces specifikace projektu a sběru změnových požadavků probíhal agilními metodikami vývoje. Po zahájení projektu bylo tedy potřeba definovat základní funkční i nefunkční požadavky systému. Dostatečná a jasná specifikace požadavků by měla usnadnit řízení projektu a poskytnout základní předpoklady k jeho úspěšnému dokončení a nasazení. Zároveň jsou závazným předpisem všech kritických funkcionalit systému, které jsou nutné pro akceptování produktu zákazníkem. Tato definice požadavků probíhala na několika schůzkách v rané fázi projektu, kdy se sešly zainteresované osoby

(zákazníci, programátoři a product owner). V tomto složení byly následně specifikovány veškeré požadavky, které jsou rozepsány v kapitolách 6.1.1 a 6.1.2. Požadavky bylo při implementaci řešení potřeba dodefinovat a provést minoritní změny zadání. Tyto změny bylo potřeba konzultovat se zákazníkem na schůzkách svolaných v průběhu implementace. V textu diplomové práce je pro přehlednost uveden seznam konečných požadavků, který byl schválen zákazníkem.

6.1.1 Funkční požadavky

Funkční požadavky specifikují jednotlivé případy užití, které bude zákazník vykonávat a systém je musí být schopen provést. Zároveň specifikují, jak by na tyto interakce měl systém reagovat a případné reakce na problémy s vykonáváním jednotlivých akcí. Tyto požadavky by tedy měly především odpovídat na otázky typu: *Co musí systém umět?* Pro sběr těchto požadavků je potřeba identifikovat všechny zainteresované osoby. Následně je od nich potřeba získat případy užití systému položením správných analytických otázek například:

- K čemu má výsledný systém sloužit?
- Kdo bude systém využívat?
- Jaké budou vstupy systému?
- Jaké budou výstupy systému?
- Jaké funkce bude systém plnit?

Odpovědi na pokládané otázky by měly obsahovat dostatek informací pro jejich následné rozpracování do konkrétních požadavků. Seznam získaných funkčních požadavků na rozšíření systému je uveden ve stromové struktuře níže.

1. Požadavky na role uživatelů
    (a) Systém bude implementovat roli Aimtec administrátora (smí provádět všechny akce)
    (b) Systém bude implementovat roli pro administraci všech aplikací daného zákazníka
    (c) Systém bude implementovat roli pro administraci jednotlivých aplikací

2. Požadavky na správu uživatelů
    (a) Webové rozhraní zobrazí existující uživatele
        i. Systém zařídí, aby obrazovku s uživateli viděl pouze uživatel s právy administrátora
        ii. Systém zařídí, aby přihlášený uživatel viděl pouze uživatele, se kterými má oprávnění manipulovat
        iii. Systém zařídí, aby přihlášený uživatel mohl uživatele přiřazovat k zákazníkům, se kterými má oprávnění manipulovat
        iv. Systém zařídí, aby přihlášený uživatel mohl uživatele přiřazovat pouze k aplikacím, ke kterým má oprávnění administrátora
        v. Systém musí umožnit provádět následující úpravy na uživateli
            A. Přiřazení uživatele k zákazníkovi (dle jeho kódu)
            B. Přiřazení uživatele k aplikaci (přiřazení přístupu)
            C. Přiřadit uživateli práva administrátora
            D. Přiřadit uživateli správu nad vyjmenovanými aplikacemi (dle zkratky aplikace)
            E. Přiřadit uživateli správu aplikací podle kódu zákazníka (dle kódu zákazníka)
        vi. Webové rozhraní bude zobrazovat status účtu
        vii. Webové rozhraní bude umožňovat obnovení hesla uživateli
        viii. Webové rozhraní bude umožňovat smazání uživatele
        ix. Webové rozhraní umožní uživateli exportovat aktuálně vyfiltrovaná data ve formátu .csv
        x. Webové rozhraní umožní v zobrazených uživatelích vyhledávat
    (b) Webové rozhraní umožní zakládání nového uživatele
        i. Založení uživatele pomocí jednoduchého formuláře
            A. Systém provede validaci povinného parametru email
            B. Systém provede správné formátování emailu na malá písmena
            C. Systém bude nabízet přiřazení uživatele jen k zákazníkům, ke kterým má přihlášený uživatel administrátorská práva

D. Systém bude nabízet přiřazení uživatele jen k aplikacím, ke kterým má přihlášený uživatel administrátorská práva

(c) Webové rozhraní umožní hromadné založení uživatelů pomocí importu ze souboru

i. Importované soubory budou ve formátu JSON

ii. Systém zařídí validaci vstupních dat ze souboru

3. Požadavky na správu aplikací

(a) Webové rozhraní zobrazí existující aplikace

i. Systém zařídí, aby obrazovku s aplikacemi viděl pouze uživatel s právy Aimtec administrátora

ii. Systém musí umožnit provádět následující úpravy na aplikaci

A. Změna URL aplikace

B. Změna jména aplikace

C. Změna statusových komponent aplikace

D. Změna typu aplikace

iii. Webové rozhraní umožní v zobrazených aplikacích vyhledávat

(b) Webové rozhraní umožní Aimtec administrátorovi zakládání nové aplikace

i. Založení aplikace pomocí jednoduchého formuláře

A. Systém provede validaci povinných parametrů: id, typ, URL, kód zákazníka

B. Systém bude nabízet přiřazení aplikace k existujícím zákazníkům

C. Systém bude umožňovat založit k aplikaci nového zákazníka

(c) Webové rozhraní umožní hromadné založení aplikací pomocí importu ze souboru

i. Importované soubory budou ve formátu JSON

ii. Systém zařídí validaci vstupních dat ze souboru

6.1.2 Nefunkční požadavky

Dalším typem požadavků jsou nefunkční požadavky. Jedná se o požadavky, které se netýkají přímo byznysové funkcionality, nýbrž slouží k dodefinování

pravidel a různých omezení, které se týkají technologického řešení. Tyto požadavky jsou rozděleny do různých kategorií, přičemž se některé z nich navzájem prolínají a je těžké určit, do které kategorie je zařadit. Mezi základní kategorie se nejčastěji řadí dostupnost, náklady, nasazování aplikace, rozšiřitelnost, výkon, spolehlivost a v dnešní době především bezpečnost. Základní požadavek na programovací jazyk, kterým má být Java nebo Python, je v této diplomové práci již specifikován v zadání. V následujícím seznamu jsou sepsány výsledné nefunkční požadavky, které jsou na výsledný systém kladeny.

1. Požadavky na technické řešení
   (a) Využití AWS DynamoDB
   (b) Využití Amazon Cognito
   (c) Backendová část implementována v jazyce Java
   (d) Frontendová část implementována v jazyce TypeScript
      i. Využití frameworku Angular
2. Požadavky na sestavení aplikace
   (a) Knihovny pro Cognito a DynamoDB sestavovány pomocí Jenkins pipeline
   (b) Aplikace backendové části sestavovány lokálně
   (c) Sestavení frontendové části pomocí Jenkins pipeline
3. Požadavky na nasazení aplikace
   (a) Backendová část nasazována do AWS Lambda funkcí
   (b) Napojení AWS Lambda funkcí na Amazon API Gateway
   (c) Nasazení frontendové části do kubernetes clusteru
4. Požadavky na bezpečnost
   (a) Požadavky budou zpracovávány pouze ověřeným uživatelům
   (b) Uživatelé uvidí jen data, ke kterým mají přidělená práva
5. Požadavky na webové rozhraní
   (a) Podpora multijazyčnosti (anglický jazyk, čeština)

6.2 Backend část

Backendová část bude dle nefunkčních požadavků implementována v jazyce Java. Pro správu kódu těchto oddělených jednoduchých aplikací bude využit jeden repositář ve verzovacím nástroji Git, kam dojde k postupnému commitování jednotlivých nově vznikajících funkcí. Jednotlivé aplikace budou využívat nástroj Maven, který slouží ke správě, automatizaci a řízení sestavování aplikací. Pomocí tohoto nástroje budou do jednotlivých aplikací vkládány knihovny nutné pro nasazování aplikací do AWS Lambda. Pro nasazování aplikací je potřeba myslet také na jejich sestavování, které bude prováděno do archivu **.jar**. Do tohoto archivu je potřeba při sestavování přidat také všechny závislé knihovny. Ve finálním řešení, které bude nasazeno do produkce, se počítá s vytvořením dvou knihoven, kdy každá bude implementovat funkce potřebné pro práci s jedním ze dvou systémů (AWS DynamoDB a Amazon Cognito). Tyto knihovny vzniknou z minimálního životaschopného produktu. Minimální životaschopný produkt bude implementován ve formě jednoduchých command line nástrojů, které budou provádět hromadné importy dat ze souborů ve formátu **.csv**. Nebudou podporovat operace mazání či odebírání práv uživatelům. Pro každý systém bude v této fázi projektu určen příslušný command line nástroj a pro provedení celého procesu přidávání uživatelů či aplikací bude nutno importovat data pomocí obou nástrojů.

Celkové řešení bude navrhnuto pomocí architektury mikroslužeb, kdy bude každá aplikace nasazovaná do lambda funkce brána jako jedna služba. Tyto služby je potřeba navrhnout a implementovat tak, aby na sobě byly maximálně nezávislé. Z toho vyplývá, že celkový počet služeb bude přibližně odpovídat počtu jednotlivých akcí, které lze vykonávat v grafickém rozhraní. Jednotlivé eventy, které budou tyto služby aktivovat, musejí podléhat autorizaci na straně AWS API Gateway, která na jednotlivých nově definovaných endpointech musí nést již implementovanou autorizační službu. Tuto službu bude pro potřeby implementovaného rozšíření nutno modifikovat, aby uživatelům povolovala přístup i k endpointům, které umožňují volat HTTP metody PUT, POST a DELETE. V aktuální verzi to tato služba neumožňuje a uživateli povoluje volat pouze metodu GET.

Pro každou aplikaci bude tedy potřeba vytvořit samostatnou Lambda funkci, které budou následně přidělena práva pro čtení a zápis dat do DynamoDB a Cognito. Pro verzování těchto lambda funkcí bude využito verzovacího nástroje uvnitř AWS. Pro developerské prostředí bude využíván vždy nejnovější nahraný archiv a pro testovací a produkční prostředí bude nutno specifikovat, jaká verze nahrané aplikace se má použít.

6.2.1 AWS DynamoDB

Rozšíření aplikace vyžaduje zavedení přístupových rolí do systému. Na tuto skutečnost není databázová vrstva připravena a je potřeba rozšířit modely jednotlivých záznamů uživatelů. Tyto změny se budou provádět do DynamoDB databáze, jelikož v ní se aktuálně uchovávají také přístupy k jednotlivým aplikacím a data z tohoto systému lze brát jako referenční. Pro potřeby zadání je nutno vymyslet mechanismus, jak uživateli přidělit roli administrátora a následně dospecifikovat, k jakým aplikacím má administrátorská práva mít. Tohoto chování bylo dosaženo přidáním několika nových sloupců do tabulky uživatelů.

Nový atribut **isAdmin** je typu boolean. Tento atribut bude následně použit při rozhodovací logice, zda je uživatel adminem či nikoliv. Výchozí hodnota je nastavena na *false*.

Oříškem bylo vymyslet jak uchovávat informace, u kterých aplikací má mít uživatel přidělená práva administrátora. Jelikož se zde jedná o vazbu typu M:N, bylo potřeba rozhodnout jak tato data uchovávat. Z analýzy NoSQL databází, dostupných best practices a aktuálního návrhu modelu jsem se rozhodl přidat nové atributy přímo na záznamy uživatelů [28]. Tyto nové atributy jsou **companiesAdmin** a **solutionsAdmin**. Oba atributy jsou typu pole textových řetězců a jsou nepovinné, kdy jejich absence znamená, že uživatel nemá administrátorská práva v žádné aplikaci, i když je označen jako administrátor. Jelikož existuje požadavek, aby se uživateli mohl udělit administrátorský přístup do všech aplikací daného zákazníka, využil jsem existence jmenných konvencí viz. 4.2. Uživateli se tedy přiřazují pouze společnosti, ve kterých je administrátorem, do atributu **companiesAdmin** a příslušné aplikace daného zákazníka jsou získány implementovanou logikou v lambda funkcích. U uživatele není potřeba explicitně vyjmenovávat všechny aplikace daného zákazníka a jsou mu automaticky přiřazena práva i v nově vzniklých. Pro vyjmenování specifických aplikací, kde má mít uživatel administrátorská práva, je provedeno pomocí atributu **solutionsAdmin**. Do tohoto atributu lze uložit pouze specifické identifikátory aplikací. Oba atributy lze kombinovat, kdy při jejich kombinaci dochází k sestavení kompletního seznamu aplikací sjednocením.

Pro nastavení maximálního oprávnění v podobě administrátora Aimtecu jsem využil kombinaci atributů **isAdmin** a **companiesAdmin**. Pro nastavení tohoto oprávnění je potřeba uživateli nastavit administrátora a zároveň ho přiřadit jako administrátora společnosti *AIM*. Tato role je využívána k přístupu na obrazovku administrace aplikací, která je v této fázi produktu dostupná pouze administrátorům Aimtecu.

6.3 Frontend část

Frontendová část bude dle požadavku implementována v jazyce TypeScript s využitím frameworku Angular. Bude se jednat o rozšíření již existující aplikace, která je aktuálně nasazena a byla vyvíjena na verzi Angular 8. Tato verze již není oficiálně udržována a tudíž je v rámci implementace naplánováno povýšení na aktuální verzi Angular 12 [9]. Toto povýšení umožní v implementaci používat modernější přístupy a nové syntaktické konstrukce. Uživatelské rozhraní by se tedy mělo rozrůst o část administrace, do které by se měl přihlášený uživatel dostat pomocí záložky v navigačním menu. Administrační část bude rozdělena na dvě sekce. V první sekci se zobrazí uživatelé a lze zde provádět akce pro manipulaci se zobrazenými záznamy. V druhé sekci budou následně zobrazovány aplikace a tato obrazovka bude implementovat specifikované akce, které lze nad aplikacemi provádět.

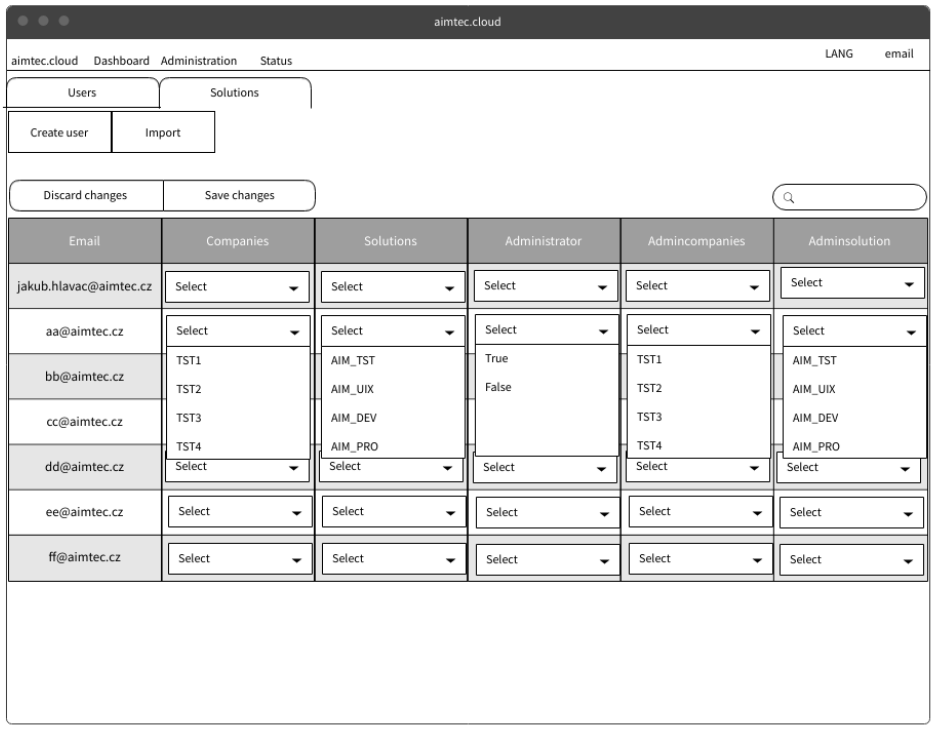
Pro jednodušší implementaci celého webového rozhraní byl použit šablonovací framework PrimeNG. Framework je open source a lze jej využívat pod MIT licencí. V aktuální verzi nabízí více jak 80 různých komponent uživatelského rozhraní. Komponenty jsou připraveny pro použití a případnou customizaci uživatelem tak, aby stylově odpovídaly návrhu webového rozhraní. Využívání jednotlivých šablon je velice snadné také díky obsáhlé dokumentaci. Výhodou vybraného řešení je existence příkladů použití s názorným zdrojovým kódem, který je dostupný na webu pro většinu nabízených komponent. Data zobrazovaná komponentami se předávají pomocí parametrů. Předávání statických textů pomocí parametrů je problematické vzhledem k požadavku na podporu multijazyčnosti, kdy je aktuálně překlad aplikace prováděn promocí Angular aparátu i18n.

Aparát i18n umožňuje v HTML tagu definovat identifikátor *i18n*, který je využíván k následnému generování předpisu ve formátu XML pro překlad textu. Parametr slouží k vytváření předpisu pro překlad textu uvnitř daného HTML tagu. Pokud je tedy do komponenty text předáván jako parametr, nelze ho tímto aparátem přeložit. Zároveň je zde absence překládání textu, který je generován pomocí javascriptu a následně vložen do stránky. Tento problém již v minulosti při implementaci aplikace nastal v případě internacionalizace položek v navigačním menu. Programátor ho vyřešil pomocí přeložených textů uložených v DynamoDB. Vytvořil zde tabulku ve které uchovává jednotlivé překlady pod klíčem a identifikátorem jazyka. Následně získává překlady při každém zobrazení položky v menu dotazem přes lambda funkci do databáze. Tento přístup se mi nezdá jako vhodný a po konzultaci s programátorem, který má aktuálně aplikaci pod svojí správou, jsme našli jiné řešení, které by se v budoucnu mělo použít v celé aplikaci. Toto řešení

spočívá v použití knihovny NGX-Translate. Knihovna využívá pro překlad aplikace definované klíče ve formátu **.json**. Klíče jsou uloženy v souborech, které jsou zabaleny spolu s aplikací ve složce *assets*. Jejich čtení se následně provádí pomocí HTTP dotazů do těchto souborů a překlad je tedy prováděn za běhu aplikace. Nový přístup také umožní budoucí odstranění dvojího sestavování aplikace, kdy se aktuálně pro každý jazyk sestavuje kompletní aplikace, a tudíž dochází ke zpomalení celého procesu sestavení. Sestavené aplikace jsou poté na server nasazovány simultánně do podsložek, které určují jejich jazyk "/cs"a "/en". K zobrazení dané jazykové mutace je nutno následně mezi těmito složkami uživatele přesměrovávat.

6.3.1 Draft

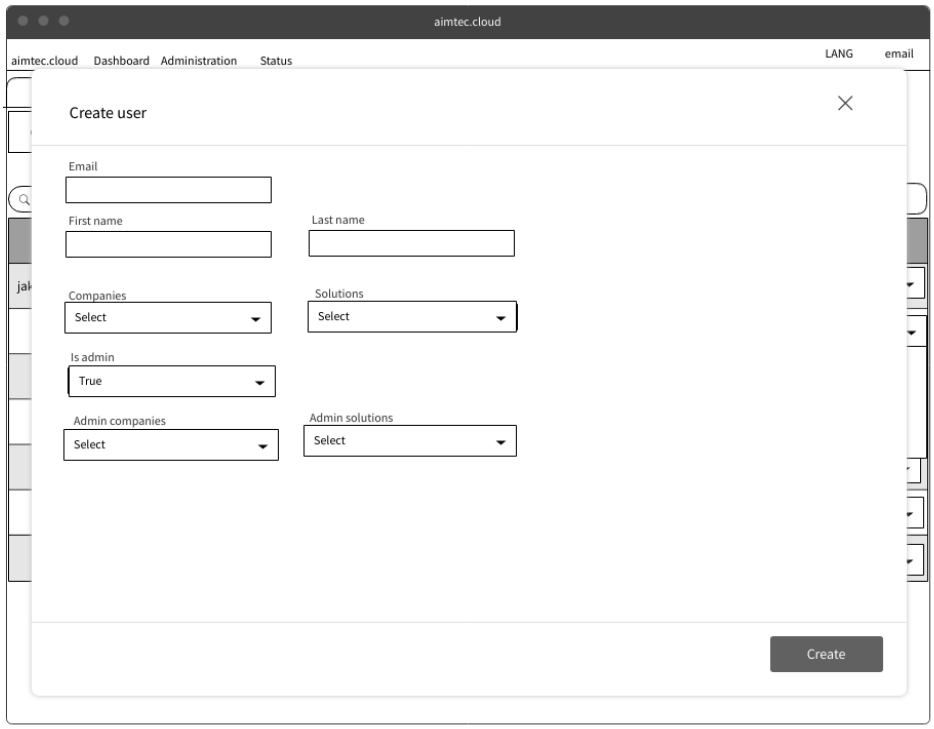
Před začátkem implementace webového rozhraní byl proveden rychlý náčrt grafického rozhraní. Tento návrh byl proveden pouze pro obrazovku uživatelů, viz. obrázek 6.1. Na záložce uživatelů jsou tedy v přehledové tabulce



Obrázek 6.1: Návrh wireframe obrazovky administrace uživatelů.

zobrazeni již registrovaní uživatelé. Tabulka obsahuje v horním řádku tla-

čítka jednotlivých akcí, které lze s daty v tabulce provádět, a na pravé straně pole, pomocí kterého je možno v záznamech vyhledávat. Následuje hlavička tabulky, kde jsou popsány jednotlivé sloupce. Úprava záznamů se provádí pomocí modálních oken pro vícenásobný výběr z příslušných hodnot. Toto je aplikováno na všechny sloupce, ve kterých lze upravovat hodnoty kromě sloupce **Administrator**. V tomto sloupci lze vybírat pouze jednu hodnotu a slouží k přepínaní práv. Pokud je uživateli nastaven tento sloupec na hodnotu *true*, jsou následně odblokovány interakce nad sloupci **Admin companies** a **Admin solutions**. Návrh pro modální dialog s formulářem pro vytvoření uživatele je zobrazen na obrázku 6.2. Na této obrazovce je zobrazen formulář,



Obrázek 6.2: Návrh wireframe obrazovky vytvoření uživatele.

do kterého je potřeba doplnit příslušné parametry. Tento formulář obsahuje jak textová pole, tak pole pro výběr z několika hodnot (jako je tomu na přehledové obrazovce). Formulář je validován na vstupní data a v případě nevyplnění povinných polí, je toto pole označeno červeně. Pro obrazovku aplikací se již od začátku projektu počítalo s obdobným stylem zobrazení, kdy byly prvky pro vybírání ze seznamu nahrazeny poli pro vkládání textu. Grafické zobrazování seznamu aplikací je tedy graficky totožné s obrazovkou

uživatelů. To samé platí pro obrazovku zakládání aplikace.

V průběhu projektu došlo k drobným úpravám návrhů. Po dokončené implementaci nad mockovanými daty byla naplánována schůzka s externím UIX designérem Petrem Širokým, kterého Aimtec využívá pro konzultace u produktů s grafickým rozhraním. Na schůzce jsme si vyslechli důležité podněty, jak vytvářet moderní uživatelské rozhraní. Z těchto podnětů a při následné ukázce již implementovaného uživatelského rozhraní došlo k diskuzi nad změnami, které konzultant doporučil zapracovat. Mezi tyto změny patřilo především vylepšení práce s daty v tabulce, kdy došlo k prohození vyhledávacího pole a tlačítek pro akce s daty tabulky. Toto prohození bylo zvoleno pro snadnější orientaci, jelikož uživatel bude nejčastěji vyhledávat podle emailu, a je tedy dobré, aby měl vyhledávací pole nad hlavním sloupcem tabulky. Došlo zde i k drobným změnám v modálním dialogu pro zakládání uživatele. Zde bylo doporučeno více vizuálně oddělit sekce, kde se vyplňují data o uživateli, a sekci, kde je následně řešen přístup do administrace a přidělování práv. Zároveň jsme pozměnili vzhled jednotlivých tlačítek, kdy byly pro akce s daty v tabulce zvoleny tlačítka pouze s ikonami, které vystihují danou akci. Veškeré tyto připomínky byly následně implementovány přímo do výsledného řešení.

7 Implementace

V této kapitole je popsána celková implementace, která byla v rámci diplomové práce práce provedena. Jedná se o definici API rozhraní v AWS API Gateway. Dále o implementaci business logiky v rámci knihoven a jednotlivých bez serverových aplikací, které jsou nasazovány do AWS Lambda funkcí. Pro práci s těmito funkcemi bylo následně rozšířeno i grafické rozhraní již existující aplikace aimtec.cloud Portál, které umožňuje uživateli komfortní správu uživatelů a aplikací.

7.1 AWS API Gateway

Pro definici API rozhraní, pomocí kterého je prováděna komunikace s backend částí aplikace v podobě AWS Lambda funkcí, je využito služby AWS API Gateway. V této službě byly nadefinovány všechny potřebné endpointy, kterými jsou volány jednotlivé AWS Lambda funkce.

Nově definované endpointy byly rozděleny do dvou hlavních skupin podle toho, k jaké části aplikace se vážou. První hlavní skupinou jsou endpointy, které získávají data o přihlášeném uživateli. Tato skupina má URL prefix **/user**, za který jsou následně definovány jednotlivé operace:

- **/companies** - GET metoda vracející všechny společnosti, se kterými může uživatel manipulovat (přidávat do nich uživatele)
- **/rights** - GET metoda, která v těle odpovědi nese objekt s oprávněními uživatele (definice objektu viz. 7.1)

```
1        {
2          "type" : "object",
3          "properties" : {
4            "isAdmin" : {
5              "type" : "boolean"
6            },
7            "isAimtecAdmin" : {
8              "type" : "boolean"
9            }
10         }
11       }
12
```

Listing 7.1: Definice JSON objektu odpovědi volání API.

- **/solutions** - GET metoda vracející seznam aplikací, se kterými může uživatel pracovat

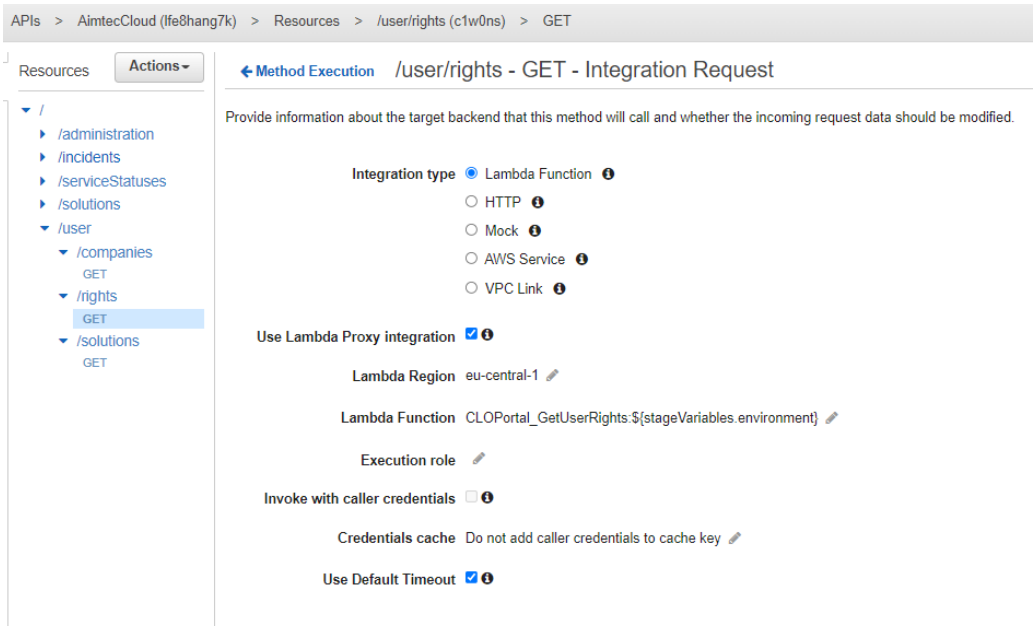
V druhé skupině, která je definována prefixem **/administration**, jsou zařazeny všechny endpointy pro práci s daty v administraci. Jedná se o API definice, které jsou využívány frontendovou částí při práci s administrační částí. Jednotlivé cesty operací jsou definovány následně:

- **/import** - POST metoda sloužící k importu uživatelů a aplikací z validovaného souboru
- **/solution** - POST metoda obsluhující požadavek vytvoření nové společnosti přes formulář, který je odesílán z frontendové části v těle požadavku
- **/solution** - DELETE metoda, která je volána při mazání aplikace ze systému
- **/solutions** - GET metoda vracející seznam aplikací, se kterými může uživatel manipulovat
- **/solutions** - POST metoda obsluhující změnový požadavek uložených dat o aplikacích, úpravy jednotlivých záznamů jsou uvedeny v těle požadavku
- **/user** - POST metoda obsluhující požadavek vytvoření nového uživatele přes formulář, obsah formuláře je formátován do JSON a je obsažen v těle požadavku
- **/user/{username}** - DELETE metoda s parametrem *username*, která je určená pro mazání záznamu uživatele
- **/user/{username}/reset-password** - POST metoda s parametrem *username*, která je určena pro resetování hesla uživateli
- **/users** - GET metoda vracející seznam uživatelů, se kterými může uživatel manipulovat
- **/users** - POST metoda obsluhující uložení změn na záznamech uživatelů, změny jednotlivých uživatelů jsou předávány v těle požadavku

Nově implementované rozšíření bude data zakládat také do systému Amazon Cognito, se kterým se před rozšířením aplikace nijak nepracovalo. Jelikož jsou pro jednotlivá prostředí, ve kterých je aplikace nasazena, definovány odlišné uživatelské skupiny v Amazon Cognito, je potřeba dodefinovat, ke které

se daný požadavek má v AWS Lambda funkci připojovat. Toto je zajištěno rozšířením parametrů již existujících stages. V každé stage došlo k dodefinování nového parametru, který se do funkce předá voláním endpointu. Parametr s jménem **cognitoIdentityPool** tedy v každé stage nese unikátní identifikátor skupiny uživatelů v Amazon Cognito.

Výše zmíněným endpointům byly následně pomocí uživatelského rozhraní AWS konzole přiřazeny v integrační části dotazu příslušné AWS Lambda funkce, viz. obrázek 7.1. Zde je potřeba zvolit za integrační typ lambda



Obrázek 7.1: Nastavení integračního dotazu v AWS API Gateway.

funkci. Následně je potřeba rozhodnout, na jaké úrovni se bude řešit integrace. Zda se bude řešit až v lambda funkci, či k transformaci dojde již na API Gateway. V diplomové práci je ve všech případech zvolena integrace v lambda funkci, jelikož se s použitím AWS knihoven jedná o snadnou implementaci. Dále byla každé API definici přiřazena lambda funkce s příslušným aliasem, viz. 7.2.3. Definice aliasu se lambda funkci předává jako parametr z proměnných daného prostředí a slouží ke spouštění správné verze implementace.

7.2 Backend

Business logika byla od začátku implementována v oddělených Java knihovnách. Tyto knihovny byly ve fázi minimálního životaschopného produktu

používány jako oddělené CLI nástroje pro hromadný import ze souboru do subsystémů (DynamoDB a Cognito) pomocí příkazové řádky. Řešení umožnilo již v průběhu vývoje provést změny v procesu zakládání uživatelů a aplikací, kdy byly tyto nástroje předány supportnímu oddělení. Knihovny byly následně upraveny a využívají se i v aplikacích nasazovaných do Lambda funkcí.

7.2.1 Proof of Concept

Základní milník celé implementace bylo prozkoumat možnosti použití jazyka Java v AWS Lambda funkcích. Bylo nutné úspěšně implementovat jednoduchou aplikaci, která pro komunikaci s Amazon Cognito a Amazon DynamoDB použije již předpřipravené sady vývojových nástrojů. Tyto nástroje jsou dostupné v balíčku AWS SDK for Java [11], který obsahuje základní implementaci rozhraní pro práci s AWS službami v jazyce Java. V rámci diplomové práce bylo využito pouze balíčků, které jsou potřeba pro práci s Amazon DynamoDB: *aws-java-sdk-dynamodb* a Amazon Cognito: *aws-java-sdk-cognitoidp*. Funkcionalita byla ověřena na jednoduché aplikaci nasazené do lambda funkce. Pro správnou funkčnost bylo potřeba s IT oddělením Aimtecu, domluvit definici potřebných AWS rolí, které budou mít přístupy pro zápis a modifikaci v DynamoDB a Cognito. Role určují s jakými oprávněními bude výkonný kód v Lambda funkcích spouštěn.

7.2.2 Minimální životaschopný produkt

Při implementaci proof of concept bylo při konzultaci s programátorem z jiného oddělení zjištěno, že v Aimtecu již bylo v minulosti implementováno jednorázové řešení na migraci uživatelů a aplikací při přechodu do cloudového prostředí. Toto řešení se skládalo ze dvou jednoduchých konzolových aplikací, které dokázaly importovat uživatele ze souboru ve formátu **.csv**. V rámci integrace kompletního systému do webového rozhraní a k urychlení převedení procesu správy uživatelů a aplikací v cloudu jsme se rozhodli tyto nástroje vylepšit a později je použít jako knihovny.

Cognito import

Prvním krokem bylo potřeba nástroj důkladně prozkoumat a zdokumentovat. V této fázi byly nalezeny problémy se zakládáním uživatelů do Amazon Cognito. Pokud byli uživatelé nově zakládáni do systému, nebyl jim odesílán uvítací email s předem vygenerovaným heslem. Jelikož ale nástroj slouží i k importu existujících uživatelů (přidávání dalších aplikací uživateli), je

potřeba určit, kdy se takový email má generovat. Pomocí parametrů z příkazové řádky lze spouštět import ve třech různých režimech:

- import - program provede import dat ze souboru (založení uživatelů, aplikací a přiřazení přístupových práv), uživatelům není odesílán uvítací email
- notify - program provede odeslání uvítacího emailu všem uživatelům uvedených v souboru
- importAndNotify - program provede import dat ze souboru a následně odešle všem uvedeným uživatelům uvítací email

Při implementaci tohoto rozšíření došlo také k rozšíření formátu vstupního souboru, kdy bylo doimplementováno zakládání uživatelů s jejich křestním jménem a příjmením. Výsledný formát vstupního souboru je následovný:

```
email,group,first name,last name
name1@domain.com,AIM_CLE_access,first_name1,last_name1
name2@domain.com,AIM_CLE_access,first_name2,last_name2
```

Listing 7.2: Formát importního CVS souboru pro Cognito.

První řádka souboru vždy obsahuje hlavičku, jednotlivé atributy jsou odděleny čárkou a jednotlivé záznamy jsou odděleny koncem řádky. Nevýhodou aktuální verze importu pomocí CVS je potřeba uvádět opakovaně email uživatele, pokud je potřeba uživatele přidat do vícero skupin jedním importem. Při zakládání nového uživatele do systému bylo implementováno automatické převádění emailu na malá písmena.

V dalším kroku bylo potřeba zajištění automatického sestavení knihovny, aby byla dostupná i běžným uživatelům a nebylo potřeba ji sestavovat před použitím. K tomuto účelu bylo využito již hotové pipelines v prostředí Jenkins, která se používá pro jiné projekty v Aimtecu. Tato pipeline sestaví aplikaci a následně ji nahraje do Maven repository, odkud je dostupná uživatelům v Aimtecu. Takto sestavená aplikace je následně po stažení spouštěna příkazem:

```
java -jar awsCognitoImporter-<version>-all.jar <
    AWS_Cognito_Pool_ID> [import|notify|importAndNotify] <
    CSV_file_name>
```

Pro správnou funkčnost je potřeba korektně nastavit přístupové údaje pro AWS SDK.

DynamoDB import

V nástroji pro import dat do systému Amazon DynamoDB byly provedeny úpravy v parametrech, se kterými lze importovat uživatele. Zároveň bylo potřeba opravit způsob ukládání již existujících záznamů, aby nedocházelo k odmazávání údajů, pokud nejsou zadány. Jelikož jsou data čtena z CVS souboru, je pro každé přiřazení aplikace k uživateli potřeba zduplikovat řádku s klíčem uživatele. Pro usnadnění vyplňování dat jsou některé parametry související se záznamem aplikace nepovinné. Pokud nedojde k jejich vyplnění a aplikace je již v systému založena, nechceme záznam upravovat. K úpravě záznamu aplikace dochází pouze v případě jejího zakládání při neexistenci. Toto zakládání bylo rozšířeno o chybějící parametry: *solution url*, *solution name* a *solution type*. Výsledný formát souboru:

```
email,company,solution,solution name,solution type,solution
    url
pnfb@uacro.com,YUK,AIM_CLE,Aimtec CLE,DCI,https://www.
    aimtecglobal.com/
pnfb@uacro.com,YUK,AIM_ACP,,,https://www.aimtecglobal.com/
pnfb2@uacro.com,YUK,AIM_CLE
```

Listing 7.3: Formát importního CVS souboru pro DynamoDB.

První řádka souboru obsahuje vždy hlavičku, jednotlivé atributy jsou odděleny čárkou a záznamy koncem řádky. Při zakládání nových uživatelů bylo v nástroji implementováno automatické převádění jejich emailu na malá písmena, aby došlo ke správnému spárování klíče v Amazon Cognito.

Stejně jako u nástroje pro import dat do Amazon Cognito byla definována Jenkins pipeline pro sestavování aplikace a její nahrávání do Maven repository, odkud je dostupná Aimtec užiatelům. Spouštění konzolové aplikace je prováděno následujícím příkazem:

```
java -jar awsDynamoDBImporter-<version>-all.jar [dev|test|
    prod] <CSV_file_name>
```

kdy parametr dev|test|prod definuje prefix tabulek, do kterých se má import provést.

7.2.3 Lambda implementace

Finálním řešením backendové části byla implementace několika jednoduchých aplikací, které jsou následně nasazovány do bezserverového prostředí v podobě AWS Lambda funkcí. Pro minimalizaci duplicitního kódu v rámci jednotlivých aplikací byly nástroje pro import dat do systémů z kapitoly 7.2.2 převedeny do podoby Java knihoven. Tyto knihovny jsou následně jednotlivými aplikacemi využívány jako závislosti. Celkový počet aplikací, které

byly v rámci diplomové práce implementovány, je 13. Jejich kompletní seznam je uveden v přílohách (viz. B).

DynamoDB knihovna

V rámci diplomové práce byla vytvořena knihovna, která obsahuje implementaci pro manipulaci s objekty v DynamoDB. Je následně použita v Lambda funkcích, aby nedocházelo k duplikování stejného výkonného kódu. Knihovna je definována jako Maven projekt a její kompletní definice je prováděna pomocí XML reprezentace v souboru **pom.xml**. V tomto souboru jsou definovány verze závislostí a typ archivu, do kterého se má projekt zabalit při sestavování. Jelikož se jedná o knihovnu, je zde zvoleno sestavování do archivu *jar*. Tento archiv je následně možno nahrát do Maven repository, odkud ho je možno využívat jako závislost v jiných Maven projektech.

Knihovna je tvořena hlavní třídou `DynamoDBClient`.

```
public class DynamoDBClient {
    private final UserService userService;
    private final CompanyService companyService;
    private final SolutionService solutionService;

    public DynamoDBClient(String environment) {
        userService = new UserService(environment);
        companyService = new CompanyService(environment);
        solutionService = new SolutionService(environment);
    }
    ...
}
```

Tato třída je v lambda funkcích inicializována pomocí konstruktoru, který má jako parametr `environment`. Pomocí tohoto parametru dochází k inicializaci tříd, které obsahují implementace funkcí pro práci s daty v DynamoDB. Toto řešení bylo zvoleno abychom byli schopni dynamicky měnit jména tabulek pro třídu `DynamoDBMapper`. Tato třída se stará o mapování dat z tabulky na Java objekty. Dynamická definice názvu tabulky pro mapování je implementována třídou `DynamoDBMapperConfig`, která se předává v konstruktoru při inicializaci mapperu spolu s inicializovaným objektem `AmazonDynamoDB`.

```
DynamoDBMapperConfig mapperConfig = new DynamoDBMapperConfig.
    Builder()
        .withTableNameOverride(DynamoDBMapperConfig.
    TableNameOverride
                .withTableNameReplacement(environment +
    COMPANIES_TABLE_NAME))
        .build();
mapper = new DynamoDBMapper(client, mapperConfig);
```

Při ukládání dat bylo v implementaci nutno myslet také na integritní omezení Amazon DynamoDB. Databáze neumožňuje ukládat atributy typu `Set <String>`, pokud jsou prázdné a neobsahují žádné prvky. V takovýchto případech dojde k vyhození výjimky, kterou vrátí používané DynamoDP API. Tato skutečnost je před každým ukládáním v knihovně ošetřena metodou, která se stará, aby všechny prázdné sety byly před ukládáním hodnot do databáze nastaveny na hodnotu `null`.

Knihovna implementuje následující metody pro práci s Amazon DynamoDB:

- void saveCompany(CompanyDynamoDB company)
- List<UserDynamoDB> getCompanyAdmins(String companyId)
- CompanyDynamoDB getCompanyById(String companyId)
- void saveSolution(SolutionDynamoDB solution)
- List<UserDynamoDB> getUsersWithSolution(String solutionId)
- void deleteSolution(SolutionDynamoDB solution)
- SolutionDynamoDB getSolutionById(String solutionId)
- List<CompanyDynamoDB> getAllCompanies()
- boolean userApproved(UserDynamoDB loggedUser, UserDynamoDB newUser)
- boolean saveUser(UserDynamoDB user)
- void deleteUser(UserDynamoDB user)
- List<UserDynamoDB> getUsersInSolutions(List<String> solutions)
- List<UserDynamoDB> getAllUsers()
- List<SolutionDynamoDB> getSolutionsByCompanyId(String companyId)
- List<SolutionDynamoDB> getAllSolutions()
- UserDynamoDB getUser(String username)

Knihovna je členěna do následujících Java balíků:

- entity - balík obsahující třídy, které reprezentují datový model entity v Amazon DynamoDB. Tyto třídy jsou používány pro mapování záznamů na Java objekty.
- services - balík obsahuje třídy implementující metody pro práci s Amazon DynamoDB.

Knihovna je sestavována pomocí Jenkins pipeline a následně nahrávána do interního Maven repository. V jednotlivých aplikacích je následně vkládána jako závislost a z tohoto repositáře je při jejich sestavování stahována.

Cognito knihovna

Pro manipulaci se záznamy v Amazon Cognito byla vytvořena knihovna, která je následně využívána v Lambda funkcích. Její použití je vhodné k zamezení duplikace stejného výkonného kódu napříč Lambda funkcemi. Podobně jako knihovna DynamoDB je definována jako Maven projekt a sestavována do archivu *jar*. Knihovna je také brána jako implementace klienta, který následně bude využíván v jednotlivých aplikacích. Hlavní třídou je zde `CognitoClient`.

```
public class CognitoClient {
    private final UserService userService;
    private final GroupService groupService;

    public CognitoClient(String identityPoolId) {
        userService = new UserService(identityPoolId);
        groupService = new GroupService(identityPoolId);
    }
    ...
}
```

Třída je inicializována pomocí konstruktoru s parametrem `identityPoolId`, který představuje unikátní klíč skupiny v Amazon Cognito. Tento objektový aparát zde byl zvolen opět z důvodu implementace dynamického dotazování do systémů pro různá prostředí. Parametr by šel předávat jako parametr v každé metodě, která je touto třídou implementována, ale docházelo by zde k znepřehlednění kódu. Pro snadnou orientaci v kódu byla implementace rozdělena do tříd podle oblasti, kterou obsluhuje (uživatel, skupiny) a pro které implementuje příslušné operace. Třídám je v jejich konstruktorech předán identifikátor Cognito skupiny a následně je ve voláních používána hodnota z inicializovaných tříd. Kompletní komunikace s Amazon Cognito je prováděna pomocí knihovny **aws-java-sdk-cognitoidp**, která implementuje potřebné metody.

Při implementaci metody `getAllUsers`, kdy je potřeba získat pro Aimtec administrátory záznamy všech uživatelů, bylo potřeba řešit problém s API omezením SDK Cognito, že Cognito API vrací při volání metody `listUsers` v základním nastavení pouze prvních sto záznamů. Pro získání všech uživatelů v systému bylo potřeba vyřešit stránkování. Pro řešení problému jsem implementoval metodu `getUsers(String pagination)`. Tato metoda odesílá dotazy do Cognito s parametrem **paginationToken**. Token je vracen v odpovědi API volání Cognito a slouží k automatickému stránkování záznamů. Pokud je jeho hodnota `null`, tak je iterováno přes všechny záznamy.

```
public ListUsersResult getUsers(String pagination) {
    ListUsersRequest listUsersRequest = new ListUsersRequest
    ().withUserPoolId(identityPoolId);
    if (pagination != null) {
        listUsersRequest.setPaginationToken(pagination);
    }
    try {
        return cognitoIdentityProvider.listUsers(
    listUsersRequest);
    } catch (AWSCognitoIdentityProviderException exception) {
        return null;
    }
}
```

Knihovna implementuje následující metody pro práci s Amazon Cognito:

- ListUsersResult getUsersInSolutions(List<String> solutions)
- ListUsersResult getAllUsers()
- Error removeUserFromGroup(String email, String solution)
- Error assignUserToGroup(String email, String solution)
- UserProcessStatus processCreateUser(User user)
- AdminListGroupsForUserResult getUserGroups(String email)
- boolean deleteUser(String email)
- UserProcessStatus processResetPassword(String userToResetPassword)
- Error createGroup(String solution, String groupDescription)
- Error deleteGroup(String solution)

Knihovna je členěna do následujících Java balíků:

- entity - balík obsahující třídy, které reprezentují datový model uživatele se všemi jeho atributy v Amazon Cognito.
- errors - balík obsahující třídy, pro uchovávání chybových stavů. Jedná se o objekty, ve kterých se ukládají chybové stavy při zpracovávání uživatelů. Tyto stavy jsou následně využívány k zobrazování neúspěšných operací uživatelům ve webovém rozhraní.
- services - balík obsahuje třídy implementující metody pro práci s Amazon Cognito pomocí volání API

Knihovna je opět sestavována pomocí Jenkins a do aplikací přidávána jako závislost.

Lambda aplikace

Pro nasazení aplikace do bezserverového prostředí AWS Lambda je potřeba definovat třídu, která bude implementovat rozhraní `RequestHandler`.

```
1 public class PostUser implements RequestHandler<
    APIGatewayProxyRequestEvent, APIGatewayProxyResponseEvent>
     {
2 ...
3 }
```

Toto rozhraní definuje metodu pro zpracovávání požadavku, který do Lambda funkce přijde. Metoda `handleRequest`

```
1 @Override
2 public APIGatewayProxyResponseEvent handleRequest(final
    APIGatewayProxyRequestEvent input, final Context context)
```

má dva vstupní parametry:

- `input` - první parametr definuje typ vstupního objektu, který bude Lambda funkce zpracovávat. Jedná se o třídu z balíčku **aws-lambda-java-events**. Tento balíček obsahuje Java reprezentaci jednotlivých akcí, které mohou do Lambda funkce přicházet. Jelikož je spouštěcí událost Lambda funkcí definována jako API Gateway a v nastavení API Gateway je přenechána kompletní integrace na Lambda funkci, zvolil jsem jako vstupní typ třídu `APIGatewayProxyRequestEvent`. Tato třída mapuje na Java objekt kompletní požadavek, který z API Gateway do Lambda funkce přijde.
- `context` - druhý parametr poskytuje metody pro získání údajů z výkonného prostředí Lambdy (jméno a verzi funkce, oprávnění pod kterými je daná funkce spouštěna, objekty pro práci s logováním do Amazon CloudWatch)

Metodu je tedy potřeba implementovat v každé třídě, která bude definována v nastavení Lambda funkce jako *handler*. Implementace je prováděna pomocí anotace `@Override` a business logika je následně prováděna uvnitř této metody.

Jelikož se jedná o bezstavovou implementaci a zároveň je kompletní integrační logika přenechána Lambda funkci. Je potřeba na začátku funkce validovat vstupní parametry a ověřit, že uživatel má dostatečná oprávnění. V jednotlivých funkcích se následně validují:

- `cognitoIdentityPool` - identifikátor Cognito skupiny, pokud není vyplněn funkce vrací Bad Request (HTTP kód 400)
- `environment` - prostředí aplikace, pokud není vyplněn funkce vrací HTTP kód 400
- `callerUsername` - identifikátor uživatele, který poslal požadavek. Tento identifikátor se získává z JSON Web Tokenu, který je uložen v hlavičce požadavku pod klíčem *Authorization*. Po převedení tokenu do Base64 používáme na jeho dekódování knihovnu **fasterxml**, která provede mapování na objekt `JsonNode`. Tento objekt obsahuje identifikátor uživatele pod klíčem *email*.

```
1         JsonNode jwtObject = mapper.readTree(new String(
    Base64.decodeBase64(jwt), StandardCharsets.UTF_8));
2
```

- bodyObject - textová hodnota u POST či PUT metod, která je předávána v požadavku pod atributem `body`. Mapování textového řetězce na Java objektovou reprezentaci je opět prováděno pomocí knihovny **fasterxml**. Zde je odlišná implementace pro použití v případě jednoduchého objektu a pole objektů. Pro jednoduchý objekt je implementována metoda `jsonAsObject`, která pomocí mapperu provede transformaci na Java objekt.

```
1     private <T> T jsonAsObject(String json, Class<T>
    clazz) {
2         try {
3             return mapper.readValue(json, clazz);
4         } catch (JsonProcessingException e) {
5             logger.log(e.getMessage());
6             return null;
7         }
8     }
9
```

U pole objektů je potřeba provést mapování přes třídu `TypeReference`.

```
1     private List<UserDTO> jsonAsUserList(String json) {
2         try {
3             return mapper.readValue(json, new
    TypeReference<>(){});
4         } catch (JsonProcessingException e) {
5             logger.log(e.getMessage());
6             return null;
7         }
8     }
9
```

Aliasy

Při vytváření Lambda funkcí v AWS konzoli bylo potřeba myslet na provoz v různých prostředích, kdy pro každé prostředí nechceme zakládat nové Lambda funkce. Zároveň je ale potřeba zajistit, aby se minimálně v produkčním prostředí vždy pouštěla konkrétní stabilní verze nahrané aplikace. Je potřeba zamezit neočekávaným stavům, kdy je nahrána nová verze aplikace pro testovací prostředí, ale v produkci je potřeba využívat předchozí verzi. Toto je zajištěno pomocí definování několika **Aliasů** viz. 3.2.2. U každé nově zakládané Lambda funkce byly založeny tři aliasy: dev, prod a test. Pro prostředí dev bylo nastaveno používání vždy nejnovější nahrané verze aplikace, kdežto u aliasů prod a test je vždy použita konkrétní verze. Tyto verze lze vytvářet pomocí uživatelského rozhraní AWS konzole viz. 8.3. Jednotlivé aliasy jsou poté mapovány na API Gateway, kdy je alias Lambda funkce navázané na endpointu definován pomocí suffixu. Tento suffix je přidáván k názvu Lambda funkce za dvojtečku [34].

Výkonnost

Jazyk Java pro Lambda funkce byl zvolen z důvodu snadné zastupitelnosti programátorů za cenu vyšších časových nároků pro studený start. Tento fakt jsem se snažil v rámci diplomové práce co nejvíce potlačit pomocí optimalizace kódu. Z dostupných zdrojů, které byly uvedeny v kapitole 3.2.2, vyšlo jako nejjednodušší řešení přiřazení vyšší velikosti paměti RAM. Velikost paměti RAM byla u všech Lambda funkcí nastavena na 1408 MB. Další zvyšování paměti již nepřineslo razantní zlepšení výkonu při studeném startu.

Dalšího zrychlení startu Lambda funkce bylo docíleno odstraněním co nejvyššího počtu závislostí aplikace na knihovnách třetích stran. Zde bylo provedeno převedení serializace a deserializace z knihovny **Gson** na knihovnu

fasterxml, která je do aplikace dotažena již v závislosti na AWS SDK pro Lambda funkce.

Poslední optimalizací studeného startu spočívalo v nastavení Java parametrů **-XX:+TieredCompilation** a **-XX:TieredStopAtLevel=1**. Tyto parametry slouží ke změně stupně kompilace. Od verze Javy 8 jsou kompilátory C1 a C2 používány kombinovaně. C1 kompilátor je navržen převážně pro klientskou část, kdy je optimalizován pro rychlé spouštění. C2 kompilátor je následně určen pro serverovou část, kdy již bylo dokončeno profilování a díky existujícím profilovacím datům je optimalizován pro nejlepší celkový výkon, ale využívá více paměti a jeho nastartování trvá déle. V Javě existuje pět různých úrovní vrstvené kompilace. Na úrovni 0 je interpretován Java byte kód, kdežto na úrovni 4 kompilátor C2 analyzuje profilovací data shromážděná během spouštění aplikace. Následně sleduje po určitou dobu používání kódu, aby nalezl optimální exekuční plán. Změna úrovně odstupňované kompilace na 1 může tedy zkrátit časy studených startů až o 60%. Změna úrovně kompilace byla zvolena také z analýzy zatížení Lambda funkcí, kdy se nepředpokládá kontinuální provoz, nýbrž jednorázové zatížení, kdy je potřeba Lambda funkci co nejdříve nastartovat a obsloužit požadavek [24]. Díky existenci **Layerů** pro Lambda funce, které se dají chápat jako automaticky se přidávající knihovny do běhového prostředí každé Lambda funkce, ke které jsou přiřazeny, je implementace této optimalizace jednoduchá. Bylo potřeba vytvořit nový layer, do kterého byl nahrán balíček obsahující kód pro změnu *JAVA_OPTS*.

```
export _JAVA_OPTIONS="-XX:+TieredCompilation -XX:
    TieredStopAtLevel=1"
```

Tento layer je poté přidán do nově vzniklých Lambda funkcí a pomocí nastavení proměnné **AWS_LAMBDA_EXEC_WRAPPER** je zajištěno jeho spuštění při startu běhového prostředí Lambda funkce.

7.3 Frontend

Rozšíření webového uživatelského rozhraní bylo implementováno do již existující aplikace aimtec.cloud Portál. Tato aplikace vznikla v roce 2019 na verzi Angular 7. Aplikace je členěna do modulů a komponent, které jsou navrženy podle Angular konceptu a best practices. Jednotlivé moduly odpovídají v aplikaci jednotlivým URL cestám, na kterých jsou poté zobrazovány jednotlivé komponenty daného modulu. Tyto komponenty definují pohledy, které se poté na základě logiky a dat zobrazují uživateli ve webovém prohlížeči. Modul obsahuje vždy jednu hlavní komponentu, do které je možno dynamicky

přidávat další komponenty ze stejného modulu. Toto je využíváno ke psaní čitelnějšího kódu, kdy není potřeba vše psát do jednoho souboru. Moduly `app-services` a `app-routing` jsou výjimky, jelikož zde nedochází k definování pohledů, nýbrž k definici služeb, které jsou následně používány napříč aplikací. Hlavním modulem celé aplikace je `app-module`, který je vstupním modulem celého řešení.

7.3.1 Povýšení verze Angular

Pro implementaci rozšíření bylo potřeba povýšit již dlouho nepodporovanou verzi Angular 7 na novější. Po diskuzi s vývojáři, kteří pracují na webovém rozhraní v Angularu pro aplikaci DCIx, bylo rozhodnuto o povýšení Angularu na verzi 12. Tato verze má garantovanou podporu do konce roku 2022. Povýšení bylo provedeno také z důvodu možnosti následně použít modernější knihovny, které pro Angular 7 nebyly dostupné, či došlo k jejich výraznému vylepšení. Toto bylo využito hlavně v rámci frameworku `PrimeNG`, který pro verzi 12 poskytuje již přes 80 komponent uživatelského rozhraní.

Jelikož se jednalo o povyšování frameworku přes 5 hlavních verzí, které obsahují různé nekompatibilní změny, bylo toto povyšování prováděno postupně, kdy došlo k povýšení vždy jen o jednu hlavní verzi. Po každém takovémto povýšení bylo nutné vyřešit problémy s knihovnami a zároveň ke kompletnímu sestavení aplikace pomocí standardních nástrojů a její nasazení do testovacího prostředí. Při povyšování byl využíván nástroj `https://update.angular.io/`, který uživateli zobrazí jednotlivé kroky, které je potřeba jednorázově provést při přechodu mezi hlavními verzemi Angularu. Update jednotlivých knihoven byl prováděn pomocí Angular CLI. Tento nástroj se ve většině případů postaral o migraci potřebných částí aplikace a o úpravu standardních konstrukcí v rámci výkonného kódu.

Při upgradu nastal problém s přechodem na TypeScript 4.0, kdy bylo potřeba upravit existující jednotkové testy. V těchto testech je potřeba v konstruktorech používat místo funkce `async()` funkci `waitForAsync()`. Zároveň bylo potřeba vyřešit při povyšování Angularu na verzi 10 problémy s Node.js. V této verzi je potřeba využívat Node verze 12 či vyšší. Hlavní problém nastal při povyšování knihovny `keycloak-angular`. V této knihovně bylo přepracováno rozhraní pro definici inicializačních parametrů Keycloak klienta. Pro zachování zpětné kompatibility standardních Aimtec konfiguračních souborů bylo potřeba použít k nastavení těchto parametrů knihovnu `keycloak-js`, která je následně využívána v inicializaci klienta pomocí knihovny `keycloak-angular`. Zároveň jsem se při povyšování také musel vypořádat s neexistencí kompatibilních používaných knihoven s no-

vějšími verzemi Angularu. Toto nastalo u knihovny `ngx-analytics`, která je již čtyři roky neudržována. Proto došlo k jejímu nahrazení za knihovnu `ngx-google-analytics`, která je v současné době nejpoužívanější Angular knihovnou pro Google Analytics.

7.3.2 Administrace

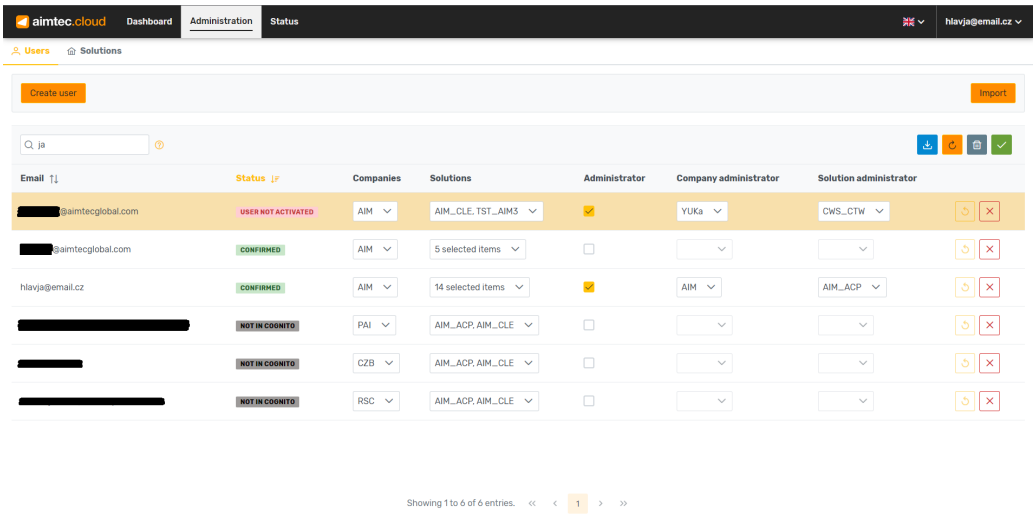
Implementované obrazovky byly přidány do modulu `administration`, na který se přihlášený uživatel dostane pomocí záložky v navigační liště. Jelikož se jedná o dvě jednoduché obrazovky, byly jednotlivé obrazovky pro editaci uživatelů a aplikací navrženy jako záložky na jedné obrazovce. K definici jednotlivých záložek byla využita komponenta **TabView**. V této komponentě byly pomocí konstrukce **TabPanel** nadefinovány dvě záložky, kdy každá zobrazuje patřičná data o uživatelích či aplikacích. Jednotlivé pohledy záložek jsou následně implementovány v oddělených komponentách `users-tab` a `solutions-tab`. Záludným oříškem v implementaci bylo potřeba zajistit, aby se načítala vždy jen data nutná pro zobrazenou záložku, která je aktivní. Toto chování bylo implementováno pomocí atributu *onChange* na komponentě **TabWiev**. Do tohoto atributu lze definovat vlastní metodu, které se jako parametr předává odchycený event, který na atributu *index* nese index záložky, na kterou je kliknuto.

```
handleChange(event: any) {
    switch (event.index) {
        ...
    }
}
```

Pro načtení příslušných dat při navštívení administrace je definováno volání této metody s parametrem pro první záložku v metodě `ngAfterViewInit()`. V rámci celé administrace jsou uživateli zobrazovány v horní části obrazovky indikační zprávy při komunikaci s backendem. Tyto zprávy obsahují status požadavku či výpis chyby, která je vrácena backendem. Mechanismus zpráv je implementován pomocí komponenty `Toast`.

7.3.3 Obrazovka uživatelů

Záložka pro práci s uživateli zobrazuje tabulku ve které jsou zobrazováni příslušní uživatelé, viz. obrázek 7.2. Záložka je implementována v komponentě `users-tab` a je členěna na dvě části. Horní část obsahuje tlačítka pro založení uživatele pomocí formuláře a pro hromadný import dat. V části pod těmito tlačítky je již uživateli zobrazena tabulka, která je implemento-



Obrázek 7.2: Obrazovka uživatelů.

vána pomocí komponenty `Table`. V parametrech této komponenty jsou následně dodefinovány potřebné atributy pro globální vyhledávání, hlavní klíč tabulky, načítací indikátor, responsivní zobrazení, jméno exportního souboru a stránkování či počet řádek.

```
<p-table #dt [value]="users" [rows]="30" [paginator]="true" [
    globalFilterFields]="['email', 'solutions', 'changed', '
    status']"
          responsiveLayout="scroll" scrollHeight="63vh" [
    rowHover]="true" dataKey="email" [showCurrentPageReport]="
    true"
          [loading]="loading" [columns]="cols" [
    currentPageReportTemplate]="'SHARED.TABLE_FOOTER' |
    translate" [exportFilename]="'Aimtec.cloud_User_Export'">
```

Hlavnička a tělo tabulky jsou následně definovány pomocí HTML tagů `thead` a `tbody`. Standardními HTML tagy jsou definovány i jednotlivé sloupce a řádky. V rámci tabulky jsou také definována dodatečná tlačítka a pole pro vyhledávání. Vyhledávací pole filtruje data podle definice, která je uvedena jako atribut tabulky. Pro vyhledávání v tabulce je použito základní vyhledávání komponenty, jelikož se ve většině případů užití nepředpokládá s vyhledáváním přes vícero polí zároveň. Tlačítka na pravé straně jsou dynamicky zapínána podle provedených akcí v tabulce. Tlačítka pro znovunačtení dat a export zobrazených uživatelů jsou aktivní vždy. Tlačítka pro práci s daty jako je uložení či zahození provedených změn v tabulce jsou uživateli zapínána až při provedení změny v tabulce. Pro implementaci smazání provedených změn bylo potřeba při provádění první změny duplikovat data, jelikož při změnách dochází k přepisování modelu, který je uveden jako vstupní

parametr tabulky.

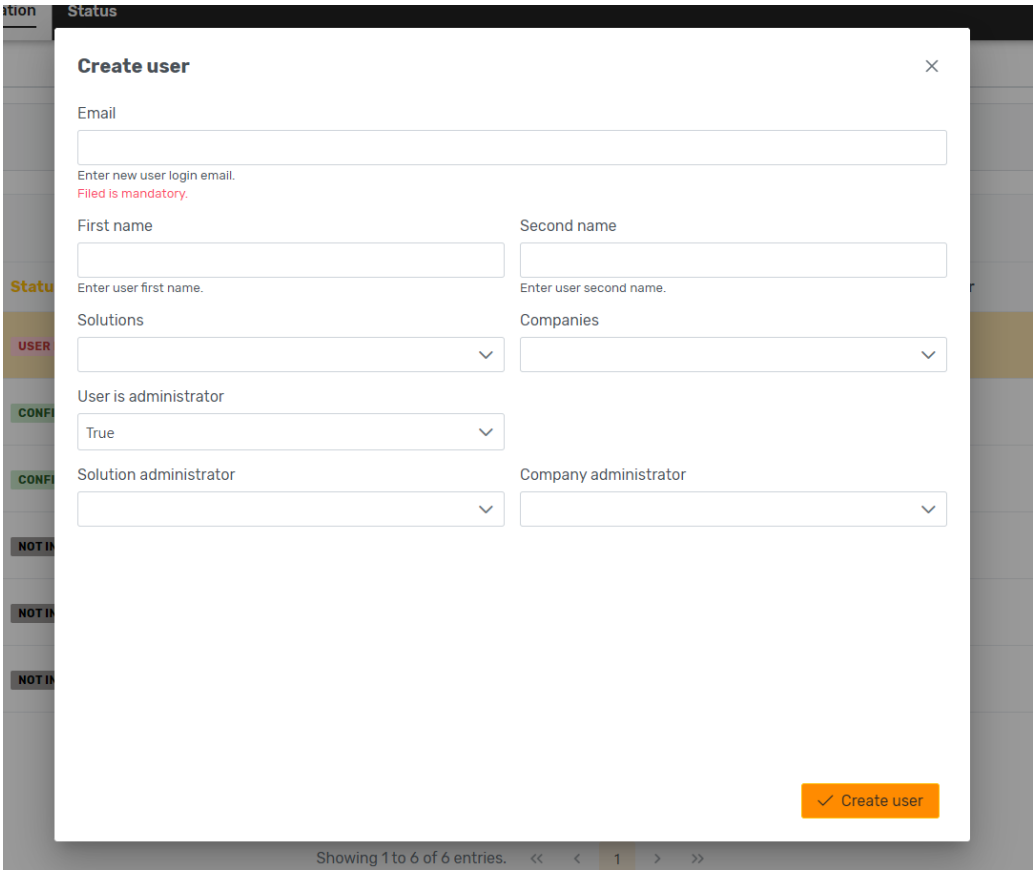
Jednotlivé řádky v tabulce představují záznam daného uživatele. Uživateli lze pomocí komponenty `MultiSelect` měnit společnosti, aplikace a administrátorské přístupy. Tyto komponenty dávají uživateli na výběr jen možnosti, které jsou definovány v rámci jeho administrátorských práv. Pro implementaci výběru pomocí této komponenty bylo potřeba vytvořit speciální datový model uživatele, který bude dané atributy ukládat v potřebném objektu. Tento model je rozšířen o několik atributů, do kterých se mapují příchozí data z endpointů pomocí standardní metody `map`.

```
1 users = users.map(user => ({
2                     ...user, primeNgSolutions: this.
   mapToPrimeNg(user.solutions),
3                     primeNgAdminSolutions: this.mapToPrimeNg(
   user.solutionsAdmin),
4                     primeNgCompanies: this.mapToPrimeNg(user.
   companies),
5                     primeNgAdminCompanies: this.mapToPrimeNg(
   user.companiesAdmin)
6                 }));
```

Tyto nové atributy v sobě nesou objekt s atributem `code`, do kterého jsou mapovány jednotlivé textové hodnoty. V pravé části každého řádku jsou následně uživateli zobrazována tlačítka pro resetování hesla či smazání záznamu uživatele. Tyto operace jsou jako jediné zpracovávány okamžitě a nedochází k nim při hromadném ukládání změn v tabulce. Při jakékoliv změně v datech je příslušná řádka záznamu podbarvena oranžovou barvou a je jí přidáván atribut `changed`, podle kterého lze poté filtrovat. Zároveň jsou všechny změněné řádky uloženy do proměnné, která tyto změny udržuje pro jejich pozdější odeslání na backend při hromadném uložení změn.

7.3.4 Založení uživatele

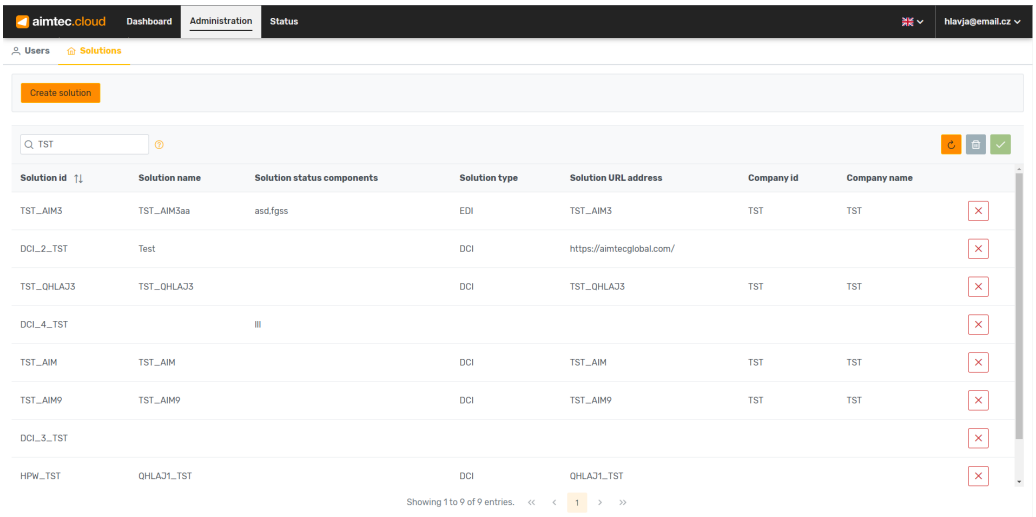
Zakládání uživatele je implementováno pomocí modálního okna, viz. obrázek 7.3, které je definováno v komponentě `add-user-popup`. Pro toto modální okno je použita komponenta `Dialog`, ve které lze nadefinovat hlavičku, tělo a patičku. V rámci těla okna je definován jednoduchý formulář, do kterého se vyplňují údaje o novém uživateli. Uživateli jsou již při otevření indikována povinná pole. Zde je to pouze email uživatele. Při vyplnění dotazníku a jeho odeslání jsou následně pole validována a v případě problému je uživateli zobrazena chybová zpráva. Při úspěšném uložení je modální okno automaticky zavřeno a nový uživatel je přidán do seznamu v tabulce.



Obrázek 7.3: Obrazovka vytváření uživatele.

7.3.5 Obrazovka aplikací

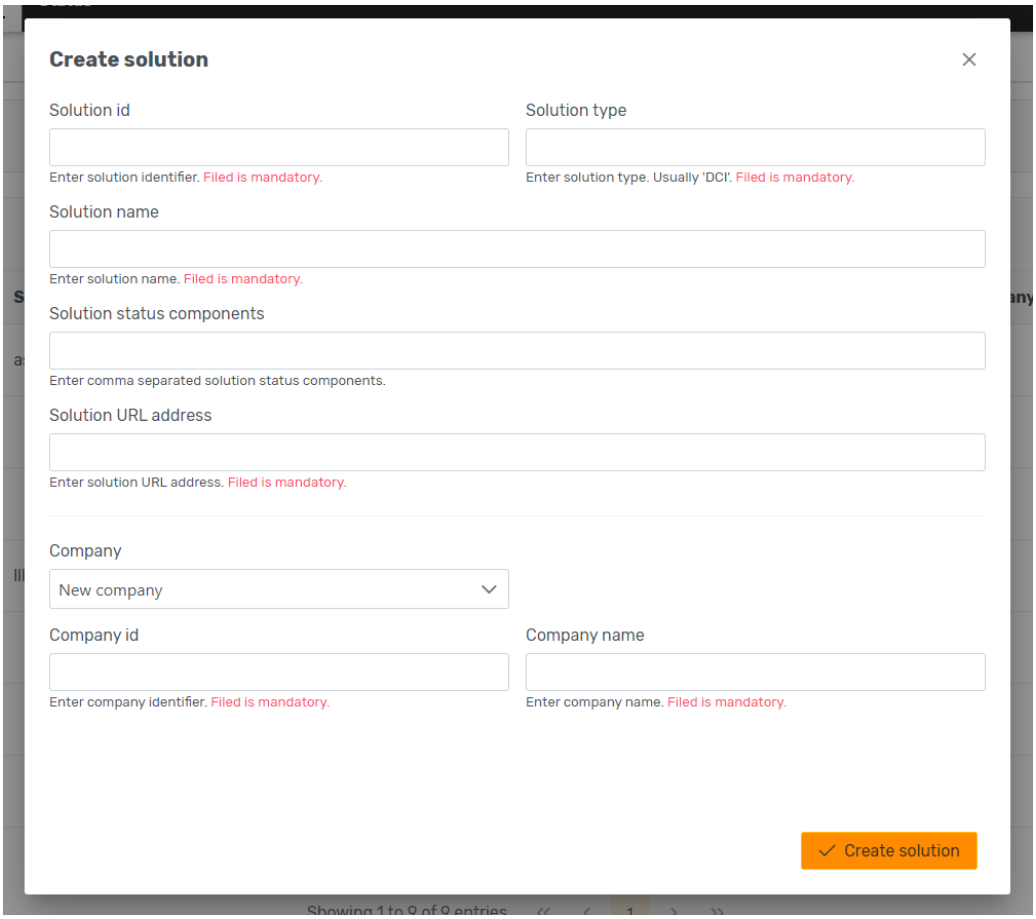
Obrazovka aplikací je implementována v komponentě `solutions-tab` a stejně jako obrazovka uživatelů je členěna do dvou částí, viz. obrázek 7.4. Tabulka je opět implementována pomocí komponenty `Table`, která je nastavena pomocí jejích atributů. Filtrování v tabulce je opět prováděno pomocí vyhledávacího řádku. Jednotlivé atributy záznamů jsou v této tabulce upravovány v textovém editoru, který je uživateli zobrazen při kliknutí. Editor je implementován pomocí komponenty `Inputtext`. Jednotlivé řádky, na kterých je provedena změna, jsou opět podbarveny oranžovou barvou a ukládání změn je prováděno hromadně pomocí tlačítka v hlavičce tabulky. Záznamy lze ze systému smazat pomocí tlačítka v pravé části příslušného záznamu. Smazání je provedeno okamžitě a v backendové funkci dojde k odstranění přístupů k dané aplikaci u všech uživatelů, kteří ji mají přiřazenou.



Obrázek 7.4: Obrazovka aplikací.

7.3.6 Založení aplikace

Modální okno pro zakládání aplikace pomocí formuláře, viz. obrázek 7.5, je implementováno v komponentě `add-solution-popup`. Modální okno je opět vytvořeno pomocí komponenty `Dialog`. Formulář obsahuje několik povinných polí, které jsou opět uživateli zvýrazněny červenou chybovou zprávou. Jedná se o id aplikace, typ, jméno a url adresu. Zároveň je potřeba aplikaci přiřadit k nějaké společnosti. Pokud je zakládána aplikace do již existující společnosti, lze tuto společnost vybrat z výběrového boxu. Při vybrání již existující společnosti se uživateli automaticky doplní příslušná pole společnosti. Pokud je potřeba založit novou společnost, musí uživatel ze seznamu vybrat možnost *New company* a následně vyplnil povinná pole. Jednotlivá pole formuláře jsou opět před odesláním validována. Při úspěšném založení aplikace je modální okno automaticky zavřeno. V případě chyby je uživateli zobrazena chybová zpráva v modální zprávě v horní části obrazovky.



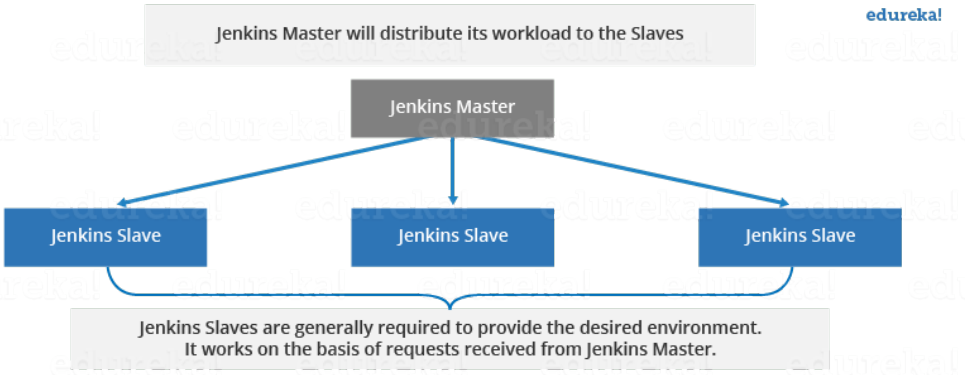
Obrázek 7.5: Obrazovka založení aplikace.

8 Sestavení a nasazení

K sestavení a nasazení aplikace bylo využito standardních nástrojů a postupů, které se v Aimtecu používají. Tato kapitola představí hlavní nástroj pro automatizaci průběžné integrace Jenkins, který je v Aimtecu používán pro sestavování a nasazování aplikací. Následně budou rozepsány jednotlivé postupy nasazování backendové části a frontendové části řešení.

8.1 Jenkins

Jenkins je multiplatformní open-source Java aplikace dodávaná pod MIT licencí. Jedná se o software, který slouží k automatizaci průběžného dodávání softwaru jako je například sestavování, testování a nasazování aplikací. Tento software je v dnešní době distribuován pomocí Docker kontejnerů, ale je možné ho stáhnout jako samostatně spustitelný balíček. Jedná se o velmi rozšířené řešení automatizace dodávky softwaru, které díky rozsáhlé knihovně pluginů dokáže integrovat nejpoužívanější vývojové nástroje jako je například Git, Maven a další. Zároveň je podporován nejznámějšími cloudovými poskytovateli, kteří nabízejí vlastní instalační balíčky Jenkinse. Jelikož dochází v dnešní době k rozpadu velkých aplikací na mikroslužby a každou službu je potřeba sestavit, došlo u Jenkins architektury ke změně, kdy se začal používat model *master-slave*. Tento architektonický model obsahuje



Obrázek 8.1: Architektura Jenkins. Dostupné online z [26]

tedy jeden hlavní server označovaný jako **master**, který se stará o plánování jednotlivých úkolů, distribuuje je na na dostupné slave nody a monitoruje jejich vytíženost. **Slave node** je spustitelný soubor Java, který běží na ser-

veru a odbavuje požadavky, které mu master přiřadí. Nody mohou běžet na různých operačních systémech a v jednotlivých projektech lze definovat, na jakém nodu se má dané sestavení aplikace provést, tudíž definujeme pravidlo master serveru, aby vždy vybral námi definovaný node.

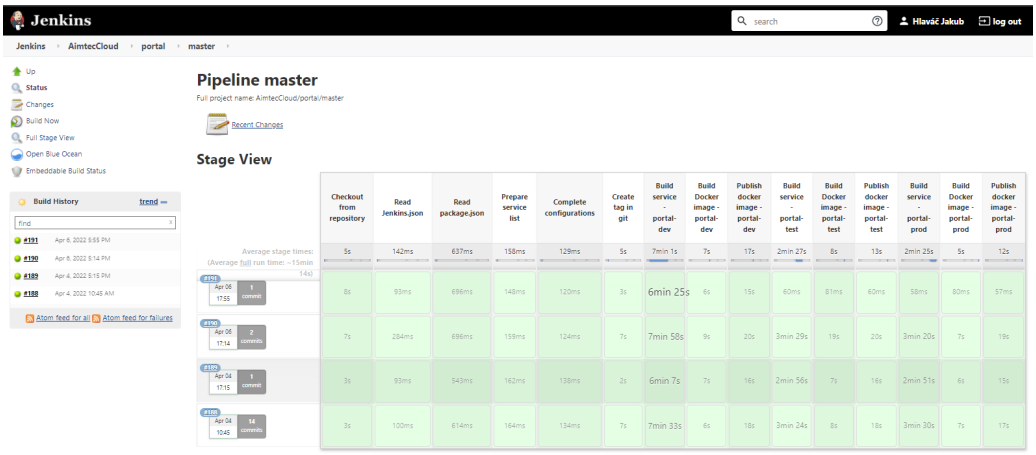
Job je základní prvek definující sestavení aplikace. Slouží k napojení na verzovací systém, nad kterým definuje jaké operace se provádějí před, během a po sestavení aplikace. Joby jsou poté nakonfigurovány, aby se spouštěly při jednotlivých eventech.

Piepeline je šablonovací nástroj jednotlivých jobů. Tato možnost byla do Jenkins dodána až v roce 2016 a přidala do systému snadnější podporu pro průběžnou dodávku softwaru. V pipeline jde rozdělit vykonávání jednotlivých kroků, což umožňuje ve vizualizaci vykonávání jednodušeji pozorovat, jaké kroky byly vykonány, či na jakém kroku sestavení či nasazení aplikace došlo k selhání. Další výhodou je také možnost sdílet kód pipeliny mezi vícero projekty či mezi programátory a týmy.

8.2 Angular aplikace

Nástrojem Jenkins je prováděno sestavování a nasazování fontendové aplikace, která je implementována pomocí Angularu a programovacího jazyka TypeScript. Pro sestavení aplikace je tedy potřeba definovat její sestavování na nodu, který má nainstalovanou příslušnou verzi systému Node.js a správce JavaScript balíčků Npm.

Sestavení aplikace nyní probíhá pomocí jobů, které jsou definované jako pipeliny. Tyto pipeliny mají svůj obslužný kód napsán v Groovy scriptech a jednotlivé kroky sestavení jsou definované jako *stage*. Sestavování aplikace nyní probíhá pouze na explicitní vyžádání, jelikož se nejedná o projekt s aktivním vývojem a nastavování jobu pro kontrolování změn ve verzovacím systému by bylo zbytečné a zároveň by periodické spouštění kontroly změn zatěžovalo infrastrukturu, která je využívána k sestavování a testování zbylého softwaru, který Aimtec dodává. Průběh sestavení spočívá v checkoutu aktuální verze zdrojových kódů pro danou branch z Git repositáře a následné sestavení aplikace pro každé prostředí, ve kterém je provozována a publikace Docker kontejneru na interní či veřejné úložiště, viz. obrázek 8.2. V rámci diplomové práce byl zefektivněn build a publikace Docker kontejnerů v závislosti na sestavované verzi, kdy se v Aimtecu pro verze, ve které stále dochází k vývoji, používá suffix **-SNAPSHOT** a tyto verze není potřeba sestavovat pro produkční prostředí. Vynecháním tohoto kroku v pipelině došlo k jejímu zrychlení o čtyři minuty.



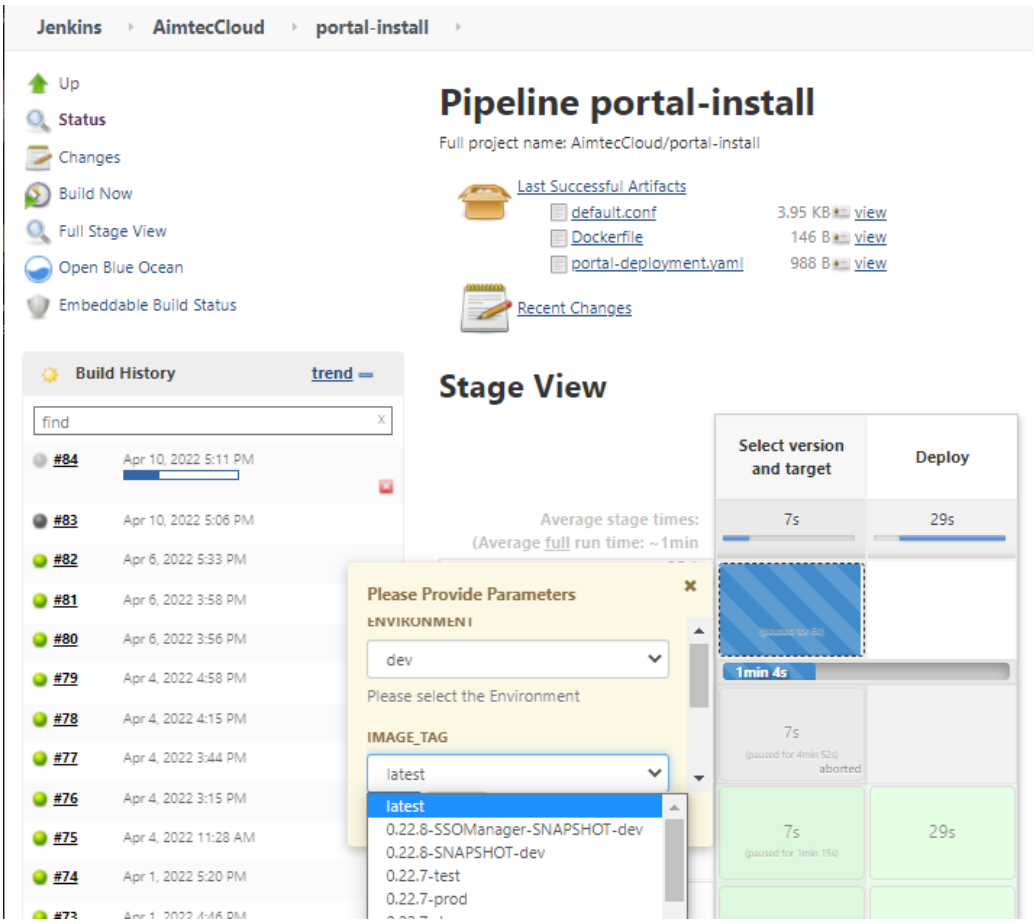
Obrázek 8.2: Sestavení aplikace na Jenkinsu.

Nasazování aplikace probíhá opět pomocí pipeliny, kdy je po spuštění daného jobu uživatel vyzván k vyplnění vstupních parametrů. Je tedy zapotřebí vybrat prostředí, do kterého se má aplikace nasadit a zároveň i verze nasazované aplikace, viz. obrázek 8.3. Zbytek za programátora již udělá Jenkins, který si automaticky z nastavení aplikace zjistí předpis nasazení, a příslušný Kubernetes server, kde je aplikace provozována. V pravém sloupci daného jobu lze poté pozorovat historii všech nasazení a jejich stav či kompletní záznam logu nasazení.

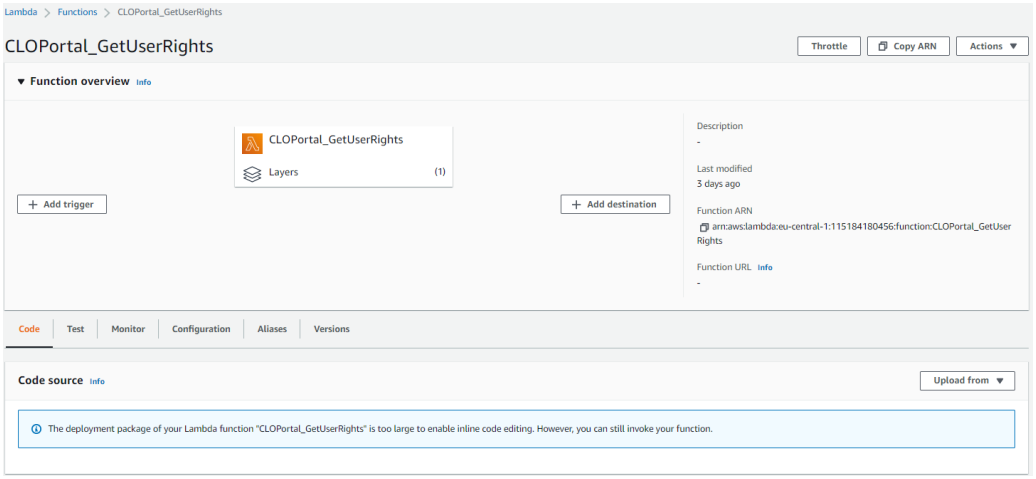
8.3 Lambda funkce

U nasazování Lambda funkcí bylo na začátku diplomové práce zamýšleno zautomatizování jejich sestavení a nasazení, bohužel kvůli bezpečnostní politice bylo nutno toto řešení odložit. Bylo by zde potřeba od IT oddělení zajistit přístupy do AWS CloudFormation, což je obdoba Jenkinse a slouží k automatizaci nasazování aplikací do služeb AWS. Bohužel tyto přístupy nebylo možno dostat a automatizace nasazování jednotlivých lambda funkcí byla provedena manuálními kroky, jako tomu bylo již při prvním vývoji aimtec.cloud Portálu.

Nasazování aplikací do Lambda funkcí je tedy prováděno ručními kroky pomocí AWS konzole, kdy je možno do jednotlivých funkcní nahrát sestavený archiv aplikace v *.jar* balíčku. Toto nahrávání je prováděno na hlavní stránce příslušné lambda funkce, kde je pro tento případ připraveno tlačítko **Upload from**, viz. obrázek 8.4. Po úspěšném nahrání balíku do lambda funkce je automaticky přiřazen výkonný kód pro alias funkce *dev*. Pro nasazení dané verze aplikace do ostatních aliasů *prod* a *test* je potřeba vytvořit novou verzi



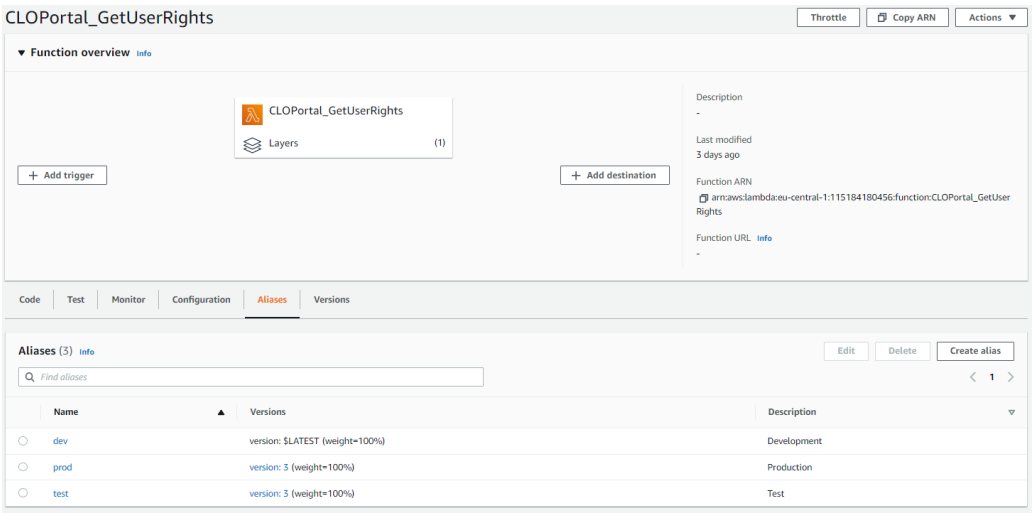
Obrázek 8.3: Nasazení aplikace na Jenkinsu.



Obrázek 8.4: GUI rozhraní lambda funkce.

dané lambda funkce, která si uchová aktuálně nahranou verzi aplikace a bude ji při zavolání verze lambda funkce spouštět. Toto se provádí pomocí tlačítka

Publish new version, které je schováno v rozbalovacím menu **Actions**. Po vytvoření nové verze lambda funkce je potřeba tuto verzi přiřadit ostatním prostředím pomocí úpravy nastavení na záložce **Aliases**, kde jsou zobrazeny všechny aliasy reflektující daná prostředí. Pro editaci aliasu je potřeba ho označit a zmáčknout tlačítko **Edit**. Uživateli se zobrazí obrazovka, na které může aliasu přiřadit verzi lambda funkce, která se bude pro dané prostředí spouštět, viz. obrázek 8.5.



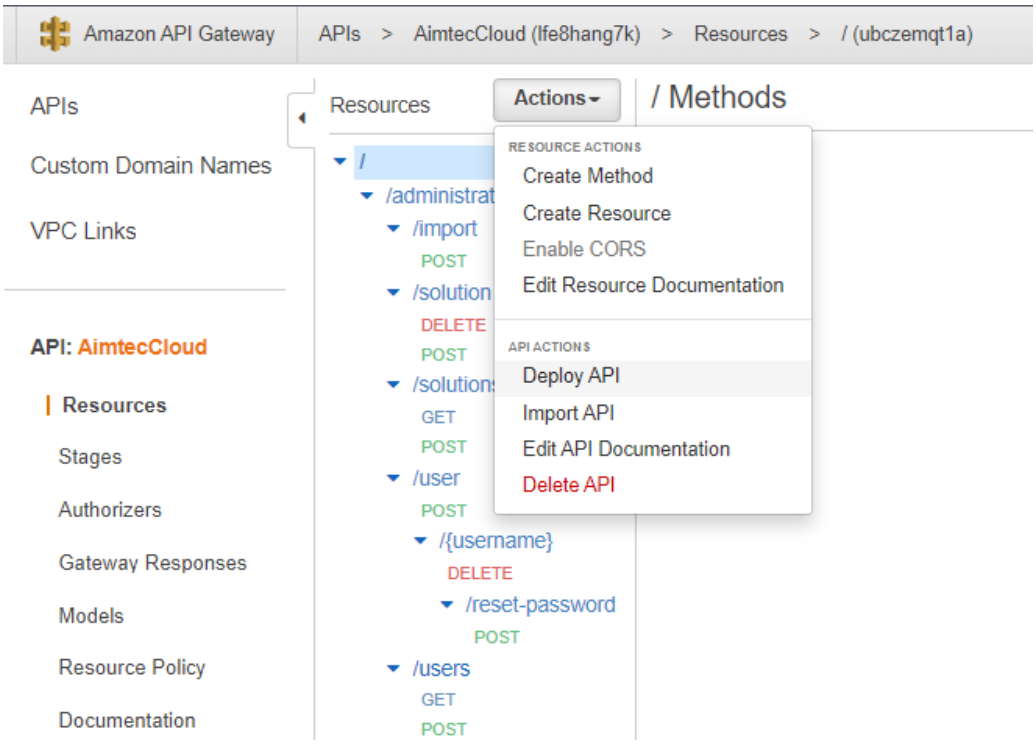
Obrázek 8.5: Seznam dostupných aliasů lambda funkce.

K zautomatizování sestavení jednotlivých aplikací, které se poté nahrávají do lambda funkcí, by bylo nutno pouze nastavit příslušný job, nicméně stále by zde zůstával manuální krok k jejich nasazení, kdy by si daný programátor musel stáhnout příslušnou sestavenou verzi z interního Maven repositáře a poté ji nahrát ručně do lambda funkce. Jelikož mají všichni zaměstnanci Aimtecu na svých lokálních počítačích nainstalovaný nástroj Maven, je pohodlnější, aby si programátor vyvíjející danou lambda funkci provedl sestavení lokálně a nebyl nucen pouštět sestavení na Jenkins a následně stahoval sestavenou aplikaci z Maven repositáře. Sestavení jednotlivých aplikací je prováděno pomocí příkazu *maven clean package*, který je pouštěn v hlavním adresáři.

8.4 API Gateway

Nasazování změn v Amazon API Gateway je prováděno také pomocí gui rozhraní AWS konzole. Po vybrání služby jsou uživateli zobrazeny všechny API definice, ke kterým má přístup. Po vybrání příslušné definice je uživatel

přesměrován do její administrace, kde pro nasazení dané verze API stačí vybrat z rozbalovacího menu **Actions** akci **Deploy API** a vybrat opět stage reflektující prostředí, kam se má API nasadit, viz. obrázek 8.6.



Obrázek 8.6: GUI rozhraní Amazon API Gateway.

9 Testování

K ověření správné implementace jednotlivých částí výsledné aplikace bylo využito několika nástrojů pro testování. Tyto nástroje byly zvoleny v závislosti na použitých knihovnách pro služby DynamoDB a Amazon Cognito, kdy pro některé zvolené služby nebylo možno napsat určité typy testů. K definici testovacích scénářů docházelo již při implementaci aplikace, aby byl výsledný produkt kvalitní a předešlo se neočekávanému chování po nasazení do produkce.

9.1 Backend

Testování backendové části bylo provedeno dvěma způsoby, které byly pro zvolené technologie dostupné. Jednotkové testy bezserverových aplikací jsou stejně důležité jako u standardních serverových aplikací. Zde jsem ale narazil na problém, kdy je v jednotlivých Lambda funkcích implementována business logika, která se snaží manipulovat s daty jak v systému Amazon DynamoDB tak Amazon Cognito. Pro testování jednotkovými testy by bylo tedy potřeba zprovoznit prostředí Docker, do kterého by se Lambda funkce nasazovaly. Následně by toto prostředí muselo mít nastavené dostatečné přístupy do služeb v AWS. Jelikož se jedná o komplikované řešení, kde by bylo potřeba definovat složité bezpečnostní procesy pro přístup ke službám mimo AWS, bylo od tohoto řešení odstoupeno. Jednotlivé Lambda funkce jsou testovány pouze integračními testy. Jednotkové testování bylo využito v rámci vytvořených knihoven.

9.1.1 Jednotkové testy

V rámci průzkumu vhodných nástrojů pro mockování Amazon Cognito v rámci jednotkových testů jsem nenarazil na existující zdokumentovaný případ užití pro jazyk Java. Volně dostupné jsou pouze návody, jak mockovat Cognito klienta v jazyce JavaScript. Proto bylo testování cognito knihovny přeskočeno a v tomto případě se spoléhá na integrační testy jednotlivých Lambda funkcí.

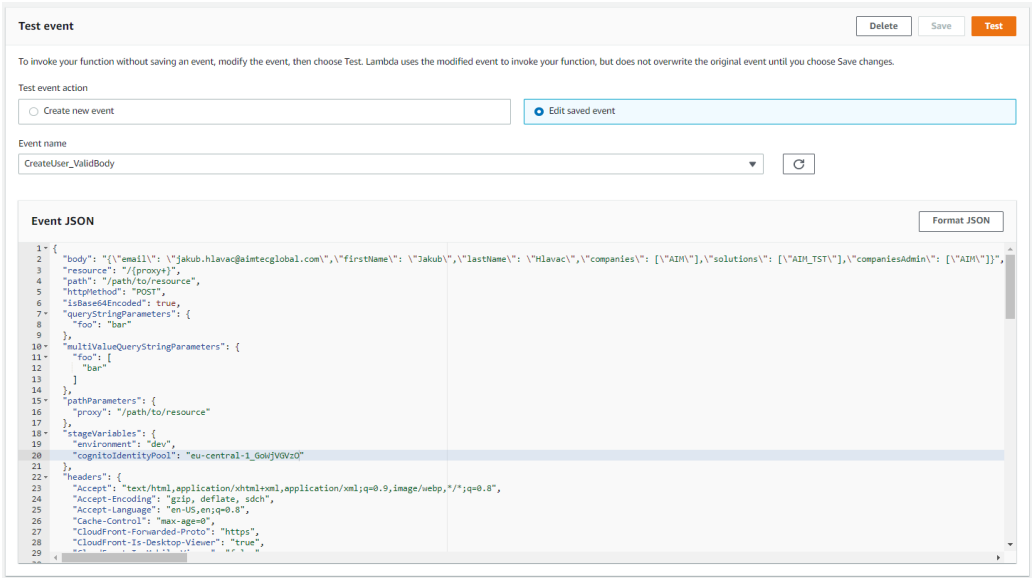
Testování jednotkovými testy bylo tedy provedeno pouze v knihovně, která zajišťuje práci s Amazon DynamoDB. Pro mockování této služby existuje v jazyce Java speciální knihovna `DynamoDBLocal`. Poslední uvolněná verze této knihovny je z roku 2021 a obsahuje několik bezpečnostních chyb,

takže je potřeba tuto knihovnu vkládat jako závislost pouze při spouštění testů definicí atributu **scope** na *test* při použití Mavenu. Jelikož tato implementace DynamoDB používá knihovnu `SQLLite4Java`, je potřeba dodatečného nastavení. Toto nastavení je potřeba z důvodu závislosti `SQLLite4Java` knihovny na prostředí, ve kterém běží. Proto je potřeba před spuštěním testů překopírovávat tranzitivní závislosti této knihovny do předem definovaného adresáře, který se následně použije při inicializaci lokální DynamoDB. Tento krok se provádí v rámci Maven fáze **test-compile** a je jeho předpis je definován v **pom.xml**. Testy byly následně otestovány všechny implementované metody ve třídě `DynamoDBClient`. Vzniklo tedy patnáct jednotkových testů.

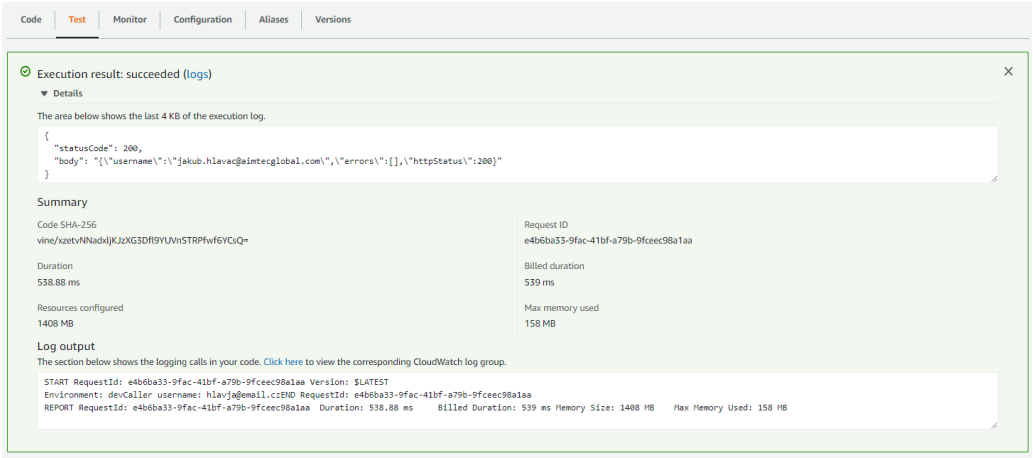
Psaní jednotkových testů v rámci implementace pomohlo odhalit chyby ještě před nasazením do testovacího či produkčního prostředí. Testovací scénáře jUnit testů odhalily zásadní chybu v implementované logice přiřazování přístupových práv. Tato logika byla díky testování opravena ještě před publikováním nové verze. Existence testů zároveň minimalizuje riziko použití nefunkční verze knihovny v aplikaci. Jelikož v případě spadlého testu nedojde k sestavení knihovny a jejímu uvolnění do Maven repository.

9.1.2 Integrační testy

Integrační testy byly z důvodu složitého zprovoznění testovacího prostředí s kompletní infrastrukturou Amazon Cognito a Amazon DynamoDB vytvářeny přímo v AWS. Zde je u jednotlivých Lambda funkcí možno definovat testovací události. Tyto události lze nakonfigurovat z několika předpřipravených šablon, které definují JSON objekt pro daný typ integrace. Jelikož jsou Lambda funkce spouštěny událostmi, které přicházejí z Amazon API Gateway, je jako výchozí šablona použita **API Gateway AWS Proxy**. Tato šablona poskytuje předpřipravenou JSON definici objektu, který je odesílán do Lambda funkce z API Gateway. Do této šablony je následně potřeba doplnit atributy, jako jsou **stageVariables**, **queryStringParameters**, **pathParameters**, a v případě POST a PUT metod i **body** s daty. Takto upravené šablony lze poté u každé Lambda funkce uložit pod vlastním jménem a následně je opakovaně pouštět viz. obrázek 9.1. Při pouštění testovací události dojde k nastartování Lambda funkce a provedení výkonného kódu. Všechny testy jsou nakonfigurovány tak, aby svými testovacími událostmi prováděly změny do systémů, které jsou určeny pro vývojové prostředí. To je zajištěno správnou konfigurací atributů **environment** a **cognitoIdentityPool** v parametru **stageVariables**. Nad touto definicí je poté zobrazen výsledek spuštěné testovací události, viz. obrázek 9.2. Ve výsledku je uživateli zobrazen kompletní výpis odpovědi, kterou Lambda funkce vrátila. Násle-



Obrázek 9.1: Testovací událost v AWS konzoli.



Obrázek 9.2: Výsledek testovací události v AWS konzoli.

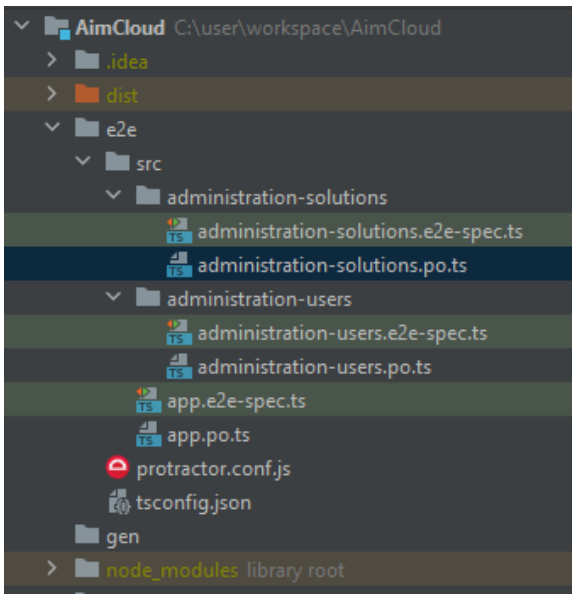
duje neformátovaný výpis logovacích zpráv s možností prokliku na kompletní výpis logu dané Lambda funkce v Amazon CloudWatch Logs (služba umožňující uživatelům shromažďovat a ukládat logy z ostatních AWS služeb) odkazem, který se nachází v textu nad zobrazenými logovacími zprávami. Při definici scénářů, kde byla v atributu **body** posílána data, bylo potřeba kódovat speciální znaky JSONu. Tato skutečnost vede k větší časové zátěži při zakládání nových testů a z mého pohledu trochu znepřehledňuje výsledný JSON v případě, kdy je potřeba definovat složitější objekty. U každé Lambda funkce byly v průměru definovány dvě patřičné testovací události, aby byla otestována správná funkčnost. Jelikož je potřeba tyto testy spouš-

tět manuálně, byl nadefinován proces, který toto zajistí, kdy je vyžadováno spuštění testů nad Lambda funkcí vždy před vytvořením nové verze, která bude použita pro produkční prostředí.

Definice integračních testů pomohla odhalit sémantické chyby při práci s Amazon Cognito. V určitých případech docházelo při neexsitenci záznamů k neočekávanému pádu aplikace, které skončilo výjimkou. Pouštěním těchto testů před publikováním nové verze Lambda funkce je zajištěna stabilita a kvalita produktu. Zároveň slouží k odhalování chyb při následném vývoji případných rozšíření business logiky.

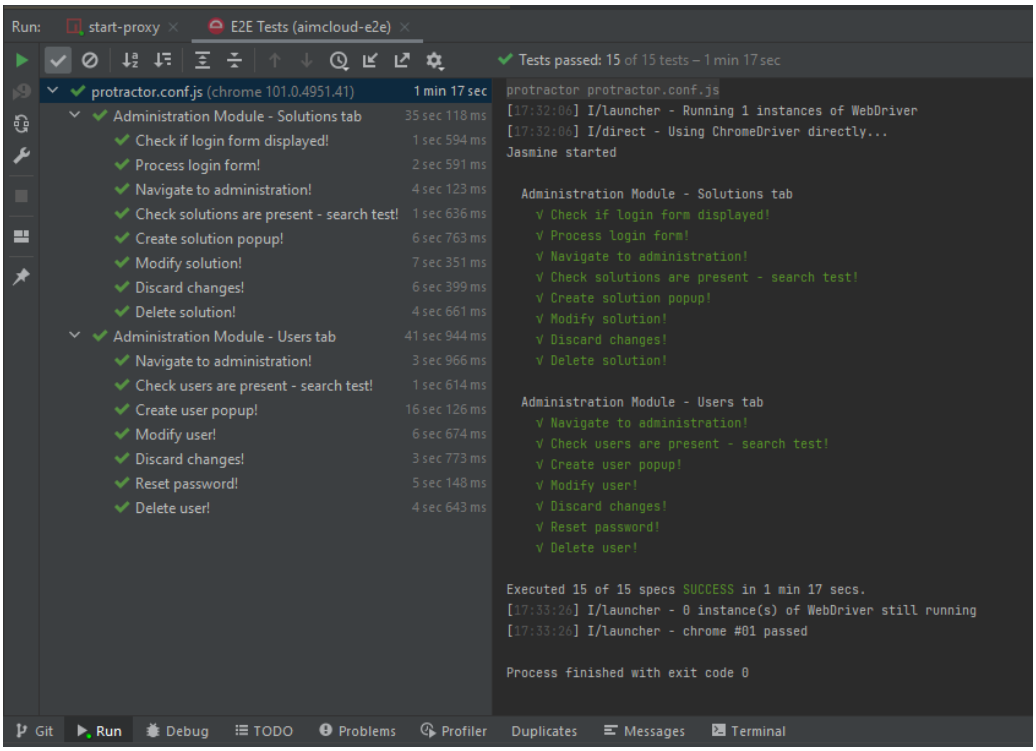
9.2 Frontend

Frontendová část implementace byla otestována pomocí automatických testů, které jeou definovány přímo v aplikaci. Tyto testy se nacházejí ve složce `e2e/src` a ke svému běhu potřebují běžící frontendovou aplikaci. O její nastartování se postará samotné spuštění testů. Jelikož se k sestavování aplikace používá Jenkins pipeline, její testování by značně prodloužilo její celkový běh, což by vedlo ke zpomalení sestavování ostatních aplikací. Do této standardní Jenkins pipeline tedy nebylo implementováno automatické spouštění testů při každém sestavování aplikace. Testy jsou spouštěny pouze lokálně pomocí příkazu `ng e2e`. Tento příkaz sestaví a nastartuje aplikaci v lokálním prostředí, následně nad nastartovanou aplikací spustí definovanou sadu testů z výše uvedené složky, viz. obrázek 9.3. Adresář je rozčleněn do



Obrázek 9.3: Adresář s testovacími scénáři.

složek podle komponent, které jsou danými testy testovány. Soubory s příponou **.e2e-spec.ts* obsahují definici jednotlivých testovacích scénářů a soubory s příponou **.po.ts* obsahují implementaci jednotlivých metod používaných v testech. Tyto metody implementují pomocí knihovny **Protractor** jednotlivé kroky klikání na elementy ve webovém prohlížeči. Kompletní definice nastavení prohlížeče či základní hodnoty URL adresy aplikace jsou definovány v souboru `protractor.conf.js`. Pro testování je nyní nastaveno používání prohlížeče Chrome, který je brán jako hlavní podporovaný prohlížeč. Pokud jsou testy pouštěny pomocí spouštěče ve vývojovém prostředí IntelliJ Idea, jsou jejich výsledky zobrazeny v uživatelsky přívětivém zobrazení v záložce **Run**, viz. obrázek 9.4. V rámci diplomové práce byly implementovány ná-



Obrázek 9.4: Výsledky testovacích scénářů v IntelliJ Idea.

sledující testovací scénáře:

- Obrazovka uživatelů:
  - Založení uživatele
  - Úprava uživatele - uložení
    - Přiřazení k aplikaci
    - Přidání práv administrátora

  - Úprava uživatele - smazání změn
  - Vyhledání uživatele
  - Resetování hesla uživateli
  - Smazání uživatele
- Obrazovka aplikací:
  - Založení aplikace
  - Úprava aplikace - uložení
  - Úprava aplikace - smazání změn
  - Vyhledání aplikace
  - Smazání aplikace

Testování end-to-end testy slouží k ověření funkčnosti spojení s backendovou částí a zároveň ověřují správné zobrazování dat uživateli. Zároveň jsou definované scénáře využívány na otestování správného chování aplikace v případě neočekávané události, kdy je uživateli potřeba zobrazovat chybové zprávy. Vytvořené testy pokrývají pouze nově implementovanou funkčnost v rámci diplomové práce. Pokrýt testy i zbytek aplikace je plánováno v případné další fázi projektu.

Frontendová část je testována také jednotkovými testy, kdy byl pro každou komponentu definován jednoduchý testovací scénář. Tento scénář ověřuje správnou inicializaci a načtení komponenty. Testy jsou vždy v adresáři dané komponenty a mají suffix **.spec.ts*. Jednotkové testy zde tedy slouží především pro kontrolu syntaktických chyb v definici jednotlivých komponent.

10 Diskuze

Již v průběhu implementace bylo možno pozorovat zrychlení a zkvalitněni celkového procesu správy uživatelů, aplikací a přístupových práv. Tento proces byl zásadně upraven již s dokončením fáze prototypu, kdy po vydání nástrojů pro hromadný import bylo možno předat zakládání a přiřazování práv na supportní oddělení firmy Aimtec. V rámci vydání tohoto prototypu došlo ke specifikaci konvencí, které jsou v rámci aimtec.cloud aplikovány. Tyto konvence byly zaneseny do společné znalostní báze v nástroji OneNote a zaneseny do návodů k obsluze nástrojů. Návod byl nejdříve otestován programátorem, který byl předem zaškolen pro případné řešení problémů, a následně došlo k jeho zpřesnění a předání členům suportního oddělení. Na toto oddělení byl proces převeden již v této fázi. Slabinou tohoto prvního řešení byla nutnost importu dat do každého systému zvlášť, kdy se muselo dbát obezřetnosti na zachování konzistence dat.

Jelikož systém předpokládá práci pouze se systémy, které obsahují data v konzistentním stavu, bylo potřeba provést kompletní analýzu uložených dat v produkčních systémech. Došlo k analýze uložených dat a následně, po diskuzi s projektovými vedoucími, k jejich opravě. Nejčastějším případem nekonzistentnosti dat byla existence záznamu uživatele s emailem, který obsahoval kombinaci velkých a malých písmen. Pro využívání Keycloaku je nutno záznamy zakládat s malými písmeny, jelikož v této knihovně nezáleží, zda je email psaný velkými či malými písmeny. Tato skutečnost byla opravena opětovným založením záznamu ve správném formátu. Následně byla proveden kontrola aplikací a jejich odpovídajících přístupových skupin v Amazon Cognito. V tomto případě byla za referenční data považována ta z Amazon DynamoDB a do Amazon Cognito byly podle těchto záznamů chybějící skupiny nově založeny.

Výsledný produkt, který byl nasazen do produkčního prostředí, umožnil zjednodušení celého procesu. Nyní je potřeba pouze komunikace mezi konzultantem a členem suportního týmu. Tomuto členovi je předán seznam uživatelů, které má do systému založit či seznam změn v přístupových právech. Následně již není potřeba vůbec kontaktovat proškoleného programátora, který se může věnovat jiným aktivitám. Změny jsou prováděny pomocí intuitivního uživatelského rozhraní, k jehož ovládání je sepsána dokumentace ve OneNote a nahráno komentované video. Na videu jsou názorně zobrazeny případy užití a slouží k pohodlnějšímu a efektivnějšímu zaškolování nových uživatelů. Tato forma zaškolování je již v Aimtecu ověřena na několika odliš-

ných projektech. V další fázi projektu je následně možno do procesu přidat i samotné konzultanty, kterým bude stačit pouze nastavit příslušná práva v systému. Při implementaci bylo myšleno také na případnou další fázi projektu, kdy by se celý proces zakládání uživatelů přenesl na administrátory zákazníků. Proto jsou Lambda funkce implementovány tak, aby při každém dotazu byla uživateli poskytnuta funkcionalita, ke které má dostatečná oprávnění.

Hlavními přínosy celé práce jsou především zjednodušení a zkvalitnění celého procesu administrace. Možnost převodu tohoto procesu na zaměstnance bez závislosti speciálního proškolování v systému Amazon DynamoDB a Amazon Cognito. Především také zajištění zachování konzistence dat v obou systémech při provádění změn pomocí uživatelsky přívětivého webového rozhraní.

11 Závěr

Cílem diplomové práce bylo rozšíření stávajícího produktu aimtec.cloud Portál o administrační rozhraní. Toto webové uživatelské rozhraní usnadní celý proces administrace uživatelů a aplikací v cloudovém prostředí, do kterého jsou zákazníci Aimtecu přesouváni. Za tímto účelem se bylo potřeba seznámit s aktuálním řešením správy uživatelů a přístupových práv v multitenantním systému, jak jsou tyto údaje ukládány do systémů a jaké jsou předpoklady pro jeho správné fungování. Zde bylo provedeno několik schůzek s programátory, kteří aimtec.cloud navrhovali a od kterých bylo potřeba tyto informace získat. Zároveň se bylo potřeba seznámit s již existující aplikací, do které bylo implementováno navržené rozšíření.

Důležitou částí práce je také hlubší analýza a porozumění služeb Amazon Web Services, které jsou v rámci celé infrastruktury aimtec.cloud používány (kapitola 3). U těchto vybraných služeb byla provedena jejich celková analýza a prozkoumány jejich možnosti. Bylo zde provedeno také srovnání hlavních poskytovatelů cloudových služeb na trhu a porovnání on-premise a cloud řešení.

Pro provedení implementace rozšíření bylo potřeba analyzovat aktuální procesy, které jsou aplikovány při administraci uživatelů a přístupových práv (kapitola 4). Zde došlo k popsání těchto procesů, které byly postaveny na speciálně zaškoleném programátorovi a jejich přenositelnost na někoho jiného byla takřka nemožná.

Pro provedení implementace bylo potřeba zdokumentovat návrh implementace (kapitola 6). Pro provedení návrhu byly od zákazníků specifikovány funkční požadavky, které musí konečná implementace splňovat. Zároveň zde došlo ke specifikaci nefunkčních požadavků, aby bylo možno výsledné řešení nasadit do již existující infrastruktury. Následně došlo k návrhu potřebných rozšíření datových modelů a první definice uživatelského rozhraní. Tento návrh uživatelského rozhraní byl následně demonstrován externímu specialistovi, který navrhl několik vylepšení.

V další části práce byla provedena kompletní implementace návrhu řešení (kapitola 7). Implementace byla provedena ve třech fázích. V první fázi došlo k ověření proveditelnosti v závislosti na zvolených technologiích. Ve zbylých dvou již došlo k implementaci, kdy výstupem každé fáze byl použitelný produkt. Nejdříve došlo k uvolnění minimálního životaschopného produktu, který vedl již v době vývoje k usnadnění procesu administrace. Na konci poslední váze byl uvolněn již výsledný produkt. Produkt byl ihned

po otestování nasazen do produkčního prostředí a bylo zdokumentováno jeho sestavování a nasazování (kapitola 8).

V předposlední části práce bylo provedeno otestování nové implementace jednotkovými a integračními testy (kapitola 9). Jednotlivá řešení aplikace byla testována zvlášť různými způsoby v závislosti na použitých technologiích.

Poslední část práce obsahuje vyhodnocení přínosů, které přineslo nasazení výsledné aplikace do produkčního prostředí (kapitola 10).

Literatura

[1] *1.3 Getting Started - What is Git?* [online]. Git, 2022. [cit. 2022/05/03]. Dostupné z: https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F.

[2] *Aimtec a.s. - O společnosti* [online]. Aimtec, 2019. [cit. 2022/03/13]. Dostupné z: https://www.aimtecglobal.com/o-spolecnosti/.

[3] *Aimtec Support* [online]. Aimtec, 2021. [cit. 2022/03/13]. Dostupné z: https://www.aimtecglobal.com/aimtec-support/.

[4] *aimtec.cloud EDI* [online]. Aimtec, 2021. [cit. 2022/03/13]. Dostupné z: https://www.aimtecglobal.com/aimtec-cloud-edi/.

[5] *Amazon API Gateway pricing* [online]. Amazon Web Services, Inc., 2022. [cit. 2022/03/14]. Dostupné z: https://aws.amazon.com/api-gateway/pricing/.

[6] *Amazon Cognito* [online]. Rahul Awati, 2021. [cit. 2022/03/28]. Dostupné z: https://www.techtarget.com/searchaws/definition/Amazon-Cognito.

[7] *Amazon Compute Service Level Agreement* [online]. Amazon.com, 2021. [cit. 2022/03/13]. Dostupné z: https://aws.amazon.com/compute/sla/.

[8] *Amazon Web Services grows revenue 40 percent, Microsoft Azure increases share, Google Cloud still loses $840m* [online]. Sebastian Moss, 2022. [cit. 2022/03/28]. Dostupné z: https://www.datacenterdynamics.com/en/news/amazon-web-services-grows-revenue-40-percent-microsoft-azure-increases-share-google-cloud-still-loses-840m/.

[9] *Angular versioning and releases* [online]. Angular, 2022. [cit. 2022/04/18]. Dostupné z: https://angular.io/guide/releases#support-policy-and-schedule.

[10] *AWS Lambda battle 2021: performance comparison for all languages (cold and warm start)* [online]. Aleksandr Filichkin, 2021. [cit. 2022/03/28]. Dostupné z: https://filia-aleks.medium.com/aws-lambda-battle-2021-performance-comparison-for-all-languages-c1b441005fd1.

[11] *AWS SDK for Java Documentation Examples* [online]. 2022. [cit. 2022/04/20]. Dostupné z: https://github.com/awsdocs/aws-doc-sdk-examples/tree/main/java.

[12] *AWS vs. Azure vs. Google Cloud: 2022 Cloud Platform Comparison* [online]. Cynthia Harvey, 2021. [cit. 2022/03/28]. Dostupné z: https://www.datamation.com/cloud/aws-vs-azure-vs-google-cloud/.

[13] Beck, K. *Manifest Agilního vývoje software* [online]. The Agile Alliance, 2001. [cit. 2022/04/09]. Dostupné z: http://agilemanifesto.org/iso/cs/manifesto.html.

[14] *Choosing a Scrum Sprint Length – Shorter Beats Longer* [online]. Levison, Mark, 2019. [cit. 2022/05/09]. Dostupné z: https://agilepainrelief.com/blog/choosing-scrum-sprint-length.html.

[15] *Cold Starts in AWS Lambda* [online]. Mikhail Shilkov, 2021. [cit. 2022/03/28]. Dostupné z: https://mikhail.io/serverless/coldstarts/aws/.

[16] *DCIx* [online]. Aimtec, 2021. [cit. 2022/03/13]. Dostupné z: https://www.aimtecglobal.com/dcix/.

[17] DeBrie, A. *The DynamoDB Book*. Gumroad, 2020.

[18] Deshpande, T. *Mastering DynamoDB*. Packt Publishing, 2017. ISBN 9781783551958.

[19] *Git* [online]. Alexander Schilling, 2021. [cit. 2022/04/09]. Dostupné z: https://alinex.gitlab.io/env/git.

[20] *Global Infrastructure* [online]. Amazon.com, 2022. [cit. 2022/03/13]. Dostupné z: https://aws.amazon.com/about-aws/global-infrastructure/.

[21] *Cloud locations* [online]. Google, 2022. [cit. 2022/03/13]. Dostupné z: https://cloud.google.com/about/locations.

[22] *Google Cloud Platform Service Level Agreements* [online]. Google, 2022. [cit. 2022/03/13]. Dostupné z: https://cloud.google.com/terms/sla.

[23] *How Many Companies Use Cloud Computing in 2022? All You Need To Know* [online]. Jacquelyn Bulao, 2022. [cit. 2022/03/28]. Dostupné z: https://techjury.net/blog/how-many-companies-use-cloud-computing/.

[24] *Increasing performance of Java AWS Lambda functions using tiered compilation* [online]. James Beswick, 2021. [cit. 2022/04/20]. Dostupné z: https://aws.amazon.com/blogs/compute/increasing-performance-of-java-aws-lambda-functions-using-tiered-compilation/.

[25] *Introducing JSON* [online]. 2022. [cit. 2022/03/14]. Dostupné z: https://www.json.org/json-en.html.

[26] *Jenkins Master and Slave Architecture – A Complete Guide* [online]. Arvind Phulare, 2020. [cit. 2022/04/10]. Dostupné z: https://www.edureka.co/blog/jenkins-master-and-slave-architecture-a-complete-guide/.

[27] *Kubernetes vs Docker – What Is the Difference?* [online]. Michael Bose, 2019. [cit. 2022/03/14]. Dostupné z: https://www.nakivo.com/blog/docker-vs-kubernetes/.

[28] *NoSql Data Modeling: 1 to 1, 1 to Many, Many to Many* [online]. Awan, Talha, 2021. [cit. 2022/04/18]. Dostupné z: https://www.techighness.com/post/no-sql-data-modeling-1-to-1-1-to-many-many-to-many/.

[29] Perkins, L. et al. *Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement.* Pragmatic Bookshelf, 2018. ISBN 9781680502534.

[30] *Portál aimtec.cloud - dashboard* [online]. Aimtec, 2019. [cit. 2022/03/13]. Dostupné z: https://www.aimtecglobal.com/aimtec-cloud/.

[31] Sbarski, P. – Nair, A. *Serverless Architectures on AWS, Second Edition.* Manning Publications, 2022. ISBN 9781617295423.

[32] *Souhrn smluv SLA pro služby Azure* [online]. Microsoft, 2022. [cit. 2022/03/13]. Dostupné z: https://azure.microsoft.com/cs-cz/support/legal/sla/summary/.

[33] Stočes, J. Je cloud bezpečný?, Jun 2019. Dostupné z: https://www.aimtecglobal.com/aimagazine/data/folders/aimagazine-33-cz-f8.pdf.

[34] *Using API Gateway stage variables to manage Lambda functions* [online]. Buliani, Stefano, 2015. [cit. 2022/04/20]. Dostupné z: https://aws.amazon.com/blogs/compute/using-api-gateway-stage-variables-to-manage-lambda-functions/.

[35] *What Is Kubernetes? An Introduction to the Wildly Popular Container Orchestration Platform* [online]. John Withers, 2021. [cit. 2022/03/14]. Dostupné z: https://newrelic.com/blog/how-to-relic/what-is-kubernetes.

Seznam zkratek

- UI - User interface (uživatelské rozhraní)
- EDI - Electronic Data Interchange (elektronická výměna dat)
- WMS - Warehouse Management System (systém řízeného skladu)
- MES - Manufacturing Execution System (výrobní informační systém)
- YMS - Yard Management System (systém plánování nakládky a vykládky)
- IaaS - Infrastructure as a Service
- SaaS - Software as s Service
- PaaS - Platform as a Service
- RAM - Random Access Memory
- IDE - Integrated Development Environment (vývojové prostředí)
- SDK - Software development kit
- SSO - Single sign on (jednotný systém přihlašování)
- RDBMS - Relational Database Management System (Relační systém správy databáze)
- API - Application Programming Interfac (Rozhraní pro programování aplikací)
- CRUD - Create, Read, Update, Delete (základní operace nad záznamem v úložišti)
- CLI - Command Line Interface (Příkazový řádek)

Seznam obrázků

Seznam tabulek

Přílohy

A Obsah ZIP souboru

Adresářová struktura a obsah ZIP souboru je popsán v následujícím seznamu:

- `Text_prace` – Složka obsahující text diplomové práce, obrázky a zdrojové kódy textu v LaTeXu.
- `Poster` – Adresář obsahující výsledný poster ve formátu `pdf` a `pub`.
- `Aplikace_a_knihovny` – Složka obsahující zdrojové kódy aplikace.
  - `AimtecCloud` - Složka obsahuje zdrojové kódy Angular aplikace, které vznikly v rámci diplomové práce.
  - `AimtecCloudLambda` - Složka obsahující jednotlivé aplikace nasazované do AWS Lambda funkcí.
  - `Cognito` - Složka obsahující zdrojové kódy implementované knihovny pro práci s Amazon Cognito.
  - `DynamoDB` - Složka obsahující zdrojové kódy implementované knihovny pro práci s Amazon DynamoDB.
- `Readme.txt` – Textový soubor popisující obsah zip souboru a adresářovou strukturu.

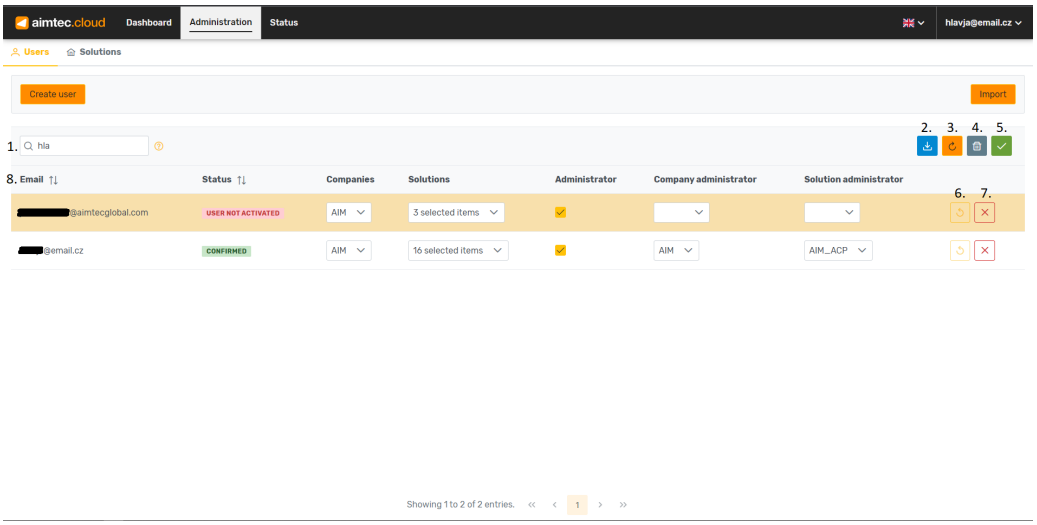
B Seznam implementovaných lambda funkcí

- `DeleteSolution`
- `DeleteUser`
- `GetAdminCompanies`
- `GetAdminSolutions`
- `GetSolutions`
- `GetUserRights`
- `GetUsers`
- `PostImportFile`
- `PostResetPassword`
- `PostSolution`
- `PostSolutions`
- `PostUser`
- `PostUsers`

C Uživatelská příručka

C.1 Administrace uživatelů

Na této stránce je dominantní zobrazení přehledové tabulky, kde jsou zobrazeni aktuálně založení uživatelé. Data v přehledové tabulce jsou modifikovatelná a lze uživatelům přidávat či odebírat přístupy do daných aplikací či z nich udělat administrátora viz. obrázek C.1.



Obrázek C.1: Přehledová tabulka uživatelů.

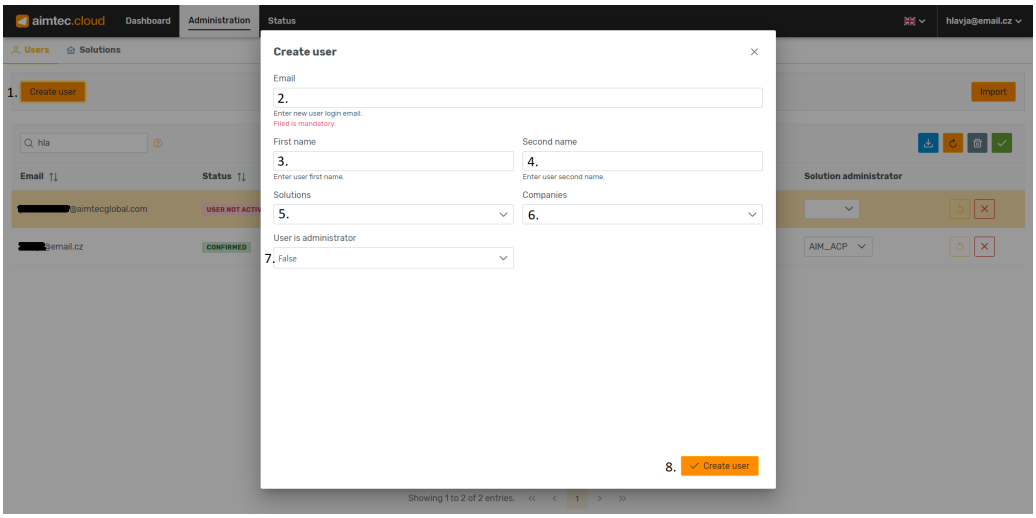
1. SEARCH POLE - slouží k vyhledávání v datech tabulky, vyhledáváme přes atributy: email, solutions a podporujeme speciální keyword: 'changed' který vyfiltruje jen záznamy na kterých jsou neuložené změny.
2. EXPORT DATA - vyexportuje aktuálně vyfiltrovaná data ve formátu *.csv*.
3. REFRESH DATA - obnoví zobrazovaná data.
4. DISCARD CHANGES - zahodí neuložené změny na záznamech.
5. SAVE CHANGES - uloží změny do databáze.
6. RESET PASSWORD - resetuje heslo uživateli.
7. DELETE USER - vymaže uživatele.
8. TABLE - přehledová tabulka.

- EMAIL - primární klíč uživatele sloužící k jeho identifikaci, nelze upravovat, pokud se uživatel založí se špatným emailem je potřeba ho smazat a vytvořit znovu správně.
- STATUS - status ověření uživatele.
  - USER NOT ACTIVATED - uživatel nepotvrdil emailovou adresu.
  - CONFIRMED - uživatel potvrdil emailovou adresu.
  - NOT IN COGNITO - uživatel není založen v systému Amazon Cognito.
- COMPANIES - jedná se o vícehodnotový výběrový box, kde lze uživatele přiřadit k příslušným zákazníkům.
- SOLUTIONS - sloupec zobrazující všechny aplikace ke kterým má uživatel právo a zobrazují se mu na dashboard stránce. Jedná se o vícehodnotový výběrový box ve kterém lze vyhledávat a přiřazovat či odebírat práva na danou solutionu.
- ADMINISTRATOR - jedná se o příznak, že daný uživatel může přistupovat do administračního rozhraní.
- ADMIN COMPANIES - přiřazení práv spravovat uživatele dané společnosti (uživateli se zobrazují jenom uživatelé z dané společnosti). Je zde jedna specialita a tou je přiřazení uživatele do admin společnosti AIM - toto přiřazení dává uživateli kompletní přístup do administrace jelikož je uživatel považován za Aimtec administrátora.
- ADMIN SOLUTIONS - jedná se o nastavení práv uživateli pro danou aplikace. Používáno pokud má zákazník vícero aplikací a nechceme mít uživatele s admin právy přes celou společnosti ale jen pro danou aplikaci aby nemohl měnit data z ostatních.

C.2 Založení uživatele

Uživatel je zakládán pomocí modálního okna, viz. obrázek C.2, po kliknutí na tlačítko **Create user**. Zrušení vytváření uživatele pomocí zavření okna křížkem. Odeslání požadavku na vytvoření uživatele pomocí spodního tlačítka **Create user** v modálním okně.

1. CREATE USER - tlačítko pro zobrazení modálního okna.
2. EMAIL - povinný atribut sloužící jako primární klíč pro identifikaci uživatele.



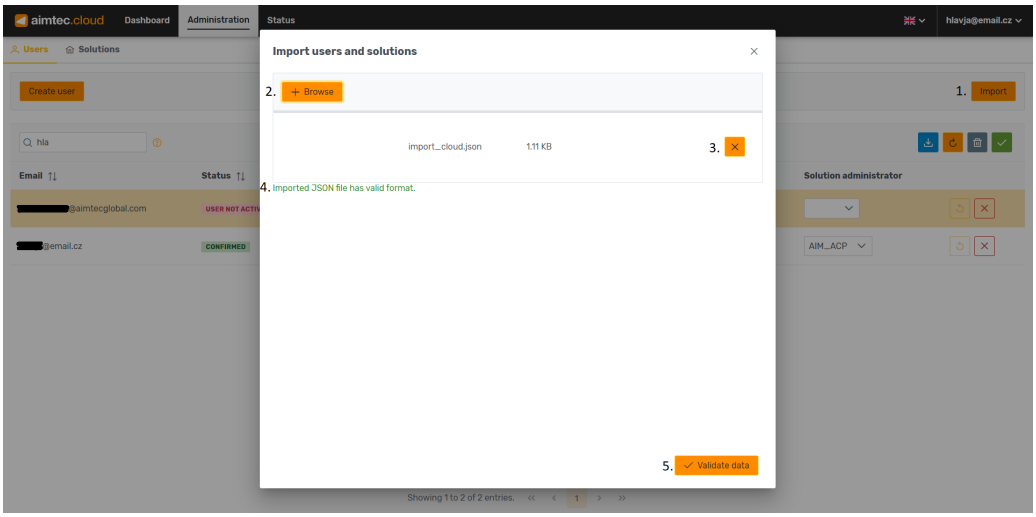
Obrázek C.2: Vytváření nového uživatele pomocí modálního okna.

3. FIRST NAME - nepovinný atribut, křestní jméno uživatele.
4. SECOND NAME - nepovinný atribut, příjmení uživatele.
5. SOLUTIONS - nepovinný atribut, políčko slouží pro přiřazení přístupu uživateli k dané aplikaci.
6. COMPANIES - nepovinný atribut, přiřazení uživatele k zákazníkovi.
7. USER IS ADMINISTRATOR - defaultně nastavený na False. Jedná se o atribut určující zda je uživatel administrátor a může přistupovat k administraci.
8. CREATE USER - tlačítko pro odeslání formuláře a založení uživatele.

C.3 Import ze souboru

Pomocí importu ze souboru *.json*, který je prováděn v modálním okně viz. obrázek C.3, lze zakládat nové aplikace či uživatele, případně hromadně přidávat aplikace k již existujícím uživatelům.

1. IMPORT - tlačítko pro otevření modálního okna.
2. BROWSE - otevření systémového prohlížeče pro vybrání souboru.
3. X - tlačítko pro odebrání souboru z importu.
4. MESSAGE - validační hláška nahraného *.json* souboru, kdy je zkontrolována syntaxe.



Obrázek C.3: Modální okno pro import ze souboru.

5. VALIDATE DATA/IMPORT DATA - tlačítko pro validaci a import dat do systému.

    - VALIDATE - validace importovaných dat pomocí backendu. Zda není importována již existující aplikace, zda všechny aplikace uvedené u uživatelů existují.
    - IMPORT - konečný import dat do systému. Pokud spadne import aplikací, následný import uživatelů není proveden, jelikož se očekává ve většině případů hromadný import při zakládání nové aplikace a ta by v systému chyběla.

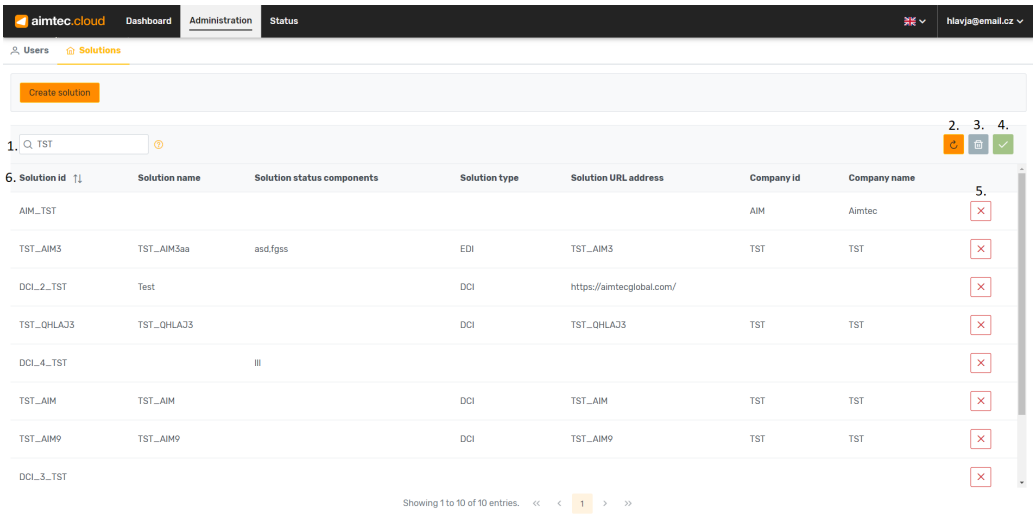
C.4 Administrace aplikací

Na stránce je dominantní zobrazení přehledové tabulky, kde jsou zobrazeny aktuálně založené aplikace viz. obrázek C.4. Data v přehledová tabulce lze upravovat kliknutím na daný atribut záznamu, kdy dojde k přepnutí zobrazení atributu do textového pole.

1. SEARCH POLE - slouží k vyhledávání v datech tabulky, vyhledáváme přes atributy: solution id, solution type a podporujeme speciální keyword: 'changed' který vyfiltruje jen záznamy na kterých jsou neuložené změny.

2. REFRESH DATA - obnoví zobrazovaná data.

3. DISCARD CHANGES - zahodí neuložené změny na záznamech.



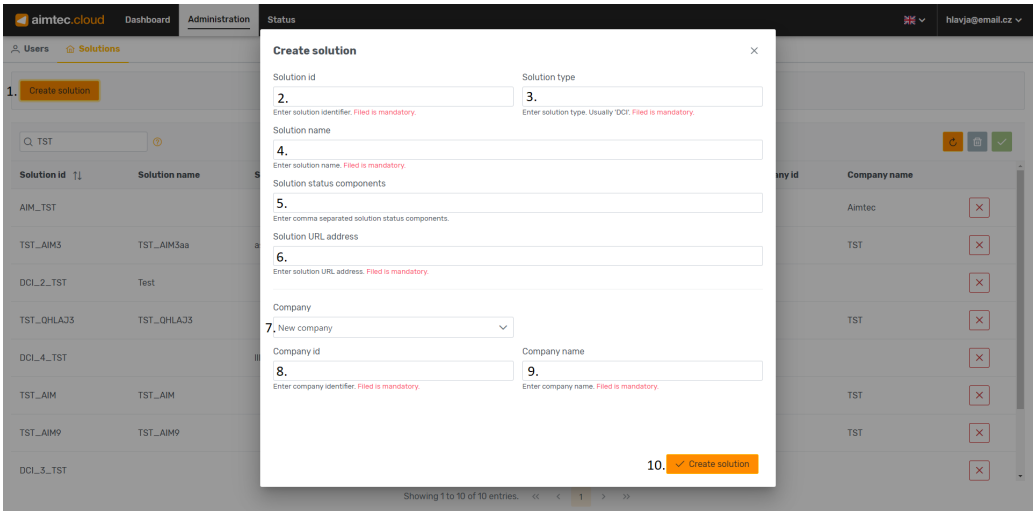
Obrázek C.4: Přehledová tabulka aplikací.

4. SAVE CHANGES - uloží změny do databáze.
5. DELETE SOLUTION - vymaže aplikaci ze systému (dojde k jejímu odstranění i v přiřazení přístupových práv).
6. TABLE - přehledová tabulka
    - SOLUTION ID - identifikátor dané aplikace, nelze měnit.
    - SOLUTION NAME - název aplikace (jedná se o podrobnější název, který se bude zobrazovat zákazníkovi na dashboard obrazovce).
    - SOLUTION STATUS COMPONENT - IT poznámky o službách aplikace.
    - SOLUTION TYPE - typ aplikace (většinou DCI).
    - SOLUTION URL - url kam bude zákazník přesměrován pokud na aplikaci klikne na dashboardu.
    - COMPANY ID a COMPANY NAME - doplňuje se automaticky podle prvních třech písmen ze SOLUTION ID (nastavené jmenné konvence).

C.5 Založení aplikace

Aplikace je zakládán pomocí vyskakovacího okna viz. obrázek C.5 po kliknutí na tlačítko **Create solution**. zrušení vytváření aplikace pomocí zavření modálního okna křížkem. Odeslání požadavku na vytvoření uživatele pomocí

spodního tlačítka **Create solution**. Všechny povinné atributy formuláře nutné pro založení aplikace jsou zvýrazněny červeným upozorněním.



Obrázek C.5: Vytváření nové aplikace pomocí modálního okna.

1. CREATE SOLUTION - tlačítko pro zobrazení modálního okna.
2. SLUTION ID - textové pole kde vyplňujeme ID aplikace podle zavedených jmenných konvencí.
3. SOLUTION TYPE - v 99% případech bude 'DCI'.
4. SOLUTION NAME - lidské pojmenování aplikace které se bude zobrazovat zákazníkovi na dashboardu.
5. SOLUTION STATUS COMPONENT - IT popis komponent které jsou na aplikaci nasazené, není povinné.
6. SOLUTION URL - URL aplikace kam povede proklik z ikonky na dashboardu. Url kde běží aplikace (musí dodat zadavatel požadavku na založení záznamu).
7. COMPANY - pro evidenci zákazníků v DynamoDB a jejich případnému následnému použití, kód zákazníka by se měl shodovat podle konvencí s prvními třemi písmeny Solution ID.
    - NEW COMPANY - pokud zakládám aplikaci pro zákazníka, kterého nemám na výběr, je potřeba vyplnit následující dvě pole.

- EXISTING COMPANY - pokud zakládám aplikaci pro již existujícího zákazníka (mohu ho vybrat ze selectboxu) tak jsou následující pole automaticky vyplněna.

8. COMPANY ID - třípísmenný kód zákazníka.
9. COMPANY NAME - lidský název zákazníka.
10. CREATE SOLUTION - tlačítko pro odeslání formuláře a založení aplikace.