FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Podpůrné služby pro online hru Inpemo
Student:	Bc. Petr Nohejl
Vedoucí:	Ing. Oldřich Malec
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce zimního semestru 2021/22

Pokyny pro vypracování

Pro online hru více hráčů Inpemo je zapotřebí realizovat webové podpůrné služby - serverovou část infrastruktury, ke které se připojují klienti. Analyzujte již existující "krabicová" řešení těchto služeb. Na základě provedené analýzy navrhněte systém poskytující tyto služby pro hru Inpemo. Při návrhu spolupracujte s Bc. Michalem Šveigrem, který bude připravovat v rámci jeho dipl. práce hru samotnou. Dbejte na snadné rozšíření systému a jeho propojení s dalšími hrami. Systém musí počítat s velkým množstvím klientů ve stejném čase a být škálovatelný.
Implementujte minimálně následující služby:
1. Messaging služba, která zajišťuje asynchronní komunikaci (notifikace klientů).
2. Matchmaking služba, zajišťující vytváření instancí her a napojení klientů do těchto instancí.
Diskutujte výhody/nevýhody vlastního řešení online systému a zdůvodněte výběr použitých technologií. Prototyp otestujte Vámi navrženými testy. Společně s Bc. Michalem Šveigrem zajistěte vydání alfa verze.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 8. června 2020

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Podpůrné služby pro online hru Inpemo

Bc. Petr Nohejl

Katedra softwarového inženýrství
Vedoucí práce: Ing. Oldřich Malec

5. ledna 2021

Poděkování

Rád bych poděkoval svému vedoucí panu Ing. Oldřichovi Malcovi za podporu a časté konzultace při vedení této práce. Dále bych chtěl poděkovat za především morální podporu Bc. Lukáši Lojíkovi a Bc. Michalu Šveigrovi. Obrovské díky za podporu patří také mým rodičům, kteří mě podporovali v rámci celého studia. Jako poslední bych chtěl zmínit a poděkovat za podporu a především trpělivost kolegům v práci.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 5. ledna 2021 . . . . . .

Odkaz na tuto práci

Nohejl, Petr. *Podpůrné služby pro online hru Inpemo*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Abstrakt

Rešeršní část práce se zabývá online službami integrovatelnými především do online her pro více hráčů. Také jsou v této části diskutovány výhody a nevýhody implementace vlastních online služeb. V praktické části je proveden návrh a implementace některých online služeb integrovaných do online hry Inpemo. Jedná se o autentizační, registrační, matchmaking a notifikační služby. Tyto služby jsou implementovány pomocí Javascriptového runtime Node.js a jsou navrženy a implementovány s důrazem na jejich znovupoužití, škálovatelnost a rozšiřitelnost. V práci je též popsáno nasazení těchto služeb pomocí Amazon Web Services. Uvedeny jsou také návrhy dalších služeb, které však nejsou implementovány. Implementované služby jsou závěrem také testovány.

Klíčová slova online služby pro počítačové hry, služba Matchmaking, notifikační služba, Node.js, Amazon Web Services

Abstract

The survey part of this thesis is chiefly focused on online services integrable mostly to multiplayer online games. The advantages and disadvantages of the implementation of custom online services are also discussed in this part of the thesis. Some of the online services integrated into the Inpemo game are designed and implemented in the constructive part of the thesis. Namely, we aim at authentication, registration, matchmaking, and notification services. These services are implemented by Javascript run-time Node.js and are designed and also implemented in a way that they can later be scaled, extended, and reused. Then deployment of these services using Amazon Web Services is described. There are also other services designed but not implemented. Services that have been implemented were also tested at the end of this thesis.

Keywords online services for PC games, Matchmaking service, notification service, Node.js, Amazon Web Services

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Úvod

Počítačové hry a jejich vývoj prochází během několika posledních dekád neustálým vývojem. Na hry jsou kladeny stále vyšší nároky. Zvyšují se i počty lidí, kteří na těchto hrách pracují, týmy vývojářů her v dnešní době zahrnují desítky až stovky lidí v mnoha různých oborech. Jedná se často o grafiky, animátory, designéry, zvukaře, skriptéry, programátory a další. Programátoři, kteří na dané hře pracují, se podílí nejen na vývoji samotné hry, ale často také na systémech spojených se hrou. Ať už jsou to systémy interní pro samotné vývojáře, a nebo externí, jako jsou třeba služby, které podporují hraní online. Samozřejmě ne všechny hry tyto externí služby potřebují (příkladem jsou hry pro jednoho hráče). Jak už však plyne z názvu této práce, jsme vázáni hrou Inpemo, což je online hra pro více hráčů a online služby jsou k jejímu hraní nezbytné. Hráči by bez těchto služeb nebyli schopni mezi sebou komunikovat, ani spolu hrát.

Funkcionality jako chat, vytváření skupin nebo hledání her jsou dnes u těchto typů her brány jako samozřejmost. Žádná z těchto funkcionalit však typicky nebývá implementována přímo ve hře, ale je vyvíjena odděleně v rámci online služeb. Tyto služby bývají do hry integrovány. V této práci se budeme zabývat převážně těmito službami a funkcionalitami, které tyto služby nabízí. Již z tohoto krátkého uvedení do problematiky můžeme pochopit důležitost těchto služeb v rámci online her více hráčů.

Cíle práce

Cílem rešeršní části práce je vysvětlit fungování online služeb ve hrách, uvést jakými způsoby se dají online služby do her implementovat a poukázat na rozdíly mezi jednotlivými přístupy. Za cíl je také kladeno prozkoumat potřeby online služeb pro hru Inpemo a v rámci uvedených poznatků rozhodnout o způsobech implementace jednotlivých služeb.

Návrh jednotlivých služeb a komunikace mezi nimi je další spíše již prak-

tický cíl. Po navržení celé sady služeb máme za cíl také její implementaci a nasazení do prostředí ve kterém mohou být služby otestovány. Tím se dostáváme k poslednímu cílu této práce, kterým je právě testování těchto služeb.

Struktura práce

Práce je strukturována do čtyř logických celků. První z nich s názvem „Online služby pro počítačové hry“ poskytuje informace všeobecně k online službám a také k souvisejícím distribučním službám pro počítačové hry. Dále také uvádí možnosti integrace online služeb do hry a poukazuje na rozdíly mezi možnými přístupy.

Další část nesoucí název „Analýza a návrh online služeb pro hru Inpemo“ představuje detailněji hru Inpemo a její požadavky z hlediska online služeb. Dále pak v této části uvádíme návrh jednotlivých služeb, zvolených technologií a také navrhujeme způsob komunikace mezi jednotlivými službami.

Třetí část práce „Realizace online služeb pro hru Inpemo“ uvádí implementační detaily jednotlivých služeb, popisuje problémy, které při vývoji vznikaly a také popisuje nasazení jednotlivých služeb do prostředí, ve kterém je dále možné je testovat.

V poslední části práce „Testování online služeb pro hru Inpemo“ popisujeme navržené a provedené druhy testů nad jednotlivými online službami, které byly v předchozí části implementovány.

Kapitola 1

Online služby pro počítačové hry

V této kapitole se seznámíme s online službami pro počítačové hry. Řekneme si, k čemu hry potřebují tyto služby využívat. Prozkoumáme existující online služby pro hry a zhodnotíme jejich výhody a nevýhody. Též si uvedeme motivaci k tomu, vytvářet vlastní služby v rámci vývoje počítačových her. Probereme, jakou infrastrukturu je zapotřebí vytvořit před implementací samotných online služeb a s čím je potřeba počítat při reálném využití.

1.1 Rozvoj a úloha online služeb ve hrách

Ruku v ruce se vznikem internetu a jeho rozšířením do širšího povědomí na sklonku milénia vznikají online hry různých žánrů. Jedna z prvních významných her, kterou můžeme zařadit do kategorie *online hra pro více hráčů* je Doom, vydaná v roce 1993. Tato hra, jako ještě mnoho dalších po ní, má postavený model hry pro více hráčů na principu komunikace po lokální síti (LAN). Krátce po vydání však bylo možné hru hrát online přes DWANGO, což byla jedna z prvních online služeb pro hraní počítačových her. Používání služby DWANGO a kontaktování herního serveru vyžadavalo vytočení telefonního čísla. Tato technologie však byla používána jen krátce a zanedlouho ji zastínilo právě rozšíření internetu. I s rozvojem internetu jsou však online služby z počátku omezené. Většina her používá hostování u jednoho z připojených hráčů. Tím, že internet používalo stále více lidí, kteří měli stále kvalitnější připojení, začíná se rozmáhat digitální distribuce her. K tomu jsou zakládány distribuční služby. Více v [1].

Jedním z nejznámnějších příkladů distribuční služby je bezpochyby Steam. Tyto služby měly z počátku především záměr prodávat a poskytovat hráčům hry online. To však přešlo v mnoha případech ke komplexním portálům, kde pomocí online služeb můžete hry nejen hrát ale v rámci nich chatovat s přáteli, zakládat různé skupiny, zakládat herní instance, s pohledu vývojáře pak hry publikovat, sbírat různé statistiky a mnoho dalšího. Dnes je již drtivá většina

her publikována na některé s distribučních platforem a i hry, které nejsou pro více hráčů, je možné propojit s různými online službami, které pak v klientské aplikaci mohou ukazovat hráči například kolik času ve hře strávil, jaký je jeho pokrok ve hře a podobně.

U online her je využití online služeb dnes již téměř neodmyslitelnou součástí. Nejznámější online distribuční služby mají dnes řádově stovky milionů zákazníků po celém světě a bez nich si svět her snad již ani nejde představit.

1.2 Detailní pohled na distribuční služby pro počítačové hry a jejich online služby

V předchozí sekci 1.1 jsme si naznačili, co distribuční portál či online služba představuje, pojďme si je ale hned zezačátku formálněji definovat. Dále se již zaměříme na jednotlivé distribuční služby a projdeme si nejdůležitější online služby, které nabízejí.

1.2.1 Digitální distribuční služby pro počítačové hry

Jedná se o software, který může být podporován na jedné či více platformách (Windows, Linux, Xbox, a jiné). V herním průmyslu tyto služby z pohledu vydavatelů her slouží k prodeji her, jejich propagaci a k dalším činnostem podporujícím distribuci a hraní hry buď přímo v rámci distribuční služby, a nebo samostatně. Z pohledu hráče či zákazníka slouží distribuční služby k nákupu her v digitální podobě, případně přímo k hraní her, k činnostem spojených s hraním či k administračním úkonům spojeným se hrou. Mezi nejznámější distribuční služby v tomto odvětví patří Steam, Epic Store, Battle net, Origin, GoG a mnoho dalších.

1.2.2 Online služby pro podporu hraní počítačových her

Obecně definice online služeb je mnohem rozsáhlejší, my se však zaměříme pouze na takové služby, které slouží k podpoře hraní počítačových her a bývají často integrovány do digitálních distribučních služeb.

Online služby pro podporu počítačových her nám poskytují informace přes internet. Mohou se lišit od jednoduchých až po velmi komplexní. Ať jsou placené nebo zdarma, všechny poskytují jednu nebo více funkcionalit více či méně zásadních k hraní počítačových her po internetu.

Také zmiňme, že existuje mnoho služeb, které implementují jednotlivé online služby, ale nejsou součastí jakékoli distribuční služby, ty si dále v textu také prozkoumáme a uvedeme, čím se tato řešení liší od služeb integrovaných přímo do distribučních služeb. Jako příklad samostatné platformy, která poskytuje online služby, uveďme server Nakama nebo engine Photon.

1.2.3 Distribuční služba Steam a sada služeb Steamworks

Jako první si uveďme nejznámější a nezrozšířenější distribuční službu vůbec - Steam. Ten dominuje v oblasti počítačových her především na plattformě Windows, pro kterou byl v roce 2003 původně vytvořen. Jeho rozsah je však dnes mnohem větší, Steam můžeme najít na platformách macOS, Linux, Android a dokonce vyvíjí vlastní operační systém založený na linuxové distribuci debian nazvaný SteamOS.

Vraťme se však zpět k online službám. K těm se jako vývojáři dostaneme přes rozhraní Steamworks, což je API, pomocí kterého můžeme do hry, která musí být publikovaná na Steamu, integrovat všechny její nabízené online služby. Steamworks poskytuje authentizační nástroje jak pro peer-to-peer hry více hráčů, tak pro hry více hráčů s dedikovanými servery. Poskytuje služby jako matchmaking, službu pro správu přátel a skupin, službu statistik, službu achievements, chatovací službu, službu integrované zvukové komunikace, online uložiště Steam Cloud, microtransakční službu pro nákupy ve hře a mnoho dalších. Některé tyto služby si v této sekci rozvedeme, zejména ty, které jsou potřebné pro hru Inpemo.

Předtím než rozvedeme jednotlivé online služby Steamu, zmíníme ještě podmínky publikování her. Steam je jedna z mála distribučních služeb, která prakticky bez omezení dovoluje vývojáři hru publikovat. Samozřejmě existují pravidla ošetřující extrémní případy, například obsah, který by jakýmkoli způsobem zneužíval děti (více v [2]). K publikování her na Steamu slouží platforma Steam Direct, kde je možné jako vývojář projít publikačním procesem. Do doby, kdy se rozhodnete hru publikovat, je však možné využívat Steamworks API a tedy jednotlivé služby zdarma (až na výjimky jako mikrotransakční službu, která poskytuje In-game transakce pomocí Steam Wallet). Pro publikování hry je pak třeba zaplatit nejprve publikační poplatek, který činí 100 USD, a dále je potřeba počítat s tím, že hry, které jsou placené, zaplatí kromě daní také provizi 30% z každé prodané kopie. Jedná-li se o prodeje převyšující 10 milionů USD, provize mírně klesá na 25% (více lze nalézt v [3]).

Dále je také důležité zmínit možnou integraci do projektu v Unreal Engine, který obsahuje rozhraní pro různé online subsystémy, z nichž jeden je Online Subsystem Steam (více zde [4]), což značně zjednodušuje integraci Steamworks API SDK do projektu. Bohužel ne všechny funkcionality jsou v tomto subsystému podporovány. Poslední věc související s integrací a používáním služeb ve hře je podmínka Steam autentizace, tedy Steam služby, kterou můžeme používat pouze po založení Steam účtu, což je poměrně razantní omezení, které fakticky zabraňuje použití těchto online služeb pro jinou distribuční službu než je Steam.

Nyní si předtavíme některé důležité služby Steamworks, které bude zapotřebí implementovat ve hře Inpemo, pro její plnou funkčnost. Jelikož steam poskytuje pouze API k jednotlivým službám, nemůžeme bohužel plně prozkoumat architekturu a detaily návrhu, ale i z dokumentace a samotného API

získáme přehled o možnostech využití.

Steam matchmaking & lobbies
Matchmaking ve Steamu je postaven na konceptu lobby. Lobby je entita, která žije na back-endových serverech Steamu a dá se přirovnat k chatovací místnosti. Uživatelé mohou vytvořit novou lobby nebo se připojit k lobby. Na rozdíl od jiných architektur matchmakingu, které přiřazují hráče společně do her dle jejich atributů či metadat, matchmaking ve Steamu hledá existující lobby, dle daných lobby parametrů.

Síťové služby a sokety
Slouží pro navazování peer-to-peer spojení a přenášení zpráv tímto navázaným spojením. Je potřeba zmínit, že se nebavíme pouze o zprávách mezi klienty(hráči), ale obecně o notifikacích, které může server hráčům zasílat, tuto službu tedy můžeme označit za notifikační. Steam má dvě hlavní rozhraní, které se mohou využívat. První je `ISteamNetworkingSockets`, což je nízkoúrovňové rozhraní, které před přenosem dat musí navázat komunikační relaci nebo dočasné spojení, pomocí soketu. Druhé rozhraní `ISteamNetworkingMessages` je spíše rozhraní vyšší úrovně, kde se zadává příjemce při každém odeslání zprávy.

Služba Friends
Jedná se o službu, která zprostředkovává přístup k informacím o jednotlivých uživatelích a interakci se Steam Overlay, což je součást Steamworks, kterou lze ve spuštěné hře aktivovat. Umožňuje uživateli přístup do seznamu přátel, ale také přístup do webového prohlížeče či chatu.

Závěrem k distribuční službě Steam můžeme říci, že je za léta svého působení velmi dobře vybavená a prakticky vše, co většina multiplayer her potřebuje, je v ní možno pomocí jejích online služeb dosáhnout. Pokud tedy hra nemíří na více distribučních platforem, či na jinou platformu než Steam, je to z hlediska dostupnosti a jednoduchosti integrace ověřená a bezpečná volba.

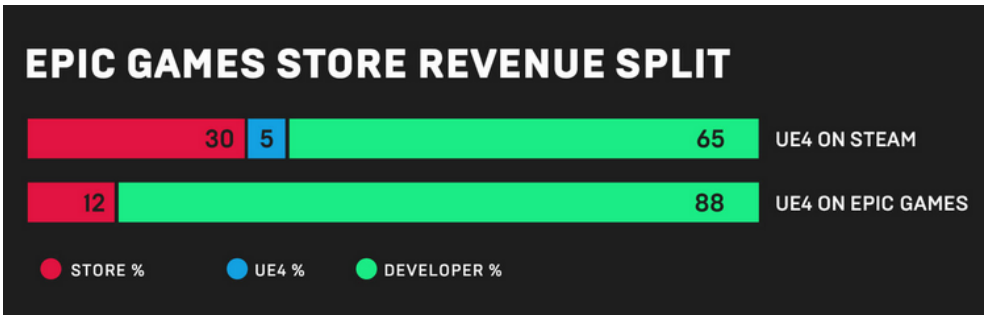
1.2.4 Distribuční služba Epic Store a sada služeb Epic Online Services

Další velice známá distribuční služba, kterou si představíme, je Epic Store. Jedná se o mnohem mladší službu než je Steam, byla uvedena na konci roku 2018. Za své krátké působení však nabrala obrovské množství aktivních uživatelů, které se pomalu blíží k hodnotám Steamu. Jedná se o desítky milionů aktivních hráčů měsíčně, s maximem 13 milionů aktivních hráčů v jeden moment. Pokud to chceme porovnat se Steamem, u něj činí maximum aktivních hráčů v jeden moment něco málo přes 20 milionů. Více lze nalézt v [5] a v [6].

Společnost Epic Games zaštiťující distribuční službu Epic store v létě roku 2020 vydala sadu služeb Epic Online Services (EOS), která si klade podobné cíle jako Steamworks. Předtím než začneme procházet jednotlivé detaily

o EOS, je nutné zmínit, že všechny uvedené informace k EOS jsou uváděny v období (podzim 2020), kdy EOS prochází aktivním vývojem a mnoho z jeho funkcionalit zatím není podporováno nebo je v experimentální fázi. Pokud tedy porovnáme EOS se Steamworks, tak hlavní výhoda spočívá v možnosti autentizace pomocí vlastního OpenID providera, či přihlášení a následném propojení Epic účtů s platformami třetích stran. Mezi takové platformy patří Google, Xbox live, Apple a mimo jiné i Steam. Tím jinými slovy umožňuje použití svých služeb pro distribuci například na Steamu. Propojování účtů s Epic účtem pak dovoluje v omezené míře mezi-platformí využití. Dovoluje dokonce omezené využití služeb bez Epic účtu s vytvořením pouze EOS účtu.

EOS jako takové jsou zdarma. Samozřejmě, pokud bychom je použili společně s distribuční službou Steam, stále by platila 30% provize Steamu, pokud bychom se však rozhodli publikovat hru na Epic Store, tak provize by byla výrazně menší, a to 12%. Vizualizaci včetně provize za využití Unreal Enginu 4 (UE4) najdete na obrázku 1.1.



Obrázek 1.1: Provize při publikování na Steam/Epic s použitím UE4. Článek s uvedeným obrázkem vizte v [7].

Předtím než budeme popisovat jednotlivé služby EOS, zmiňme ještě pravidla pro publikování na distribuční službě Epic Store, kde jde požádat o publikaci, je ale nutné schválení ze strany Epic Store a to za podmínek, které nejsou blíže specifikovány.

Stejně jako u Steamworks pojďme podrobněji popsat jednotlivé služby, které je potřeba implementovat do hry Inpemo. Služby jsou nejsnáze přístupné přes EOS SDK, které je vydáno v několik jazycích (například C++ a C#).

EOS lobbies & sessions

Jedná se o dvě rozhraní, která zajišťují podobné funkcionality, a to sdružování hráčů, avšak za jiných okolností. Lobby obvykle uživatelé vytvářejí (nebo se k nim připojují) za účely chatování s ostatními uživateli, vytváření týmů, vybírání možností před hrami nebo kvůli společnému čekání na připojení dalších hráčů.

Sessions často spíše spravují data specifická pro hru a pro instance her. Pro vytvoření instance hry je tedy nutné vytvořit session, která ponese mimo jiné údaje o instanci hry a jejích hráčích.

Matchmaking, který s lobbies a sessions úzce souvisí, se implementuje pomocí vyhledávání mezi lobbies, potažmo přímo mezi sessions. Opět jako u Steamu jde však spíše o systém vyhledávání existujících lobbies/-sessions, ne o přiřazování přímo dle hráčských atributů. Více informací lze nalézt v [8] nebo v [9].

Síťové služby

EOS podobně jako Steam nenabízejí téměř žádný náhled za své API z hlediska notifikací. V SDK pro asynchronní funkce tedy existují callback funkce, které zasílají notifikace o úspěchu či neúspěchu dané operace. Podobně jako u Steamu, kde jsme si představili rozhraní ISteamNetworkingMessages a ISteamNetworkingSockets má EOS rozhraní NAT P2P, které umoňuje bezpečné (DTSL) peer-to-peer připojení mezi dvěma přihlášenými hráči.

Služba Friends

Opět velmi podobně jako u Steam, můžeme u EOS najít Friends rozhraní, které nám více méně dovoluje ty samé funkcionality. Operace se seznamem přátel (zobrazení, cachování, změny), zobrazování stavu[1]přátel.

Epic Online Services nabízí podobné služby jako Steamworks, s tím, že některé - jako třeba Chat, zatím chybí, ale jejich představení je plánováno na rok 2021, takže kdy se tyto služby do EOS doplní, je jen otázkou času. Zároveň fakt, že EOS nejsou rovnou vázány na Epic Store a jsou zdarma, značně zvyšuje možnost jejich využití. Při publikaci na Epic Store v době, kdy neexistoval EOS, museli jít tvůrci při vývoji buď cestou nějakého serveru pro online služby, což přináší další problémy s integrací do herního enginu, či použít nějaké vlastní řešení, což zahrnuje jak spoustu času stráveného implementací, tak potřebné finanční prostředky na infrastrukturu.

1.2.5 Server pro online služby Nakama

Jedná se o server, pomocí kterého je možné zřídit si vlastní online služby bez potřeby jejich implementace. Nepatří k žádné distribuční službě, tedy jejím použitím nejsme vázáni na publikování hry například na Steamu. Nakama je poskytována ve dvou variantách a to *Open source* a *Nakama Enterprise*.

[1]Stav, označován také jako status. Dva základní stavy jsou Online a Offline. Uživatel však může mít nastavený jiný vlastní stav.

Nejprve se pojďme podívat na *Open source* verzi a její omezení. Jedná se o on-premises[2]řešení, které není škálovatelné, je tedy možné mít pouze jednu běžící instanci, což samozřejmě velice omezuje možnosti využití této varianty. Pokud server neočekávaně selže, tak po dobu svého zotavování bude nedostupný, takže pokud bude jeho součástí například i služba matchmaking, potom nebude pro hráče možné se po danou dobu připojovat do nových her. Zároveň server běží pouze na jednom CPU a má omezenou velikost databáze. Tato verze se tedy může jevit jako do praxe těžko použitelná, zmiňme však, že nemá žádná omezení na úrovni instance serveru, tedy lze využívat všechny dostupné služby.

Nakama Enterprise dle očekávání doplňuje všechny nevýhody a omezení *Open source* verze v různých variantách škálování od 600 po téměř 5000 USD měsíčně. K tomu samozřejmě nabízí podporu nasazení serveru do produkčního prostředí, včetně orchestrace[3], škálování (Kubernetes[4]) a nasazení pomocných nástrojů pro kontrolu zdraví clusteru.

Co se týče jednotlivých služeb, Nakama poskytuje prakticky veškeré služby, které jsou pro většinu online her potřeba. Jako příklad uveďme friends, chat, matchmaking, notifikace, skupiny a klany, statistiky, pořadí, autentizace a další služby. Nakama má velice podrobnou dokumentaci a tím, že je open source, je možno najít implementace jednotlivých služeb přímo na githubu. Ještě zmiňme, že jelikož má Nakama také autentizační službu, pracuje s databázovým clusterem CockroachDB založeným na PostgreSQL.

Obdobně jako Steam či EOS má i Nakama SDK v jazyce C++ (a mnoha dalších). Samotné SDK však neřeší komplexní integraci do UE4 Online subsystému. Na rozdíl od Steamu, který už má Online subsystém vytvořený, je nutné při použití Nakama Online subsystém implementovat. Více o integraci s UE4 v [10].

Pojďme si tedy popsat služby serveru Nakama, které by mohly být použity pro hru Inpemo.

Služba Friends

Poskytuje funkce zobrazování, přidávání, odebírání, blokování přátel. Služba navíc implementuje sociální graf, takže podobně jako třeba na síti Facebook zde funguje vyhledávání společných přátel.

Služba Notifikací

Notifikace jsou zprostředkovány pomocí callback funkcí obdobně jako

[2]On-premises je řešení provozované na vlastních prostředcích ve vlastních prostorách. Typicky se může jednat o software, který je nainstalovaný na serverech firmy, která daný software používá.

[3]Orchestrace popisuje automatickou koordinaci a řízení komplexních počítačových systémů, middleware a služeb.

[4]Kubernetes je open-source systém sloužící k orchestraci kontejnerů, automatizaci nasazování, škálování a správu aplikací.

u EOS nebo Steamworks, interně pak pro komunikaci s klientem Nakama používá websockety.

Služba Matchmaker

Nakama Matchmaker umožňuje hráčům najít soupeře a spoluhráče pro zápasy a to tak, že udržuje skupinu (pool) uživatelů, kteří aktuálně hledají soupeře, a sleduje požadavky, které na službu Nakama Matchmaker od uživatel přicházejí, a poté tyto uživatele seskupuje na základě kritérií v zaslaných požadavcích.

Každý matchmaker požadavek se skládá ze čtyř částí:

- Vlastnosti (Properties)
- Dotaz (Query)
- Minimální počet hráčů (Min player count)
- Maximální počet hráčů (Max player count)

Vlastnosti (Properties) se skládají například z levelu hráče, regionu odkud se připojuje či typu vybrané hry. Dotaz (Query) určuje, jakým způsobem chce hráč hledat oponenty. Nakama interně používá vyhledávací a indexovací modul Bleve k vyhledání protivníků/spoluhráčů ve skupině (pool) matchmakeru.

V návaznosti na notifikace pak můžeme vidět využití socketu při registraci eventu o výsledku matchmaking procesu (vizte výpis kódu 1.1).

```
1 socket.onmatchmakermatched = (matched) => {
2   console.info("Received Matched message: ", matched);
3   console.info("Matched opponents: ", matched.users);
4 };
```

Výpis kódu 1.1: Ukázka použití event handleru.

Nakama je velmi kvalitní komerční řešení, které jde v placené verzi implementovat a použít s obdobným úsilím jako například EOS, které není závislé na žádné distribuční platformě a které se tím, že je škálovatelné určitě může jevit pro spoustu vývojářů jako dobrá volba. Je to však velmi nákladné řešení a jeho cena se může v průběhu času měnit. Je tedy možné, že pokud by došlo k zdražení a pokud by vývojáři nechtěli sahat do klíčové infrastruktury hry (kterou rozhodně online služby jsou), byli by prakticky nuceni server Nakama stále využívat, minimálně do doby než by mohli přejít na jiné řešení, což by se mohlo pohybovat v řádu několika měsíců. Nezapomeňme také, že stále je zapotřebí k tomuto serveru najít distribuční službu, která si bere provizi (jak už bylo uvedeno například Steam 30%). Velký rozdíl oproti EOS a Steamworks představuje matchmaking služba, která je u serveru Nakama robustnější a schopna pokrýt více případů a typů online her. Nezapomeňme, že Nakama server je také nutné nasadit, buď na vlastní servery nebo do cloudu, což přináší další finanční výdaje.

1.2.6 Photon Engine

Photon engine nabízí produkty, které můžeme dohromady označit jako online služby. Tyto produkty jsou však na rozdíl od serveru Nakama odděleny. Každá služba se může integrovat samostatně, ale je závislá na podkladové službě photon server/cloud, se kterou musí být integrována. Pojďme si tedy představit photon server/cloud a dále i ostatní služby.

Photon server a photon cloud

Photon server je on-premises real-time socket server a multiplatformní framework pro vývoj her pro více hráčů. Slouží jako podkladový server (cluster), na který se dají připojit jednotlivé služby. Bohužel photon server má limitace, které lze řešit pomocí photon cloud, což není on-premises řešení, ale SaaS, které má stejné funkcionality jako photon server, navíc však dovoluje integraci například se službou photon chat, kterou bychom při implementaci technologií photon mohli chtít využít, stejně jako u výše popsaných řešení (Steam, EOS, Nakama).

Jak photon server, tak photon cloud navíc podporují dedikované herní servery. Pokud by vývojáři chtěli mít servery, na kterých poběží jednotlivé herní instance, a nechtěli by, aby serverová část běžela u jednoho z hráčů, je toho pomocí photon serveru či cloudu možno docílit. Photon server/cloud obsahuje load balancer, který je schopen řídit přístupy na jednotlivé herní servery, kde běží herní instance a rozřazovat tak hráče do těchto herních instancí (to už však souvisí s photon realtime).

Photon chat

Tato služba umožňuje podobně jako chat v Nakama nebo EOS všechny funkcionality, které bychom od chatu čekali, tedy zprávy mezi dvěma uživateli, skupinové zprávy, ale i funkcionalitu navíc jako filtrování zpráv, a zahrnuje v sobě i zjednodušenou verzi friends služby, která umožňuje zobrazování online statusu uživatelů.

Photon realtime

Photon Realtime je služba (SaaS) Photon serverů (cloud), připravená na hraní her pro více hráčů. Pomocí rozhraní LoadBalancing API můžete přiřadit hráče ke sdílené herní relaci (zvané *room*) a synchronně přenášet zprávy v reálném čase mezi připojenými hráči napříč platformami.

Photon realtime tedy implementuje matchmaking pomocí herních relací, podobně jako Steam či EOS, hledání her probíhá na základě vlastností lobby (properties). Pod vlastnostmi lobby si můžeme představit například název zvolené mapy.

Produkty photon engine poskytují některé velmi zajímavé funkcionality, například podporu pro dedikované servery s load balancerem, avšak je také mnoho věcí, které nepodporují. Služba friends je velmi omezená. Poskytuje

pouze stav uživatele ale už ne například seznam přátel. Taktéž úplně chybí autentizace a je potřeba ji tedy vyřešit zvlášť. Tím, že jsou jednotlivé služby oddělené, sice získáme výhodu toho, že pokud by například selhala služba Chat, je možné, že dál bude fungovat Matchmaking, avšak také to znamená, že jednotlivé služby jsou placeny zvlášť.

Stejně jako u Nakama serveru je pro photon služby nutné buď používat on-premises server, za který budeme v cloudu vynakládat finanční prostředky, nebo mít vlastní serverovou infrastrukturu. Nabízí se samozřejmě i zmiňované photon cloud řešení, které je samo o sobě hostované, to se ovšem promítne na jeho ceně.

1.3 Zabezpečení online služeb

Především pro online hry více hráčů jsou podpůrné online služby kruciální. Pokud přestanou fungovat, nebo nefungují správně, může to ovlivnit zážitek hráče ze hry, či to může zcela zabránit ve hraní hry. Hráčům navíc v dnešní době nejde jen o hru samotnou, ale také o účet se hrou spojený, kde mohou mít odemčený nejrůznější herní obsah a mít ve hře jistý postup. To všechno jsou důvody, proč by hra samotná, ale i její služby, měly být správně zabezpečeny.

Bezpečnost je třeba řešit na všech úrovních, tedy již na úrovni serverů, kde jsou dané online služby nasazeny. Pokud by se útočníkovi podařilo dostat na server(y), jednalo by se o kritický bezpečnostní incident. Zároveň musí být ošetřena bezpečná komunikace mezi službami a také komunikace mezi službou a klientem. Dále pak musíme zajistit, aby služby mohly používat pouze autentizovaní a autorizovaní uživatelé. Při návrhu tak musíme brát v potaz všechny výše zmíněné a i další bezpečnostní mechanizmy a správně je implementovat.

1.3.1 Distribuční služba Battle.net a její prokázané zranitelnosti

Battle.net od společnosti Blizzard Entertainment, je zcela uzavřená distribuční služba, na kterou se dostávají pouze tituly z partnerských studií společnosti Activision Blizzard. Tato distribuční služba byla publikována již v roce 1996, pro původní hry Blizzard Entertainment jako například Diablo a StarCraft.

Krátce po vydání služby Battle.net byl skupinou lidí zrekonstruován síťový protokol pro komunikaci se službami, a tím bylo umožněno ke hraní her, které byly publikovány na Battle.net, používat jiné než oficiální servery, což mělo samozřejmě pro Blizzard finanční následky. Více v [11]. I později v roce 2012 se staly servery Battle.net obětí hackerů, kterým se podařilo dostat k emailům a odpovědím na bezpečnostní otázky po zadání zapomenutého hesla, více v [12].

Tedy i u společnosti Blizzard, která patří k jedné z největším v celosvětovém videoherním průmyslu, můžeme dohledat bezpečnostní incidenty, které mohou mít obrovský dopad jak na vývojáře, tak na hráčskou komunitu.

1.4 Porovnání prozkoumaných online služeb

V této sekci budeme porovnávat služby prozkoumané v sekci 1.2. Porovnání těchto služeb se pokusíme provést obecně, aniž bychom je vztahovali na potřebu hry Inpemo (konkrétní potřeby pro tuto hru budeme dále rozvádět v následující kapitole, kde také učiníme výběr vhodného řešení pro tuto hru).

Jedná se tedy o podpůrné online služby Steamworks, EOS, Nakama a Photon Engine. Některé rozdíly byly už v předchozí sekci 1.2 uvedeny, avšak často ne v kontextu všech čtyř možností. V tabulce 1.1 můžeme vidět porovnání dle vybraných parametrů. Jejich pořadí není uvedeno dle důležitosti. Výjimkou z obecných potřeb je poslední řádek tabulky 1.1, který řeší integraci s UE4, což se nepřímo vztahuje ke hře Inpemo, tato integrace však samozřejmě platí pro všechny hry vyvíjené v tomto herním enginu.

Je důležité uvést, že všechny porovnávané služby mohou mít i jiné využití než to, které je uvedené v tabulce 1.1. Některé řádky tabulky jsou také zjednodušeny, pokud uvedeme například řádek *UE4 integrace*, tak všechny služby uvedené v porovnání mají SDK v jazyce C++, tedy určitá forma integrace je již hotová, to ale bereme jako úplný základ, který pro hry v UE4 vyžadujeme. Steam má SDK přímo integrované do UE4 a je součástí Online Subsystému, což znamená, že můžeme jednodušeji používat funkce, máme již například nadefinované callback funkce pro notifikace. EOS sice není součástí online subsystému v UE4, existuje však plugin, který umožnuje jednoduché použití EOS v UE4 a to přímo ve vysokoúrovňovém vizuálním skriptovacím prostředí (blueprints). Nakama a Photon poskytují pouze SDK v C++ a žádná integrace do Online subsytému případně do skriptovacího prostředí neexistuje.

Závěrem ještě uveďme, že i když jsme přihlíželi k faktu zda je daná služba zdarma či nikoli, nemohli jsme jednoduše porovnat ceny placených služeb. Některé služby nabízí velké množství variant placených služeb (někde se platí vše najednou, někde po jednotlivých službách a podobně). Někde navíc ceny nejsou předem určené a je třeba je poptávat.

1.5 Shrnutí kapitoly

Ukázali jsme si, jaké možnosti pro implementaci online služeb do her exitují uvedli jsme si také pojem distribuční služba a zdůraznili důležitost online služeb v online hrách. Vybrali jsme čtyři rozdílná řešení, která je možné pro snazší implementaci a integraci online služeb do hry použít a ty jsme také v závěru kapitoly porovnali, dle kategorií, které jsou obsaženy v tabulce 1.1.

Připravili jsme si tím také podklady potřebné k samotné analýze hry Inpemo a jejích služeb. Především na základě probraných porovnání s přispěním dalších skutečností o hře se budeme schopni správně rozhodnout, které technologie pro online služby zvolit a z jakého důvodu.

	Steamworks	EOS	Nakama	Photon
Matchmaking	Ne (seznam místností)	Ne (seznam místností)	Ano	Ne (seznam místností)
In-game Chat	Ano	Ano	Ano	Pouze v cloudu
Hostováno	Ano (SaaS)	Ano (SaaS)	Ano	Pouze Windows
Self-Hostable	Ne	Ne	Windows, macOS, Linux	Pouze Windows
Autentizace	Steam	EOS nebo vlastní OIDC provider	Ano (build-in)	Ne
Transp. Protokoly	vlastní socket-based protokol	vlastní socket-based protokol	WSS, rUDP (secure), WebRTC	WSS, rUDP (secure), TLS, HTTPS
Open Source	Ne	Ne	Ano	Ne
Cena	Provize distributora	Zdarma	2 verze (placená a zdarma)	Placená (za službu)
Distributor	Ano	Ne	Ne	Ne
UE4 integrace	Ano (online subsystem)	Plugin	Ne	Ne

Tabulka 1.1: Tabulka porovnání online služeb.

Kapitola 2

Analýza a návrh online služeb pro hru Inpemo

V této kapitole si nejdříve uvedeme samotnou hru Inpemo, abychom porozumněli detailům, ke kterým budou online služby použity. Po porozumění návrhu hry pak provedeme analýzu potřeb online služeb a na základě těchto potřeb a také na základě porovnání technologií zejména ze sekce 1.4 rozhodneme o technologiích a postupech pro realizaci online služeb pro hru Inpemo.

Po uvedení jednotlivých požadavků na online služby pro hru Inpemo navrhneme jednotlivé služby a také komunikaci jednotlivých služeb, a to jak komunikaci s ostatními službami, tak komunikaci s hráči. Zvolíme technologie, které budeme používat a popíšeme nejen komunikaci samotnou, ale také komunikační protokoly mezi službami. Navrhneme také způsob integrace navržených online služeb do hry Inpemo. A zmíníme také, jakým způsobem budou online služby nasazeny.

2.1 Úvod do hry Inpemo

Jedná se o 3D low poly[5]hru vyvíjenou v UE4. Hra se dá hrát pouze online a je pro více hráčů. Spadá do žánru či typu her zvaných Brawler. Tento typ hry spočívá v tom, že proti sobě stojí několik postav v aréně (omezená oblast) a každá z těchto postav je ovládána jedním hráčem.

Každá postava má ve hře Inpemo několik kouzel, kterými postavy neudělují ostatním postavám poškození, jak je běžné v jiných podobných hrách, ale pouze je svými kouzly odstrkují. Pokud je jedna z postav odstrčena mimo arénu (omezenou oblast, například do vody), začne dostávat poškození a začnou jí ubývat životy. Pokud hráč nedostane svoji postavu zpět do bezpečné

[5]3D objekty ve hře se skládají z polygonů, termín *low poly* označuje, že daný 3D model nazývaný také jako polygonová síť (polygon mesh) obsahuje malé množství polygonů. Ve hře Inpemo modely obsahují řádově stovky polygonů. Pro srovnání, některé *high poly* hry mohou obsahovat modely, které obsahují statisíce až miliony polygonů.

zóny včas a životy mu klesou na nulu, pak postava umírá. Hra končí v moment, kdy ve hře zůstane poslední živá postava.

Postavy, které umřou, se mění v duchy a dále mohou interagovat s prostředím hry, ale ne přímo s ostatními živými postavami. Pokud je více než jeden duch ve hře, po pravidelném časovém intervalu se spouští mini hry, které duchům mezi sebou umožňují soupeřit o vrácení se do hlavní hry. Mini hry a jejich fungování zde nebudeme popisovat, neboť to není předmětem této práce a není to nutné k pochopení konceptu hry.

Kromě kouzel mohou také hráči používat předměty, které je možné zakoupit za měnu, kterou získávají ve hře pomocí různých herních událostí (opět nebudeme uvádět detaily) označovaných též jako eventy. Většina předmětů pak má podobné chování jako kouzla, vyjma pasivních předmětů, které například mohou danému hráči přidat životy. Tím, že se postavy mohou vracet zpět do hry pomocí mini her, by mělo být eliminováno předčasné opouštění herních instancí hráči, jejichž postava již umřela, protože stále mají naději zvítězit.

Herní instance probíhají rychle, jedna hra trvá ve většině případů méně než 10 minut. Aréna, ve které se hraje, se s postupujícím časem zmenšuje, a tak je pro hráče lehčí ostatní odstrčit a ubrat jim tím životy. Délka herní instance se odvíjí od aktivity hráčů, dále pak od módu. Mód hry určuje, v jakém rozložení (týmech) budou hráči proti sobě soupeřit. Hra Inpemo podporuje 4 módy s modifikacemi možností úpravy maximálního počtu hráčů. Zde je výpis jednotlivých módů:

Solo
: Obecně známý pod anglickým termínem Deathmatch, což bychom mohli přeložil jako zápas na život a na smrt. Každý hráč v tomto módu hraje sám. Maximální počet hráčů v tomto módu je 8.

Duo
: Týmy po dvou hráčích. Maximální počet hráčů v tomto módu je 8, tedy 4 týmy.

Trio
: Týmy po třech hráčích. Maximální počet hráčů v tomto módu je 9, tedy 3 týmy.

Quad
: Týmy po čtyřech hráčích. Maximální počet hráčů v tomto módu je 8, tedy 2 týmy.

2.2 Funkční požadavky online služeb pro hru Inpemo

Nejprve popíšeme funkční požadavky. Jako první věc po spuštění hry je vyžadována autentizace hráče. Je tedy třeba integrovat autentizační službu. Po

úspěšném přihlášení můžeme využít friends službu a pomocí ní například zjistit, jestli nějaký z našich přátel hru Inpemo právě hraje. Některé funkce služby friends však vyžadují fungování notifikací a tedy obecně integraci notifikační služby. Dále můžeme chtít s přítelem chatovat, tedy je třeba integrovat do hry chatovací službu. Vyhledání hry a přiřazení dalších hráčů je další fundamentální funkcionalita, kterou zajišťuje služba Matchmaking, kterou také musíme do hry integrovat.

V rámci realizace služeb se zaměříme na ty nejklíčovější služby, bez kterých by nemohl vzniknout funkční prototyp hry, a těmi jsou:

- Autentizační a autorizační služba
- Notifikační služba
- Služba matchmaking

Další výše zmíněné služby budou uvažovány v rámci návrhu služeb a jejich rozšiřitelnosti.

2.3 Nefunkční požadavky online služeb pro hru Inpemo

Začněme s nefunkčním požadavkem, který jsme již vlastně zmínili v sekci 1.3, a to Bezpečnost online služeb. Bezpečnost souvisí s funkčním požadavkem autentizace, ale obsahuje mnoho dalšího, ať už zabezpečení přístupu na server(y), kde dané služby budou nasazeny, nebo zajištění přenosu informací mezi službou a klientem. Integrace do hry vyžaduje, aby online služby byly veřejně přístupné. Počet hráčů, kteří hru budou hrát, může být a pravděpodobně bude velmi proměnlivý, je tedy nutné zajistit správnou škálovatelnost. Dalšími velmi důležitými nefunkčními požadavky jsou spolehlivost a dostupnost online služeb. Pokud služba/služby nebudou spolehlivé a dostupné, může to hráči zamezit ve hraní hry. To souvisí s dobou odezvy služby, která by měla být co možná nejkratší. Rozšiřitelnost služeb, tedy přidání další služby v budoucnosti, by mělo být co možná nejjednoduší.

V rámci návrhu a realizace online služeb budeme dbát na to, aby byly tyto nefunkční požadavky dodržovány. Pojďme je zde ještě přehledně vypsat:

- Bezpečnost
- Přístupnost
- Škálovatelnost
- Spolehlivost
- Dostupnost

- Odezva
- Rozšiřitelnost

2.4 Analýza z pohledu distribuce hry

V dnešní době je téměř nutností hru vydat pomocí nějaké distribuční služby, my jsme si zde uváděli Steam a Epic Store. Týmem vývojářů hry Inpemo[6], včetně autora této práce, bylo zvoleno, že hru Inpemo by mělo být možno distribuovat přes obě platformy (Steam, Epic Store). Spojením „by mělo být" zdůrazněme fakt, že Epic Store má blíže nespecifikované podmínky a může si vybírat, které hry bude publikovat v rámci své distribuční služby.

Pokud však počítáme s publikováním jak na Steam, tak na Epic Store, je jasné, že nemůžeme použít Steamworks, jelikož tato sada služeb je vázána čistě k platformě Steam. Museli bychom tedy implementovat zvlášť služby pro Steam a zvlášt pro Epic Store, což se z důvodů časové náročnosti nejeví jako efektivní řešení. Navíc bychom ztratili možnost meziplatformního hraní, tedy, pokud by jeden hráč přihlášený na Steam chtěl hrát s hráčem přihlášeným na Epic, nebylo by jejich společné hraní možné.

Mohli bychom tedy použít sadu služeb EOS, z první kapitoly však víme, že i ta má svá omezení, umožnuje nám sice meziplatformní hraní a není vázána na distribuční službu, má ale omezení například ve službě matchmaking, která umožňuje vyhledávání na úrovní lobbies. Mohli bychom tedy preferovat server Nakama, který má mnohem lépe nastavitelný matchmaking a vlastně až na horší integraci s UE4 poskytuje vše, co bychom potřebovali. Nakama server je však placený a tento fakt je velice důležitý. My preferujeme i za cenu větší náročnosti implementace neplacené řešení. Photon, který je stejně jako Nakama server placený, navíc vychází hůře ve srovnání z kapitoly číslo jedna.

Vidíme tedy, že žádné z uvedených řešení není ideální. Chtěli bychom takové řešení, které poskytuje co nejvíce nastavitelné služby, především matchmaking, kterému bychom mohli volit parametry dle našich představ, a zároveň řešení, které bude zadarmo a bude podporovat multiplatformní hraní. Na základě těchto skutečností a srovnání z první kapitoly bylo zvoleno, že se pro hru Inpemo bude implementovat vlastní řešení. Služby budou plně odpovídat potřebám hry Inpemo. Pro některé služby, jako například vytváření sessions, se stejně dobře hodí sada služeb EOS, která až na výjimky splňuje požadavky pro hru Inpemo, proto bylo rozhodnuto, že se tyto služby (EOS) propojí s vlastními službami, a to ulehčí implementaci některých služeb (jak jsme zmiňovali výše, některé služby nebudeme implementovat, ale použijeme ty z EOS) a umožní větší flexibilitu při vývoji služeb pro hru Inpemo.

[6]Tým programátorů rozhodující o zvoleném přístupu řešení online služeb pro hru Inpemo zahrnuje autora této práce, Bc. Michala Šveigra, Bc. Karím Abu Nofala a Bc. Tomáše Bohuslava.

2.5 Návrh architektury sady služeb pro hru Inpemo

Nejprve se pokusíme navrhnout jednotlivé služby, tedy zvolit technologie potřebné k jejich implementaci, a dále navrhnout, s kterými jinými službami či částmi systému budou komunikovat a jak na sobě budou závislé. Celou naši sadu služeb, kterou v této kapitole navrhneme, si pojmenujme *Bvision*. Ještě zmiňme, že některé služby a použité technologie jsou mezi sebou často provázané. Narazíme tedy na to, že se často budeme odkazovat na jiné sekce, kde budeme funkcionalitu nebo technologii popisovat podrobněji.

2.5.1 Autentizační služba

Pro zjednodušení názvosloví budeme nazývat tuto službu autentizační, ovšem jak uvidíme níže v této sekci, tato služba zajišťuje také autorizaci uživatelů. Toto bude po spuštění hry první služba, se kterou přijde uživatel do kontaktu. Jak jsme zmiňovali v sekci 2.4, chceme, aby naše služby byly použity jen tam, kde to bude nutné. Funkčnost, kterou můžeme, přenecháme sadě služeb EOS. Uživatelé nechtějí být nuceni vytvářet si nové účty, pokud to není nezbytně nutné. Pro jejich pohodlí bychom tedy chtěli, aby měli možnost používat buď Steam účet (v případě, že budou hru spouštět přes platformu Steam) nebo Epic účet (pokud budou hru spouštět přes Epic Store). Bohužel tyto účty nemůžeme používat ve hře přímo, nebyli bychom pak schopni zobrazovat přátele uživatelů z obou platforem najednou, ale pouze z té, kde by byl uživatel právě přihlášen.

Je tedy nutné mít vlastní službu, která bude autentizovat a autorizovat uživatele. Tato služba bude mít přístup ke všem uživatelům a bude tedy moci provádět většinu operací multiplatformně, včetně matchmakingu. V prototypu budeme uvažovat uživatele, které sami registrujeme, do budoucna se ale u této služby musí počítat i s autentizací pomocí Steam a Epic (jak bylo zmíněno výše, aby si uživatel nemusel zakládat nový účet).

Nezapomeňme, že stále chceme využívat EOS na některé další služby. EOS musí ovšem také nějakým způsobem ověřovat uživatele. Jediná možnost, jak toho docílit pomocí vlastní služby, je implementovat vlastní OIDC provider. EOS jinou možnost nepodporuje, více lze nalézt v [13]. OIDC je pouze nadstavba nad OAuth 2.0 serverem, která umožňuje kontrolu identity uživatele. EOS umožňuje integrovat tento OIDC provider pro přihlášení, více k tomu si opět ukážeme v následující kapitole.

Kvůli výše uvedeným skutečnostem je nutné zvolit pro autentizaci a autorizaci OAuth 2.0 server, který podporuje OIDC. Pro implementaci jsme zvolili javascriptový back-end runtime **Node.js** [14] a to především díky zkušenostem autora této práce s touto technologií a dobré podpoře knihoven a frameworků touto technologií, které dále používáme u ostatních služeb.

Pro implementaci je nejprve třeba rozhodnout, zda jít cestou implementace vlastního OAuth 2.0 serveru s OIDC či zkusit využít nějaký existující

server implementující OAuth 2.0 autorizační framework a OpenID Connect Core 1.0 dle standardů [15] a [16]. Cesta vlastní implementace by byla dle standardů velmi náročná a navíc vlastní implementace autorizačního serveru není náplní této práce. Rozhodli jsme se tedy použít node-oidc-provider [17], což je server, který implementuje výše uvedené standardy. Samozřejmě je to pouze server, který neobsahuje žádnou databázi, žádnou perzistentní správu přístupových tokenů a podobně. Bližší detaily vztahující se k implementaci uvedeme v následující kapitole.

Uvedli jsme si, že autorizační server potřebujeme z důvodu integrace s EOS, je však i jiný důvod, proč vůbec celou tuto službu navrhujeme, a ten se týká jednoho z nefunkčních požadavků, a to bezpečnosti. Potřebujeme mít možnost, jak zjistit, zda-li hráč, který se snaží přistoupit k dané službě, na to má patřičná oprávnění. Pokud bychom nevyžadovali u každé služby autentizaci a autorizaci, bylo by velice jednoduché například na matchmaking službu posílat falešné požadavky, čímž by mohlo dojít například k nefunkčnímu vyhledávání her v matchmakingu.

Každý požadavek na všechny služby (vyjma registrační a autentizační služby) musí nejdříve ověřit uživatele pomocí přístupového tokenu (access token). Opět, detaily ohledně vytváření, ověřování a obecně práce s refresh tokeny budeme řešit v následující kapitole.

2.5.2 Registrační služba

Jak jsme uváděli v předchozí sekci, tato služba by hráči měla být využívána minimálně, jelikož cílem je zjednodušit hráči přístup přes Steam a EOS a umožnit přihlašování právě přes tyto účty, budeme ale hráči dávat i možnost vytvořit si účet přímo u nás, pokud bude chtít.

Tato služba tedy bude umožňovat registraci hráče. Po registraci bude muset hráč projít ověřením účtu pomocí verifikačního emailu, v emailu bude odkaz na ověření účtu a až poté bude moci hráč pokračovat do hry. Tato služba by mohla být propojena přímo s autentizační službou, chtěli jsme však funkcionalitu co nejvíce oddělit a mít možnost tak třeba registrace pozastavit. Při případném výpadku jedné služby může zároveň druhá stále fungovat. Toto jsou tedy hlavní důvody, proč je tato služba samostatně. Její implementace bude také provedena v Node.js, jednotlivé použité frameworky a detaily implementace budou uvedeny v následující kapitole.

2.5.3 Databáze pro navžené online služby

V předchozí sekci 2.5.1 jsme si uvedli autorizační server, který samozřejmě bude muset mít svoji vlastní databázi, kam se budou ukládat například informace o registrovaných uživatelích, přístupové tokeny a podobně. V této práci budeme tuto databázi používat pouze v autentizační službě a v registrační službě, počítejme ovšem opět s rozšiřitelností a případným využitím

dalšími službami. Například služba statistik ukládající do databáze výsledky her, ukládání informací ve službe achievements nebo ve friends službě ukládání přátel registrovaných přímo přes Bvision služby. Toto rozšíření databáze je velmi pravděpodobné, a proto jsme se rozhodli navrhnout databázový cluster.

Existuje opět mnoho variant, jak takový cluster implementovat. Pojďme si uvést varianty ke zvážení a vyberme, kterou variantu použijeme.

Amazon RDS

Začneme s nejméně náročnou technologií na implementaci, tedy s Relational Database Service od Amazonu[7]. V této službě si můžeme na pár kliknutí vytvořit databázový cluster s jedním z mnoha DB enginů (Postgres, Aurora, MySQL a další). Jsme schopni jak horizontálně, tak vertikálně škálovat. Máme k dispozici monitoring, automatické zálohování a vytváření snapshotů DB. Co se týče bezpečnosti, tak zálohy, snapshoty i samotná DB mohou být snadno šifrovány a celý cluster může mít vlastní virtuální síť (VPC). Více v [18]. Samozřejmě, jelikož se jedná o službu Amazonu, není zadarmo. Při použití tohoto řešení tedy musíme počítat s dalšími měsíčními náklady na tuto službu. Ceny se odvíjí od mnoha faktorů, počet hodin měsíčně, kdy je cluster spuštěn, velikost DB v GB, počet I/O požadavků, velikost zálohy v GB a velikost přesunutých dat po síti. Jen za samotný cluster, který by měl ideálně běžet pořád, zaplatíme vyšší desítky USD měsíčně. Více o cenách RDS lze nalézt v [19] nebo v [20].

MongoDB

Databázový cluster založený na shardingu, což je architektura databáze, která rozděluje data podle rozsahů klíčů a distribuuje data mezi dvě nebo více instancí databáze. Sharding umožňuje horizontální škálování. Amazon RDS umožnoval použití různých SQL DB Enginů, Mongo DB je však NoSQL databáze a záznam v této databázi je dokument, který se dá přirovnat k JSON objektu. Dokument je datová struktura složená z dvojic polí a hodnot, kde hodnoty polí mohou zahrnovat další dokumenty, pole či pole dokumentů.

V rámci MongoDB máme na výběr on-premises nebo cloud řešení, s tím, že on-premises má placenou a neplacenou verzi. Cloudové řešení je s velkými omezeními také zdarma.

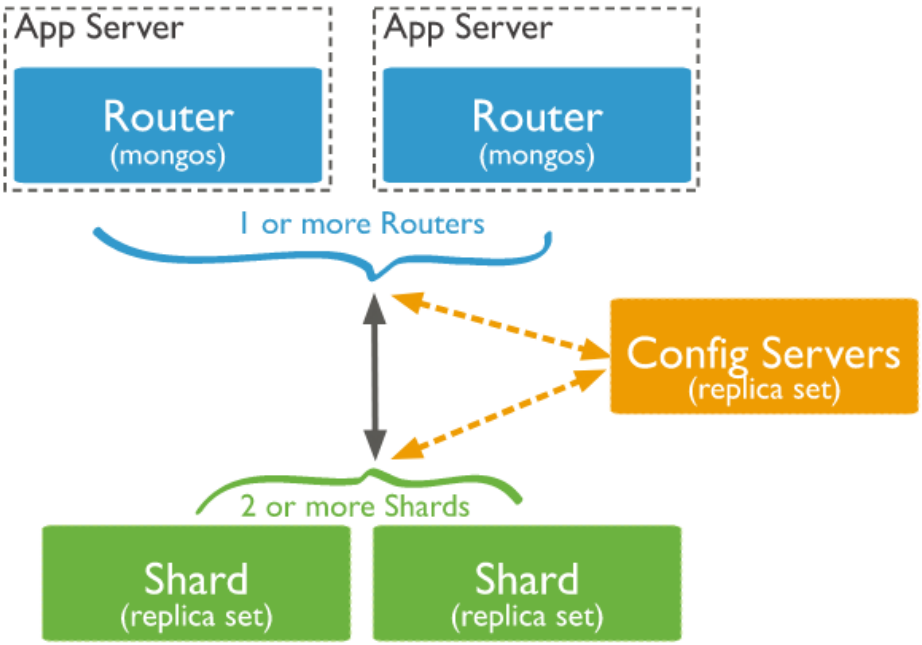
Pojďme si ještě podrobněji popsat, jak funguje shardování v této DB. Obsahuje 3 typy komponent:

- Shard - Obsahuje podmnožinu dělených dat. Je nasazen jako *replica set*, který implementuje replikaci a automatický failover při selhání.

[7]Stejně dobře bychom mohli využít například Google Cloud SQL nebo Azure SQL Database, zde se však pokoušíme uvést možné přístupy, jak DB cluster implementovat.

- Mongos - Funguje jako směrovač dotazů (query router) a poskytuje rozhraní mezi klientskými aplikacemi a shardovaným clusterem.
- Config server - Ukládá metadata a nastavení konfigurace pro cluster.

Výslednou architekturu pro dva shardy a dva směrovače můžete vidět na obrázku 2.1. Více informací pak můžete získat v [21].



Obrázek 2.1: MongoDB cluster architektura. Obrázek lze také najít v [21].

Z hlediska bezpečnosti MongoDB používá protokol TLS 1.1, bohužel v rámci neplacené verze neumožňuje takzvané *Encryption at rest*, tedy šifrování dat samotných.

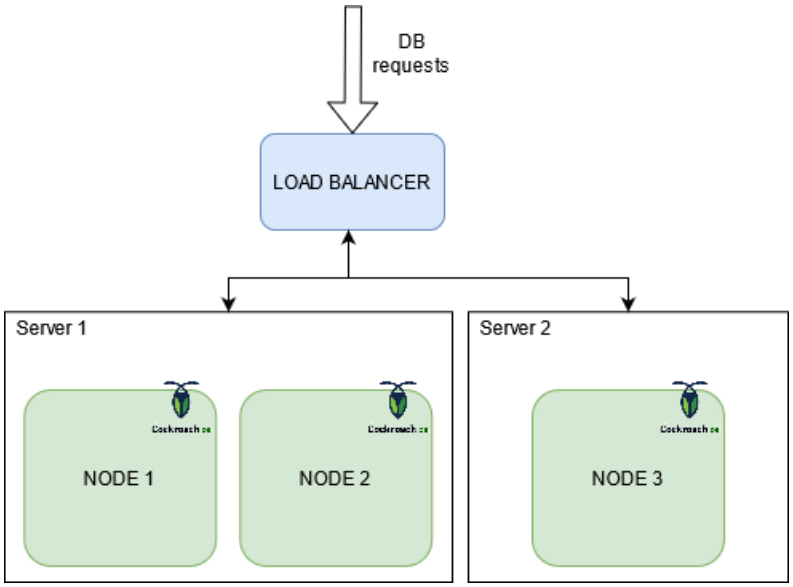
CockroachDB

Jedná se o distribuovanou SQL databázi postavenou na transakčním a silně konzistentním úložišti klíč-hodnota (key-value) a umožnuje horizontální škálování. Cockroach definuje pojem *Node*, což je samostatná instance se spuštěným CockroachDB. Jednotlivé nody se spojí a tvoří cluster. Tyto nody navíc můžeme za běhu do clusteru přidávat, či je z něj odebírat, CockroachDB navíc obsahuje monitoring pro celý cluster.

Podobně jako MongoDB, poskytuje CockroachDB také plně hostované řešení Cockroach Cloud, které je placené, dále pak CockroachDB Enter-

prise (On-premises řešení), které je takřka totožné s neplacenou verzí, jen navíc poskytuje podporu.

Pro nasazení CockroacheDB můžeme zvolit jednu z mnoha strategií. Nasazení na více strojů, kde budou jednotlivé nody rozprostřeny, či mít více nodů na stejném stroji. Nasazení pomocí Kubernetes či Docker Swarm je možné. CockroachDB má také velice dobrou podporu na všechny velké poskytovatele Cloud služeb (AWS, Digital Ocean, Google Cloud Platform). Na obrázku 2.2 můžeme vidět ukázku architektury pro více nodů na více strojích (více v [22]). Co se týče bezpečnosti, CockroachDB



Obrázek 2.2: CockroachDB cluster architektura.

používá bezpečnostní protokol TLS 1.2. Protokol TLS 1.2 používá asymetrické šifrování k vytvoření zabezpečeného kanálu a autentizaci komunikujících stran. Poté používá symetrické šifrování k ochraně průchodu dat (Encryption in flight). TLS 1.2 na rozdíl od TLS 1.1 (použitého v MongoDB) umožňuje používat bezpečnější hashovací algoritmy, jako je SHA-256. Umožňuje také využití pokročilých šifrovacích sad, které podporují kryptografii eliptických křivek (ECC). Více o rozdílech TLS verzí v [23].

Šifrování *Encryption at rest* je bohužel placená funkce (stejně jako šifrování záloh), která umožňuje šifrování všech souborů na disku pomocí AES v režimu counter mode. Bezpečnost je řešena velmi podobně jako u MongoDB. Více v [24].

Pojďme si ještě přehledně v tabulce 2.1 porovnat jednotlivá řešení a rozhodnout se pro technologii, která bude mít co možná nejnižší náklady a bude

odpovídat našim požadavkům.

	Amazon RDS	MongoDB	CockroachDB
SQL	Ano	Ne	Ano
horizontalní škálování	Ano	Ano[8]	Ano
load balancing	Storage, compute and latency	Storage	Storage, compute and latency
Distribuované transakce	Ano (MDTC)	Některé	Ano
Změna ve schéma DB	Některé změny za běhu	Pouze offline	Za běhu
Multi-region	Ano	Ne	Ano
Multi-cloud	Ne	Ne	Ano
Encryption at rest	Ano	Ano (placené)	Ano (placené)
Verze zdarma	Ne	Ano	Ano
On-premises	Ne	Ano	Ano
SaaS	Ano	Ano[9]	Ano[10]

Tabulka 2.1: Tabulka porovnání možných řešení databázových clusterů.

Na základě výše uvedených vlastností jednotlivých řešení a jejich porovnání vyberme CockroachDB. Můžeme použít verzi zdarma, kterou musíme samozřejmě také někam nasadit. V rámci implementace služeb budeme také řešit jejich nasazení. Služby tedy můžou běžet na stejných serverech jako databázový cluster. Více o serverech a nasazení jednotlivých služeb si povíme dále v této kapitole. Jediná velká nevýhoda, která vychází z porovnání, je, že CockroachDB nám ve vybrané variantě neposkytne Encryption at rest. Nepotřebujeme nutně zvolit NoSQL DB a v ostatních ohledech oproti MongoDB vychází CockroachDB obecně lépe, vizte tabulku 2.6. Rozhodování bylo také ovlivněno zkušeností autora této práce se serverem Nakama, kde je používán právě databázový cluster CockroachDB, a tedy jeho předchozí znalostí, zároveň znalostí PostgresSQL, se kterým uživatel databáze zachází velmi obdobně jako s CockroachDB, jak si ukážeme později. Například při implementaci registrační služby.

Jediné dvě služby (autentizační a registrační), které budeme implementovat a zároveň budou používat databázi, jsme si již v této kapitole popsali, můžeme pro ně tak i navrhnout databázi. Schéma bude velice jednoduché, potřebujeme pouze správu účtů s tím, že všechny účty budou mít stejná oprávnění (všichni hráči ve hře budou mít stejné možnosti). Pro registraci

[8] Omezený zápis.

[9] Zdarma pouze do velikosti DB do 5GB, jinak se jedná o placenou verzi v cloudu.

[10] Jedná se o placenou verzi v cloudu.

bude zapotřebí ukládat registrační tokeny. Access tokeny budou ukládány do paměti, při případném restartu služby bude tedy zapotřebí vygenerovat tokeny znovu, to znamená znovu se přihlásit (do budoucna se počítá s ukládáním access a refresh tokenů do perzistentního úložiště).

2.5.4 Notifikační služba

Nyní navrhneme službu, která slouží jako středobod celé sady služeb Bvision. Všechny asynchronní požadavky musí projít přes ni a všichni hráči k ní musí být připojeni. Jelikož bude služba takovýmto způsobem vytížena, je s tím zapotřebí při návrhu počítat.

Základní funkcionalitou notifikační služby je, že bude posílat notifikace (data) hráči. Služba musí vědět komu data poslat (jakému hráči) a musí odněkud data získávat (z jiné služby).

Začněme tedy spojením mezi hráčem a notifikační službou. Jako první varianta nás může napadnout, že notifikační služba může hráčům zasílat požadavky s daty, narazíme ale na problém, jak daného hráče najde, co se má stát, když takový hráč ani neexistuje, nebo není k notifikační službě připojen. Bylo by tedy dobré, aby dané spojení bylo perzistentní. Pokud hráč spustí hru a přihlásí se jako první, pošle požadavek na notifikační službu, že se k ní chce připojit. Toto spojení lze vytvořit různými způsoby. Základní způsob, jak toho docílit, by bylo používat techniku zvanou *Short polling*. Ta funguje tak, že hráč[11]v pravidelných intervalech bude posílat požadavky na notifikační službu, která buď vrátí, že žádná nová data od doby, kdy se hráč ptal, nepřišla, a nebo případně nová data vrátí. Nevýhodou je, že nevíme, kdy v daném intervalu zpráva opravdu přišla, známe jen rozmezí. Technika *Long polling* také posílá požadavky, kdy se ptá, zda přišla nová data. Pokud nová data přišla, vrátí je, v opačném případě drží spojení s hráčem, dokud je to možné, poté, pokud data stále nepřišla, odpoví pomocí timeout a požadavek je třeba poslat znovu. Další variantou je websocket, který umožnuje obousměrnou interaktivní komunikační relaci, tedy skutečné perzistentní spojení. Tuto metodu, tedy websocket, použijeme. Poskytne nám výhodu v tom, že budeme dostávat data ve chvíli, kdy jsou zpracována notifikační službou. Zároveň máme perzistentní spojení, stačí tedy udělat pouze jeden TCP trojcestný handshake na rozdíl od variant pollingu, kdy se handshake provádí s každým požadavkem. Websocket také používáme z toho důvodu, že notifikace různých typů budou velmi časté, ať už se jedná o zprávy z chatu, notifikace o zařazení do matchmakingu, či odemknutí achievmentu. To je jen zlomek notifikací, které bude hráč moci přijímat. Stejně jako u autentizační služby použijeme technologii Node.js. Více informací k samotné implementaci websocketu uvedeme v realizační části práce.

[11]Toto se děje na pozadí v rámci instance hry, ve skutečnosti požadavky posílá instance hry (klient), který má unikátní ID a reprezentuje daného hráče, jelikož je přihlášený a má přiřazený platný přístupový token (access token).

Data se do notifikační služby budou propagovat z ostatních služeb. Uveďme příklad, kdy služba matchmaking chce kontaktovat hráče s tím, že byl přiřazen do hry. Jedná se o asynchronní požadavek a víme, že matchmaking služba nekontaktuje hráče přímo, ale přes notifikační službu, která je implementována právě kvůli takovým požadavkům. Máme opět možnosti jako výše (tedy short/long polling nebo websocket) a jelikož notifikace z jednotlivých služeb mohou být zasílány často, budeme tedy preferovat perzistentní spojení. Pokud by byla vytíženost služeb malá, tedy měli bychom málo hráčů, vše by fungovalo, matchmaking pošle data přes websocket do notifikační služby a ta data zpracuje a zjistí, jakému hráči je má poslat. Pokud budeme zátěž zvyšovat, může se stát, že notifikační služba přestane stíhat vyřizovat všechny požadavky. Abychom tomuto zamezili, můžeme zvýšit kapacity notifikační služby (škálování). Také můžeme navrhnout komunikační vrstvu mezi jednotlivými službami, které kontaktují notifikační službu. Místo přímého posílání zpráv mezi například matchmaking službou a notifikační službou zvolíme publish-subscribe model, který implementujeme pomocí messaging queue. Detaily o messaging queue si probere zvlášť v následující sekci 2.5.5, kde si řekneme, jaké má pro nás messaging queue v této situaci výhody.

Víme tedy, jak data v notifikační službě přeposílat, ale zaměřme se ještě na rozšiřitelnost této služby. Jestliže používáme pro připojení k hráčům websockety, musí notifikační služba tyto sockety spravovat a být schopna v nich vyhledávat, aby měla možnost poslat notifikaci správnému hráči. Toho můžeme docílit uložením aktuálních připojení. Ta však mohou být velmi nestabilní a velmi proměnlivá, zároveň jich může být velké množství. Mohli bychom je ukládat společně s ostatními daty do databázového clusteru CockroachDB. Je však možné sockety držet přímo v paměti serveru. To má z pohledu konzistentních transakčních Databází obrovské nevýhody, ale pro tato konkrétní data (informace o socketech), která jsou jen dočasná (jen na dobu, kdy je hráč připojen do hry), to má výhodu v rychlosti přístupu k datům. I když si představíme nejhorší možný scénář, že přijdeme o všechny sockety (to by mohlo nastat při pádu notifikační služby), i přesto jsme schopni jednotlivá spojení jednoduše obnovit, a pokud jsou služby implementovány správně, dojde jen k drobné časové prodlevě při zasílání notifikací. Až nyní se tedy dostáváme zpět k rozšiřitelnosti, kdy, pokud notifikační službu implementujeme jako cluster, musíme mít na paměti, že všechna spojení jsou navázána s některým z nodů[12]v clusteru, a tedy je třeba, aby notifikační cluster přeposílal data na správný node, případně mít sdílenou paměť pro všechny nody v clusteru a pak posílat data na libovolný node (typicky méně vytížený).

Pro sdílenou paměť pak používáme technologii Redis, což je úložiště datové struktury v paměti, používané jako databáze nebo cache. Pokud tedy jeden Node bude chtít poslat notifikaci nějakému hráči, najde potřebná data

[12]Nodem v clusteru je myšlena jedna z běžících instancí služby. V tomto konkrétním případě si můžeme jako node představit instanci Node.js serveru.

ve sdílené paměti (mezi všemi Nody) implementované pomocí Redis a poté notifikaci hráči zašle.

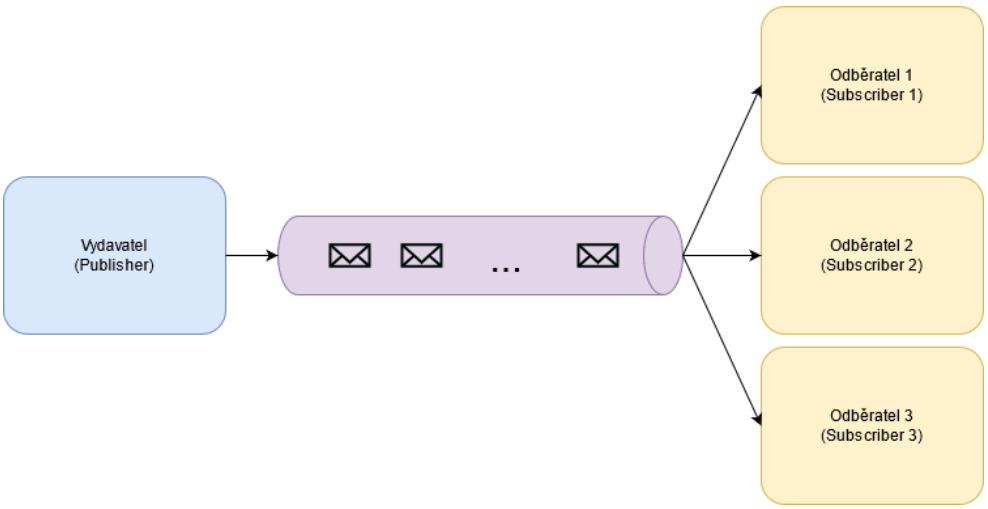
Samotná notifikační služba bude jen jednoduše zpracovávat zprávy, které do ní přijdou, respektive, které si vezme z messaging queue a bude je správně směrovat k hráčům. Stejně jako u autentizační služby budeme tuto službu implemenontovat pomocí technologie Node.js. Jednotlivé detaily k implementaci si pak ukážeme v následující kapitole.

Zmiňme ještě poslední detail, který díky této službě můžeme a budeme implementovat a tou je kontrola odezvy. Každý hráč, pokud je přes socket ke službě připojen, má nějakou odezvu na tuto službu. Jelikož jde o perzistentní spojení, můžeme tuto odezvu sledovat a pravidelně aktualizovat. K čemu odezvu potřebujeme vysvětlíme v sekci 2.5.7, která se zabývá matchmakingem.

2.5.5 Messaging queue pro navržené online služby

Messaging queue jsme již zmínili v předchozí sekci 2.5.4. Tato technologie se dá implementovat mnoha různými způsoby a také se dá různě používat, pro náš případ chceme využít model publish/subscribe, který si nejprve vysvětlíme a poté rozhodneme o technologii, kterou pro implementaci použijeme.

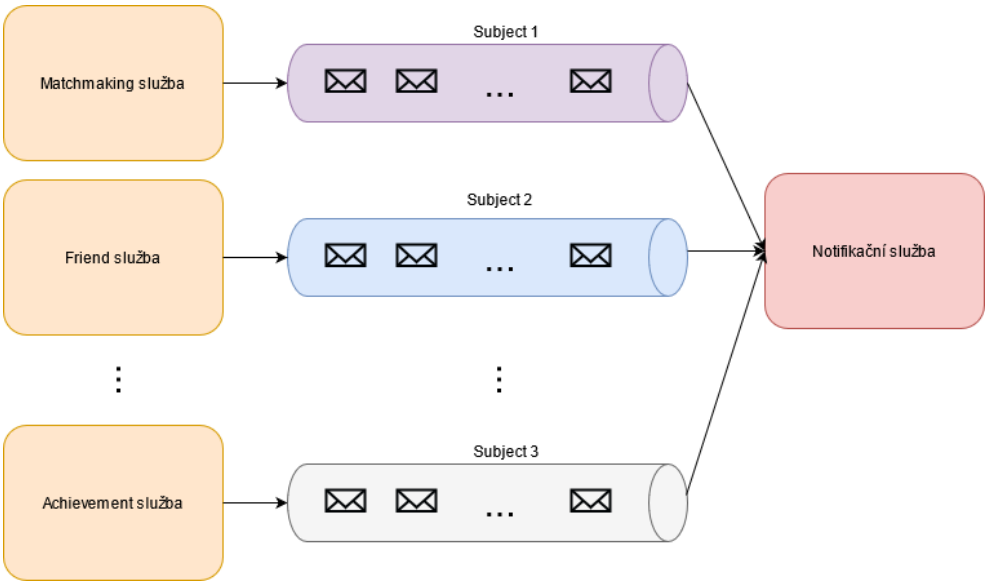
Vzor publish/subscribe funguje tak, že služby posílají (publikují) zprávy, tyto služby nazýváme vydavatelé (publishers) a ty, které zprávy dostavají, nazýváme odběratelé (subscribers). Služby, které se zajímají o zprávy vydavatelů, se přihlásí k odběru (subsribe) předdefinovaného kanálu (subject)[13], o kterém vědí, že do něj vydavatelé budou zprávy posílat. Více o jednotlivých vzorech lze nalézt v [25].



Obrázek 2.3: Vzor publish/subscribe.

[13] V některých implementacích můžeme tento termín najít také pod názvem topic.

Tento vzor obecně počítá s tím, že můžeme mít více odběratelů, což můžeme vidět na obrázku 2.3, kde je standardní ukázka návrhu tohoto vzoru. V našem případě je odběratel pouze jeden (notifikační služba), do budoucna se ale může stát, že bychom potřebovali například ještě dalšího odběratele, proto volíme tento volnější vzor místo vzoru many-to-one, který povoluje pouze jednoho odběratele. Náš návrh najdeme na obrázku 2.4. Můžeme na něm také vidět, že zprávy řadíme do jednotlivých kanálů dle služeb. Notifikační služba odebírá všechny kanály. Samozřejmě bychom mohli mít pouze jeden kanál, ale pro přehlednost a také kvůli bezpečnosti mějme jednotlivé služby oddělené.



Obrázek 2.4: Bvision návrh publish/subscribe.

Nyní se dostáváme k technologiím, které můžeme použít pro implementaci messaging queue. Je jich celá řada. Popíšeme si tři, mezi kterými si zvolíme jednu pro výslednou implementaci na základě našich potřeb.

NATS

Jedná se o middleware, který je schopen posílat až miliony zpráv za vteřinu, implementuje vzor Publish/Subscribe, který jsme navrhli a dá se velmi jednoduše škálovat. Jedná se o velmi jednoduše integrovatelný middleware, který nabízí mnoho způsobů zabezpečení. Auntetizace může být vyžadována pomocí přístupových tokenů, Nkey, či přístupových údajů (jména a hesla) a zabezpečení komunikace probíhá pomocí protokolu TLS. Zmiňme ještě, že NATS používá ke komunikaci vlastní stejnojmenný text-based protokol.

Použití je velmi jednoduché, klientskou část můžeme opět implementovat pomocí technologie Node.js přímo v dané službě. Pro instalaci

serveru nepotřebujeme žádné další předpoklady. NATS lze velmi jednoduše škálovat a zatížení systémových prostředků NATS serverem je velmi malé. NATS neumožňuje persistentní zprávy, takže pokud nějaký node z nějakého důvodu vypadne, zprávy na něm uložené jsou ztraceny. Tato nevýhoda podporuje tvrzení, že NATS je jednoduché a rychlé řešení. Více v [26].

RabbitMQ

Je komplexnější messaging systém, který podporuje více funkcionalit než NATS. Je například schopen zaručit doručení zprávy pomocí vzoru Store and Forward, který umožnuje ukládat zprávy na disk či do paměti. RabbitMQ může být také clusterovaný a podporuje také publish/subscribe vzor, který však implementuje jinak. Vydavatelé neposílají zprávy přímo do kanálu či fronty, ale do prostředníka s názvem exchange, který zajišťuje, co se má se zprávou stát. Existují různé exchange typy, které mohou poslat zprávu do jedné nebo více front. RabbitMQ má také javascriptového klienta, kterého můžeme použít v Node.js (více v [27]).

Apache Kafka

Je nejkomplexnější systém ze všech tří porovnávaných, podporuje replikace zpráv, jejich perzistenci a také paralelní čtení z jednotlivých kanálů (u Kafka označované jako topics). Tento systém, jak vyplývá z předchozích informací, je škálovatelný. Dokonce je doporučené ho pouštět pouze jako cluster. Umožňuje také monitoring celého clusteru a kompakci[14]logů v rámci interní databáze. Kafka za to, že má mnoho vlastností navíc, ale zároveň trochu platí výkonem: ukládat zprávy, zajišťovat replikaci a podobně je časově náročnější, než zprávu prostě poslat, proto tedy propustnost zpráv za vteřinu bude mnohem menší než u NATS serveru. To se dá do jisté míry eliminovat zvětšením clusteru, což však stojí HW prostředky. Oproti RabbitMQ jde Kafka trochu dál, například zmíněnou kompakci logů RabbitMQ neposkytuje, avšak nasazení celého clusteru vyžaduje zvýšené usílí a také je potřeba počítat se závislostí na Javě a technologii Zookeeper, která zajištuje synchronizaci v tomto distribuovaném systému. Klientská část v Node.js stejně jako u předchozích řešení existuje. Více o Apache Kafka lze nalézt v [28] nebo v [29].

Pro potřeby hry Inpemo a jejích služeb nám stačí funkcionality poskytované systémem NATS, přihlédněme i k jednoduchosti nasazení serveru a používání klientské části. Pokud bychom však vyžadovali komplexnější řešení, které podporuje i perzistenci zpráv, Apache Kafka stejně jako RabbitMQ poskytují škálovatelná řešení a v obou případech zdarma. Více detailů k implementaci NATS si ukážeme v následující kapitole.

[14]Kompakce logů zachovává pouze poslední verzi klíče a jeho hodnotu. Kompakce logů tedy slouží k obnovení stavu po selhání systému pro in-memory služby pomocí *compaction topic logu*.

2.5.6 Komunikační protokol pro zasílání notifikací

V předchozích sekcích jsme navrhli, jak bude fungovat notifikační služba a jakým způsobem bude komunikovat s hráči a ostatními službami. Zprávy, které budeme posílat, by však měly být strukturované, měly by mít svá pravidla a to z mnoha důvodů. Pokud například nastane chyba, chceme, aby byla jednoznačně identifikovatelná. V rámci implementace hry Inpemo můžeme počítat s daným formátem zpráv, se kterým si služby budou umět poradit. Pojďme nyní navrhnout vlastní text-based protokol. Pro správnou integraci do hry Inpemo byl návrh tohoto protokolu prováděn ve spolupráci autora této práce a Bc. Michala Šveigera provádějícího implementaci samotné hry Inpemo.

Navrhneme zprávy dvěmi směry, zprávu z notifikačního serveru ke hráči nazveme *Event* a zprávu opačným směrem (od hráče na notifikační server) nazveme *Message*. Pojďme si nyní definovat, co bude zpráva obsahovat a proč. Protokol je text-based a zprávy jsou ve formátu JSON.

Event - notifikační služba → hráč

Tyto zprávy budou nejčastěji notifikace předávající data o asynchronním požadavku z jiné služby. Ve výpisu kódu 2.1 můžeme vidět příklad zprávy typu Event.

```
{
    service_url: 'https://matchmaking.bgrgames.com',
    service_code: 11,
    timestamp: 1444677045952,
    event_type: 'match_found',
    event_code: 201,
    payload: {
        ....
    }
}
```

Výpis kódu 2.1: Příklad zprávy typu Event z protokolu Bvision (JSON).

První dva atributy jsou `service_url` a `service_code` udávající, ze které služby původně požadavek pochází. Atribut `service_url` vyjadřuje stejnou informaci, ale v lidsky čitelné podobě, která obsahuje URL k dané službě.

Zde je seznam služeb, který se samozřejmě s implementací dalších služeb může rozšiřovat:

10 : NOTIFICATION_SERVICE

11 : MATCHMAKING_SERVICE

12 : FRIENDS_SERVICE

13 : CHAT_SERVICE

Další atribut je `timestamp`, který obsahuje unix timestamp o 13 číslicích. Dále pak dvojice atributů `event_type` a `event_code`, event kódy jsou

identifikátory výsledku operace v dané službě. Služby obecně mohou zajišťovat mnoho operací, a tak je nutné počítat s větším rozsahem kódů, zvolme tedy kód o třech číslicích. Stejnou informaci nese `event_type` v lidsky čitelné podobě. Jako poslední zbývá atribut `payload`, což je objekt, který může obsahovat libovolné podatributy, které jsou specifické pro každý event. Pro event `match_found` mohou například obsahovat pole všech hráčů, kteří byly do hry nalezeni, a kdo z nich hru hostuje. Pro příklad atributu `payload` vizte výpis kódu 2.2

```
payload: {
    players:[
        {
            nickname: "pahicz",
            host: false,
        },
        {
            nickname: "usak",
            host: true,
        }, ...
    ]
}
```

Výpis kódu 2.2: Příklad atributu payload ze zprávy typu Event (JSON).

Zde je výčet navržených event kódů/typů pro službu matchmaking:

201 : match_found

202 : match_canceled

203 : match_confirm_dialog_update

401 : queue_kicked_latency

402 : queue_kicked_timeout

Zde pro notifikační službu:

100 : welcome

101 : pong

Pro služby, které nebudou v rámci této práce implementovány, nebyly event kódy navrženy.

Message - hráč → notifikační služba
Toto jsou jen zprávy speciálního charakteru od hráče směrem na notifikační službu. Tyto typy zpráv budeme označovat jako Message, jsou stejně jako zprávy typu Event ve formátu JSON. Ve výpisu kódu 2.3 můžeme vidět příklad této zprávy typu Message.

Atributy `msg_type` a `msg_code` jsou podobné atributům `event_type` a `event_code`, udávají požadavek hráče na notifikační server. Dále pak

atributy `timestamp` a `payload`, které jsou stejné jako u zpráv typu Event. Zmiňme, že protokol v sobě neobsahuje informaci o tom, kdo danou zprávu odeslal. Zpráva je posílána po socketu a právě na této úrovni je tato informace řešena. Pomocí socketu musí být zajištěno, že notifikační služba ví, od koho daná zpráva přišla a potažmo i kam kterou zprávu odeslal (což je vysvětleno v přechozí sekci 2.5.4).

```
{
    msg_type: 'ping',
    msg_code: 101,
    timestamp: 1444677045952,
    payload: {
        ....
    }
}
```

Výpis kódu 2.3: Příklad zprávy typu Message z protokolu Bvision (JSON).

V této práci implementujeme dva Message typy:

101 : ping

102 : refresh_token

Oba Message typy ještě budeme podrobněji probírat u služby matchmaking a u implementace autentizační služby.

Nyní máme protokol navržený a můžeme pomocí něj posílat a také přijímat zprávy v notifikační službě. V implementační části pak rozvedeme implementaci protokolu jak v notifikační službě, tak přímo ve hře Inpemo.

2.5.7 Služba Matchmaking

Tuto službu jsme již v práci několikrát zmiňovali, v sekci 2.1 jsme představili jednotlivé módy, které má hra Inpemo podporovat a tedy, které musí také podporovat matchmaking služba. Pojďme nyní představit, jak z hlediska síťové komunikace budou fungovat jednotlivé instance hry a k čemu jsme v dřívějších sekcích zmiňovali odezvu (či latenci).

V online hrách pro více hráčů existují dva základní přístupy, jak může být hra hostována:

Dedikované servery

Robustnější řešení, které počítá s autoritativním serverem, který řídí hru. Na tomto serveru se vytvoří instance hry, ke které jsou hráči připojeni. Všechny změny jsou posílány na tento server a ten je potvrzuje. Pokud jsou například nějaké postavě ve hře ubrány životy, server musí tuto operaci ověřit a rozposlat jednotlivým hráčům (tento proces se nazývá replikace a nebudeme ho zde podrobněji rozvádět, neboť se jedná o komplexní problematiku v rámci UE4 a zároveň to není obsahem této

práce). Server tak má kontrolu nad jednotlivými akcemi hráčů. Každý z hráčů může hru kdykoliv opustit a pro hru to nebude mít z hlediska síťové komunikace žádné konsekvence, pouze dojde k odpojení klienta od serveru. Dedikované servery musí být poskytovány vývojáři dané hry, pro každou instanci hry se tak musí spustit serverová část hry UE4 na daném serveru, což je konzolová aplikace, která však vyžaduje nemalý výpočetní výkon právě kvůli replikacím. Z hlediska CPU je tedy mnohem vytíženější než jednotlivé online služby.

Pokud bychom počítali s větším počtem hráčů, rostou s tím také náklady na dedikované servery. Herní instance by měly být (pro kvalitní herní zážitek) co možná nejstabilnější a měly by mít co nejmenší odezvu. Pokud bychom pro tyto servery chtěli mít vlastní hardware, vyžadovalo by to obrovské množství práce na správě infrastruktury, abychom mimo jiné zajistili vysokou dostupnost serverů. Jako další možnost se jeví herní instance zakládat v cloudu, Amazon má pro tyto účely službu Gamelift, což umožňuje velmi flexibilní poskytování serverů, na kterých jednotlivé instance mohou běžet. Amazon v této službě nabízí dva typy ceníků, *spot pricing* a *on-demand pricing*, první z uvedených je sice levnější, ale je zde potřeba počítat s tím, že Amazon, pokud potřebuje prostředky uplatnit v rámci svých jiných služeb, může dané instance vypnout. Je tedy možné, že virtuální server na kterém může běžet několik instancí hry, bude během krátké doby vypnut a hry tak budou ukončeny předčasně (10 minut před vypnutím serveru přichází varování). *On-demand pricing* se tedy pro hru Inpemo jeví jako lepší varianta, která garantuje, že herní instance nebude předčasně vypnuta.

Tato služba poskytuje velké spektrum virtuálních strojů, kde je možné si vybírat velikost paměti, počet virtuálních CPU a podobně. Vše se odvíjí od toho, kolik prostředků reálně spotřebujeme. Pokud vezmeme hrubý příklad, kdy vybereme dvě instance `c5.2xlarge`, které mají 8 vCPU a 16 GiB paměti, odhadněme, že na jedné této instanci může běžet 6 instancí herního serveru[15], jsme tedy schopni spustit 12 herních instací. Hráčů v jedné instanci může být 2-9, opět hrubým odhadem můžeme počítat 6 hráčů na instanci. Celkově jsme schopni obsloužit circa 72 hráčů. Pokud by taková zátěž byla každou hodinu v měsíci, pouhých 72 hráčů by stálo přes 150 USD měsíčně.

Toto řešení má spoustu výhod a pro hráče zaručuje nízkou latenci a vysokou spolehlivost herních instací, je však velice nákladné.

Hostování u klienta

Jedná se o variantu, kdy je vybrán jeden z hráčů, který bude zastávat také roli serveru. Oproti dedikovanému serveru je hlavní nevýhoda v tom,

[15]Toto je opravdu jen hrubý odhad, který se odvíjí od implementace hry, více o této problematice lze nalézt v [30].

že pokud hráč, který danou hru hostuje, z nějakého důvodu hru opustí, nastává problém. Základním řešením je, že herní instance by tak skončila, hra Inpemo však implementuje rehosting, což je technika, která je schopna rozpoznat, že hostující hráč hru opustil a roli serveru přiřadí jinému hráči[16]. Další nevýhodou je, že pokud hráč, který hru hostuje, má vysokou odezvu nebo větší ztrátovost paketů, herní zážitek se tak zhorší všem hráčům, neboť všichni budou mít zvýšenou odezvu na server, který je spuštěn u daného hráče. Výhodou je, že toto řešení nevyžaduje žádné náklady. U her, které mají menší rozpočet na vývoj, je toto řešení často využívané.

Z finančního hlediska jsme nuceni využít řešení hostování u klienta. Z toho již plyne, že je velmi důležité, jak zvolíme, kdo bude hru hostovat. Zde nám pomůže kontrola odezvy, kterou nám umožňuje získat notifikační služba. Toto úzce souvisí se samotnou implementací ve hře Inpemo, kdy hráč posílá ping zprávu na notifikační službu, ta ji přijme a odpoví zprávou pong, hráč jako odezvu bere čas mezi posláním zprávy ping a obdrženým zprávy pong. U hráče, který hru hostuje, je navíc nutné počítat s tím, že kromě instance hry u něj bude navíc spuštěná instance serveru, která bude vyžadovat další systémové prostředky. Zatím však z nároků na hru Inpemo usuzujeme, že tato zátěž navíc by měla být zvládnutelná pro všechny hráče, a nebude tak třeba v rámci matchmakingu kontrolovat, zda má hráč na hostování odpovídající hardware.

Ve hře na klientovi je pevně daný počet dvaceti záznamů, které se cyklí, každé dvě vteřiny se promaže nejstarší záznam a uloží se nová hodnota odezvy, zprůměrováním všech hodnot získáme průměrnou odezvu za posledních 40 vteřin. Ve hře bude také možnost v nastavení povolit či zakázat hostování her, pokud si je například hráč vědom, že má horší internetové připojení, bude dobré tuto možnost využít a hostování tím zakázat.

Nyní si můžeme ukázat příklad požadavku na matchmaking server. Vizte výpis kódu 2.4, který obsahuje POST požadavek pro nalezení hry. Můžeme vidět, že data jsou stejně jako u Bvision protokolu v JSON formátu. V datech zasíláme `accessToken`, který je nutný pro ověření požadavku, dále pak `mode`, který nabývá hodnot (solo, duo, trio, quad)[17]. Pole `players` obsahuje jednotlivé hráče, kteří jsou součástí požadavku pro vyhledávání.

Ve výpisu kódu 2.4, jelikož jde o mód solo, je pouze jeden hráč. Pokud bychom zvolili mód quad, mohli bychom posílat požadavek až se čtyřmi hráči. Jinými slovy tento endpoint počítá se skupinovými požadavky, pokud hráč chce hrát mód, kde bude hrát se třemi dalšími spoluhráči (tedy quad), může buď hru hledat sám, a nebo ve skupině dvou až čtyř lidí. Všechny tyto možnosti musí matchmaking podporovat. Požadavek však vždy posílá pouze jeden hráč, a to vedoucí skupiny, nebo jednotlivec bez skupiny. Noti-

[16] Rehostování může mít za důsledek pozastavení hry na dobu několika vteřin.
[17] Módy jsme již řešili v sekci 2.1.

fikační služba však posílá notifikaci o nalezení hry všem, kteří jsou v daném požadavku (všem členům skupiny).

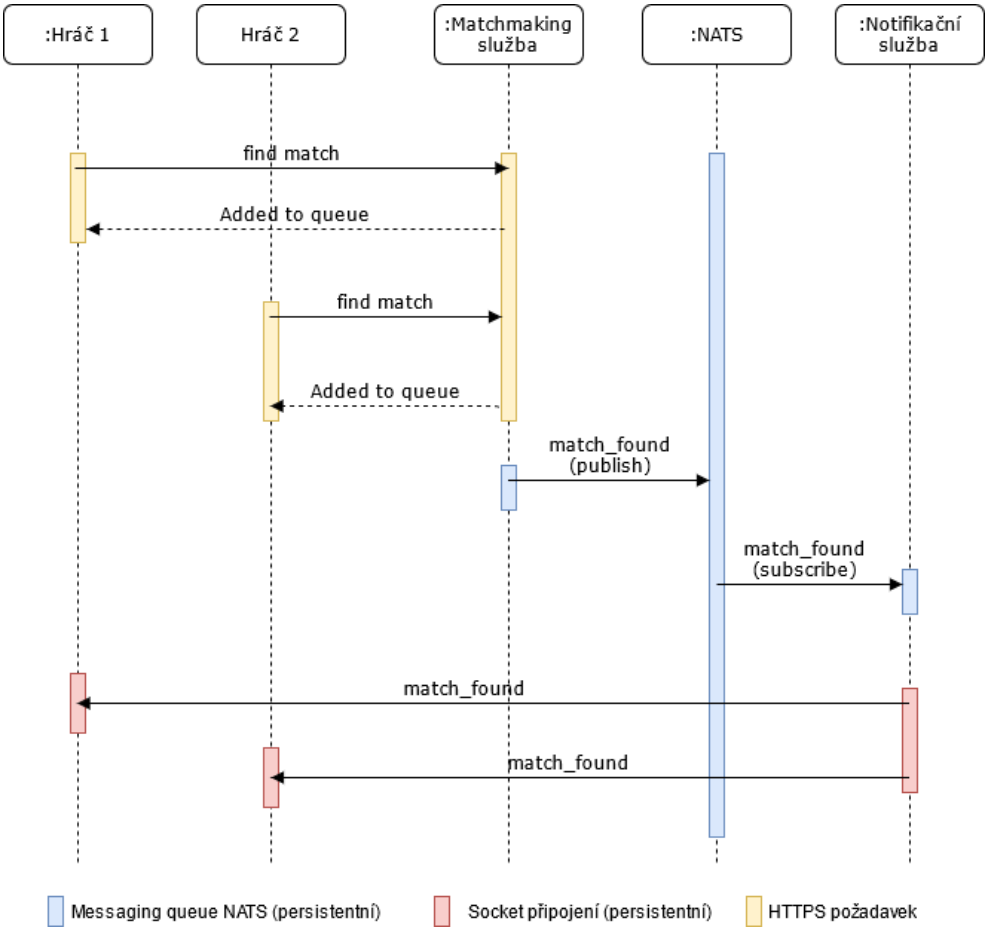
```
POST https://matchmaking.bgrgames.com/queue HTTP/1.1
Content-Type: application/json
...

{
  accessToken: eyJhbGciO ... nI95bDM-Q,
  mode: "solo",
  players: [
    {
      userId: "737dc462-a83d-4da1-93ff-9e9ffc922431",
      latency: 68,
      host: true
    }
  ]
}
```

Výpis kódu 2.4: Příklad požadavku POST na službu matchmaking.

Nyní si na sekvenčním diagramu v obrázku 2.5 ukážeme proces vyhledání hry. Pro přehlednost uvažujme, že hra bude pouze pro dva hráče, a pro konzistenci s kódem uvádějme zprávy v angličtině.

První požadavek, který bude vypadat podobně jako ve výpisu kódu 2.4 pošle *Hráč 1*. Tento požadavek služba matchmaking zpracuje, zařadí hráče do příslušné fronty a obratem vrátí odpověď, že byl hráč do fronty přidán. To samé se stane pro *Hráče 2*. Pokud tedy máme zjednodušený příklad, kde hra je vytvářena pouze pro dva hráče, matchmaking služba tyto dva hráče přiřadí do stejné hry, čímž se kapacita hry naplní. Matchmaking služba musí brát v potaz atribut značící, zda hráč může hru hostovat, pokud by oba hráči měli ve hře zablokovanou možnost hostování, potom tyto dva hráče k sobě matchmaking služba přiřadit nemůže (každá hra musí obsahovat alespoň jednoho hráče, který je schopen hru hostovat, pokud bude takový hráč jen jeden, po jeho případném odchodu ze hry není bohužel hru možno reshostovat a hra tak musí být ukončena). Pokud existuje více hráčů, kteří danou hru mohou hostovat, vybere matchmaking služba hráče z nejmenší odezvou. Následně tedy služba tyto dva hráče k sobě přiřadí a může na NATS server poslat zprávu o tom, že hra byla nalezena (match_found). Tato zpráva musí obsahovat všechny potřebné informace pro notifikační službu, tedy unikátní identifikátory všech hráčů v dané hře a informaci, který z hráčů hru hostuje. Notifikační služba si přebere zprávu z NATS serveru a rozešle notifikace jednotlivým hráčům. Takto tedy bude vypadat běžné vytvoření hry přes matchmaking službu pro hru Inpemo.

Nezamněnujme nebo nespojujme funkcionalitu služby matchmaking se službou sessions. Služba matchmaking pouze říká, kteří hráči spolu budou hrát, ale už je nijak nespojuje mezi sebou, o to se stará služba sessions, kterou hra Inpemo implementuje pomocí služby EOS sessions. Matchmaking služba bude



Obrázek 2.5: Sekvenční diagram pro vyhledání hry pomocí služby matchmaking - simplifikace pro dva hráče.

stejně jako ostatní služby implementována pomocí Node.js, více o implementaci naleznete v následující kapitole.

2.5.8 Friends služba

Toto je služba, kterou nebudeme v rámci této práce implementovat, v rámci návrhu však chceme na této službě ukázat jednoduchost integrace další služby do systému. Veškerá komunikace s touto službou bude probíhat podobně jako se službou matchmaking. Asynchronní požadavky na tuto službu budou předávány na NATS server, odkud si je bude přebírat notifikační služba. Na rozdíl od služby matchmaking bude mít tato služba i synchronní požadavky. Například požadavek na zobrazení všech přátel daného uživatele. Tuto službu podobně jako autentizační službu nemůžeme implementovat pomocí EOS, a to z důvodu, že bychom viděli pouze uživatele přihlášené pomocí dané platformy

(tedy buď jen Steam, nebo jen Epic).

Tato služba bude poskytovat následující požadavky:

- zobrazení seznamu přátel
- přidání přítele
- odebrání přítele
- blokování uživatele
- zobrazení statusu přítele

Pro potřeby hry Inpemo nemusí být tato služba sofistikovaná, ale stačí když bude poskytovat základní funkcionality, které bychom od takové služby čekali. V následující sekci 2.5.10, budeme tuto službu také uvádět jako potencionální rozšíření a budoucí součást celého systému.

2.5.9 Hostování navržených služeb

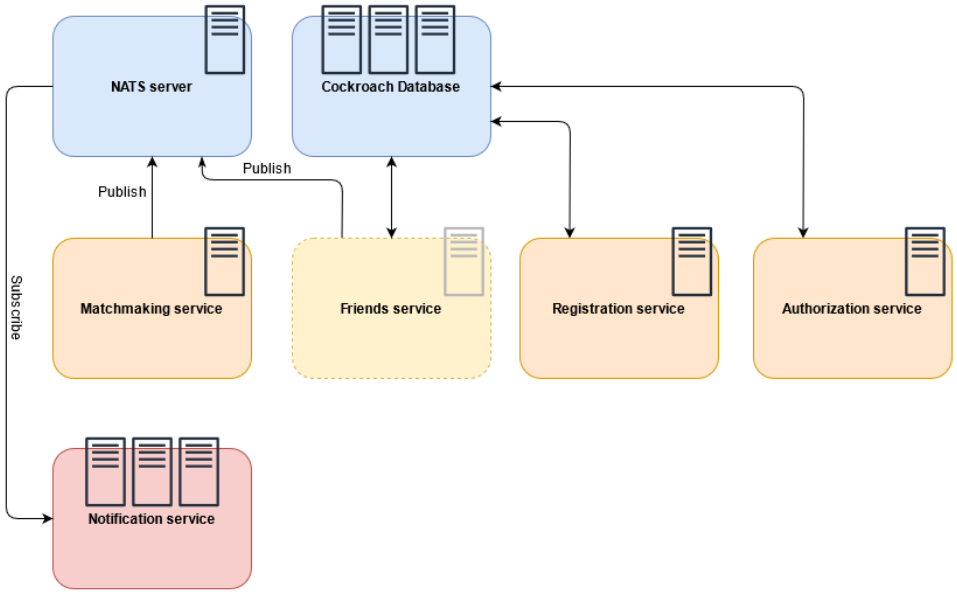
Jak už jsme několikrát zmínili, jedním z našich cílů a nefunkčních požadavků je, aby byly služby pro online hru Inpemo dostupné a spolehlivé, předpoklad pro to je, aby systémy, na kterých tyto služby budou nasazeny, byly spolehlivé. Jak jsme již zmiňovali u dedikovaných serverů, časové i finanční náklady na vlastní hardware jsou značné. Pro představu můžeme zmínit vlastní servery, které jsou použity na interní systémy při vývoji, jako verzovací systém (Perforce), ticketovací systém (Youtrack) a další. Tyto interní servery máme dva. Bylo třeba jim jednak zajistit hardware, jednak disky kvůli zálohování a stabilní připojení k internetu. U těchto serverů navíc nemusíme řešit dostupnost, pokud server spadne, týká se to pouze vývojářů. Vytvořit vlastní spolehlivý systém, který je schopen eliminovat výpadky elektřiny a internetu, není jednoduchý úkol. Řešením které je zároveň velmi flexibilní, je cloudové. Online služby, které navrhujeme, jsou vlastně jen jednoduché Node.js aplikace, které oproti například herním dedikovaným serverům nezabírají tolik hardwarových prostředků. Prostředky navíc můžeme velmi jednoduše měnit dle potřeby, buď vytvářet nové instance, či jednotlivé instance zvětšovat/zmenšovat.

Poskytelů cloudových služeb opět existuje mnoho, se zkušenostmi autora práce bylo rozhodnuto o použití Amazon Web Services. AWS nabízí mnoho služeb, z nichž některé jsme si již dokonce uvedli. Například databázové služby RDS nebo Gamelift, pro vytvoření virtuálních strojů budeme používat EC2, což nám umoňuje vytvořit Windows či Linux virtuální stroj, který si můžeme nakonfigurovat. Více o výběru a konfiguraci si řekneme v další kapitole.

2.5.10 Pohled na celou sadu služeb Bvision

V předchozích sekcích jsme si popsali jednotlivé služby, middleware, který mezi nimi používáme, a také databázový cluster do kterého budeme ukládat

data o hráčích. Na obrázku 2.6 je pohled na celý navržený systém. Červenou barvou je označena nejdůležitější komponenta systému a tou je notifikační služba, oranžovými barvami pak ostatní služby (Služba Friends je označena světlejší barvou, kterou chceme ukázat to, že služba byla do systému pouze navržena, ale v rámci této práce nebude implementována). Modrou barvou jsou označeny interní části systému, které nebudou přímo přístupné hráčům, ale jen jednotlivým službám, které je budou používat a komunikovat s nimi.

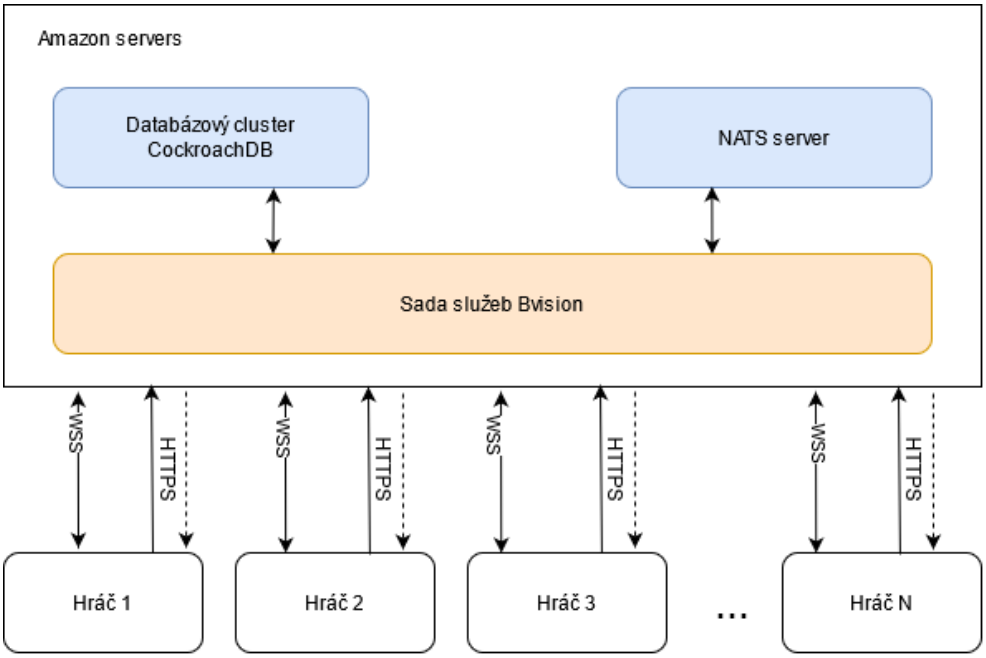


Obrázek 2.6: Sada služeb Bvision, včetně vnější komunikace s instancemi her (hráči).

Na obrázku 2.7 můžeme ještě vidět celý systém zvenčí, kdy se jednotlivé instance hry (hráči) připojují na online služby a komunikují buď přes protokol HTTPS nebo WSS (tento protokol se používá pro persistentní spojení s notifikační službou - socket). Dále je zde pak ještě vidět vrstva za službami, tedy komunikace s databázovým clusterem a NATS serverem.

2.6 Shrnutí kapitoly

V této kapitole jsme vysvětlili potřeby hry Inpemo a ukázali, proč tato hra online služby potřebuje a jakou v rámci této hry služby hrají roli. Provedli jsme analýzu nejen požadavků na tyto služby, ale také technologií, které využijeme na implementaci jednotlivých části systému, ať už šlo o cloudové technologie, databázové systémy, či technologii na přeposílání zpráv, tedy messaging que-



Obrázek 2.7: Jednotlivé komponenty sady online služeb pro hru Inpemo.

ues. Navrhli jsme též vlastní komunikační protokol a specifikovali jsme chování jednotlivých služeb.

Kapitola 3

Realizace online služeb pro hru Inpemo

V této kapitole realizujeme navržené služby a podpůrné systémy z předchozí kapitoly, podrobně popíšeme realizaci a proces nasazování jednotlivých služeb do AWS. Také si ukážeme jak byly jednotlivé služby integrovány přímo do hry Inpemo.

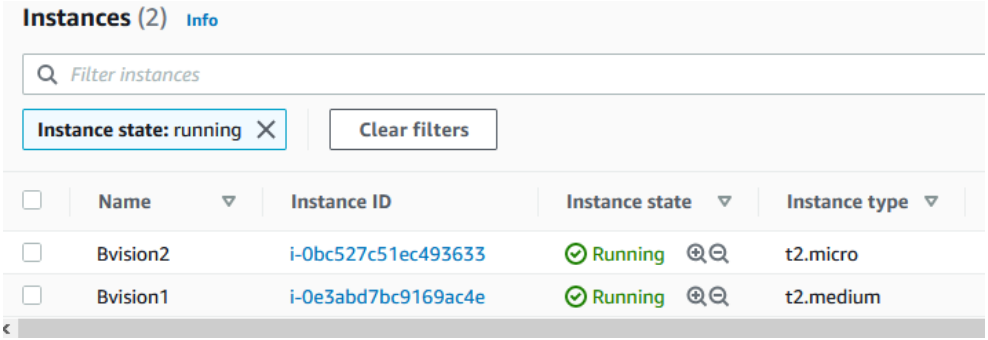
3.1 Výběr a konfigurace EC2 instancí na AWS

Dříve, než začneme konfigurovat databázový cluster a nasazovat jednotlivé služby, je nutné vybrat a nakonfigurovat virtuální servery (instance EC2), na kterých služby poběží.

Vybrali jsme jednu instanci typu `t2.micro`. Tato instance má jeden virtuální procesor z řady Intel Xeon, 1GiB paměti a síťovou propustnost mezi 70-100 MBit/s. Pro online služby tedy využíváme jednu tuto instanci s operačním systémem Ubuntu 18.04 a to z toho důvodu, že ji poskytuje Amazon na rok zdarma. Dále zatím hlavně kvůli testování rozšiřitelnosti máme další instanci `t2.medium`. Tato instance má 2 virtuální procesory ze stejné řady Intel Xeon, 4Gib paměti a síťovou propustnost mezi 250-300 Mbit/s. Prefix t2 značí že se jedná o takzvanou general-purpose instanci z druhé řady. Amazon poskytuje další general-purpose instance s jinými typy procesorů, dále pak instance zaměřené na výkon procesoru používané především na vědecké modelování, strojového učení či dedikované herní servery (compute optimized), na výkon paměti(memory optimized) nebo optimalizované na sekvenční přístup pro čtení a zápis k velkým datům (storage optimized). Druhá instance už však zdarma není a za její používání platíme zhruba 30 USD měsíčně.

Instance `t2.medium` používá stejný operační systém Ubuntu 18.04. Obě instance jsou v regionu us-east-2 v Americkém státě Ohio. To znamená, že odezva na tyto servery je větší, pro testovací účely nám však zatím stačí.

Tento region jsme vybrali na základě ceny, která byla nižší než u evropských regionů. Na obrázku 3.1 můžeme vidět obě instance pojmenované Bvision1 a Bvision2. Obě instance mají přidělenou veřejnou IP adresu a DNS záznam, přes které na ně lze přistupovat.



Obrázek 3.1: Instance ve službě AWS.

Pro co možná největší míru zabezpečení musíme také správně nastavit takzvané *security groups*, ve kterých můžeme zakazovat či povolovat jednotlivé porty (i na jednotlivé IP adresy), obecně tedy chceme, aby například služba, která používá ke komunikaci s hráči protokol HTTPS, měla vystavený pouze odpovídající port 443. Jak jsme si již mohli všimnout v předchozí kapitole ve výpisu kódu 2.4, POST požadavek posíláme na URL, která obsahuje doménu, která byla pořízena v rámci vývoje hry. Jednotlivé DNS záznamy tedy nastavujeme tak, aby směřovali na požadované servery. Dále se v této kapitole ke konfiguraci a zabezpečení instancí ještě vrátíme.

3.2 Realizace databázového clusteru CockroachDB

Jednotlivé instance jsme si již představili v předchozí sekci 3.1. Máme tedy spuštěné dva virtuální Ubuntu servery (Bvision1 a Bvision2). U nich je jako první třeba správně nastavit zmiňované *security groups*.

3.2.1 Nastavení spojená s AWS instancemi

Pro CockroachDB potřebujeme povolit dva porty. Port 26257 pro komunikaci mezi nody a také pro komunikaci mezi službou a nodem (veškeré databázové požadavky se posílají přes tento port). Dále povolíme port 8080, který slouží k přístupu do monitoringu a k přístupu do DB konzole. Z důvodů jednodušího testování povolíme tyto porty pro všechny IP adresy. K monitoringu se nyní dostaneme odkudkoli. V produkci musí být port 8080 buď úplně zakázán, nebo omezen na specifickou síť do, které případně budeme přistupovat přes VPN spojení. Stejně tak port 26257 by měl povolen jen v rámci VPC, jelikož DB, jak víme z minulé kapitoly, používají pouze služby, a nikoli přímo hráči.

Dalším krokem je synchronizace hodin na serverech. Jelikož jsou servery ve stejném regionu (us-east Ohio), je toto vyřešeno za nás.

Nyní je potřeba nastavit load balancer, který rozloží zátěž mezi jednotlivé nody. Jelikož realizujeme cluster přes více serverů, je toto nutnou součástí řešení. Obecně nám load balancer bude posílat požadavky dle vytížení jednotlivých nodů. Pokud některý node selže, load balancer je to schopen rozpoznat a požadavky přeposílat na jiný node. Jako load balancer použijeme také AWS službu, a to network load balancer (NLB). Při nastavování NLB jsme postupovali podle návodu v [31]. Mimo jiné je nutné správně nastavit *Availability Zone*, což udává, které EC2 instance bude moci NLB obsluhovat, pak také příslušné porty, port pro komunikaci DB dotazů 26257 a také port zajišťující kontrolu zdraví nodů, tedy 8080. Na tomto portu bude NLB přistupovat k takzvanému *health endpoint*, který, pokud má node nějaký problém, vrací `503 service unavailable`. Celý endpoint pak vypadá takto: `http://<node-ip>:8080/health?ready=1`. Ještě zmiňme, že NLB od AWS, který zde používáme, je placená služba, která stojí zhruba 20 USD měsíčně.

3.2.2 Konfigurace CockroachDB a spuštění nodů

Pro další konfiguraci již potřebujeme mít nainstalovaný CockroachDB na jednotlivých serverech. To můžeme na Ubuntu udělat buď pomocí Dockeru, K8s, přímým stažením binárního souboru, a nebo sestavit ze zdroje (build from source). Pro jednoduchý cluster bez orchestrace nám stačí stáhnout binární soubor, který si zpřístupníme, vizte výpis kódu 3.1.

```
1 wget -qO- https://binaries.cockroachdb.com/cockroach-v20.2.3.
    linux-amd64.tgz | tar xvz
2 cp -i cockroach-v20.2.3.linux-amd64/cockroach /usr/local/bin/
```

Výpis kódu 3.1: Instalace CockroachDB.

Dále před samotným spuštěním nodů CockroachDB musíme nejprve vygenerovat certifikáty a klíče. V testovacích podmínkách si vystačíme se self-signed CA, navíc jelikož má být databáze přístupná pouze v rámci online služeb, nejedná se o tak velký problém. Pokud bychom self-signed certifikát používali například na nějakém front-endu služby, už by to byl problém o poznání větší, a to i v testovacím prostředí. Prohlížeče self-signed certifikáty označují jako nedůvěryhodné a je pro ně třeba v prohlížeči udělovat výjimku. Typicky, pokud narazíme na stránku se self-signed certifikátem, můžeme vidět hlášku podobnou této: „This certificate is not trusted because it is self-signed.“.

Pojďme si nyní popsat jednotlivé kroky vytváření všech potřebných certifikátů.

Vytvoření certifikační autority (CA)

Nejprve vytvoříme vlastní (self-signed) certifikační autoritu. CockroachDB k těmto účelům má vlastní příkaz. Stejně dobře však můžeme

například použít sadu nástrojů `openssl`. Pro vytvoření CA vizte výpis kódu 3.2.

```
cockroach cert create-ca --certs-dir=certs --ca-key=safe-
    directory/ca.key
```

Výpis kódu 3.2: Vytvoření CA.

Vzniknou nám soubory `ca.crt` a `ca.key`, soubor s klíčem, jak můžete vidět ve výpisu kódu 3.2, bude uložen do adresáře safe-directory, který bychom měli mít velmi dobře zabezpečený.

Vytvoření certifikátů pro jednotlivé nody

Dále pomocí vygenerované CA vytvoříme certifikáty pro nody pomocí příkazu ve výpisu kódu 3.3. Tento příkaz použijeme pro každý node.

```
cockroach cert create-node
    <internal IP> <external IP>
    <hostname> localhost 127.0.0.1
    <NLB IP> <NLB hostname>
    --certs-dir=certs
    --ca-key=my-safe-directory/ca.key
```

Výpis kódu 3.3: Vytvoření node certifikátů.

Také z tohoto příkazu dostaneme dva soubory `node.crt` a `node.key`, které pro každý node musíme na server nahrát (pokud je nevytváříme přímo na něm).

Vytvoření certifikátů pro klienta

Pro výchozího uživatele databáze musíme také vytvořit certifikát. Pomocí příkazu ve výpisu kódu 3.4

```
cockroach cert create-client root --certs-dir=certs --ca-key
    =my-safe-directory/ca.key
```

Výpis kódu 3.4: Vytvoření certifikátů pro root uživatele.

Poté co budeme mít cluster funkční, založíme v DB uživatele noderoach, kterého budou používat online služby. Pro tohoto uživatele budeme také používat certifikát, který vygeneruje podobně jako ten pro uživatele root.

Nyní máme vygenerovány všechny potřebné certifikáty. Ještě jednou zdůrazněme, že všechny certifikáty by měly být bezpečně uloženy a jen tam, kde jsou třeba.

Dále můžeme pokračovat s nastavením jednotlivých nodů. Pro spuštění nodů budeme používat démon `systemd`, který funguje jako správce služeb. Každý node tedy budeme vytvářet jako službu.

Způsobem, jak zvýšit bezpečnost systému, je spouštět každou službu pod vlastním uživatelem. Pokud by případně útočník byl schopen nějakým způsobem jednu službu prolomit a dostat se přímo na náš server, měl by mít pouze

práva daného uživatele. Toto samozřejmě není primární ochrana systému, avšak toto oddělení služeb bývá pravidlem. Vytvoříme tedy nejdříve nového uživatele pomocí příkazu `useradd cockroach`. Uživateli vytvoříme adresář, který bude používat a překopírujeme do něj potřebné certifikáty (adresáři nastavíme patřičná oprávnění). V adresáři pro služby `/etc/systemd/system/` vytvoříme samotnou službu. Vizte službu ve výpisu kódu 3.5.

```
1  [Unit]
2  Description=Cockroach Database cluster node
3  Requires=network.target
4  [Service]
5  Type=notify
6  WorkingDirectory=/var/lib/cockroach
7  ExecStart=/usr/local/bin/cockroach start --certs-dir=certs
       --advertise-addr=3.131.151.10 --join=3.131.151.10,3.21.233.74
        --cache=.25 --max-sql-memory=.25
8  TimeoutStopSec=60
9  Restart=always
10 RestartSec=10
11 StandardOutput=syslog
12 StandardError=syslog
13 SyslogIdentifier=cockroach
14 User=cockroach
15 [Install]
16 WantedBy=default.target
```

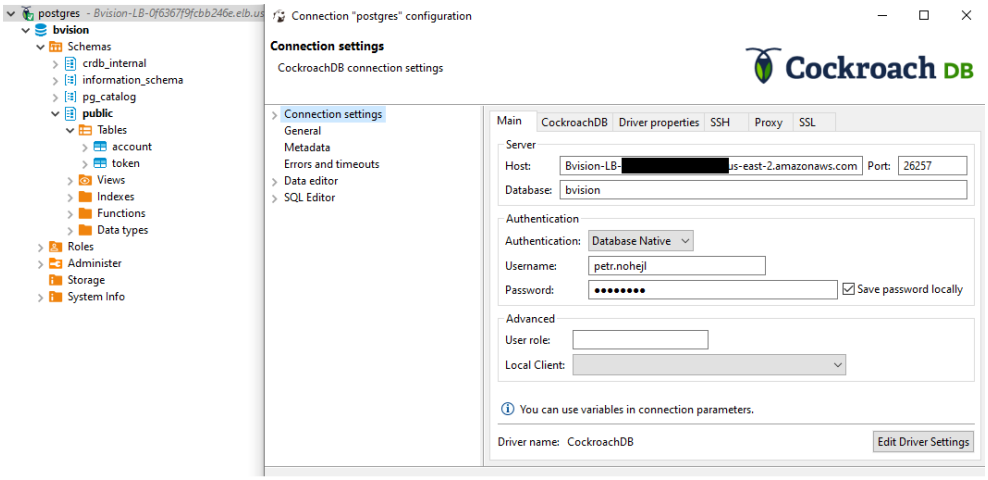
Výpis kódu 3.5: Definice služby cockroachDB nodu.

Na řádku 7 můžeme vidět příkaz, kterým se služba spouští. V parametrech je nastavená složka s certifikáty, IP adresa, na které node běží, dále adresy všech nodů v clusteru a velikost cache a paměti v procentech. Zajímavé jsou také řádky 9 a 10, kde je specifikováno, že služba se má vždy spustit znovu, pokud nějak selže, a že se má případně pokusit o restart po 10 vteřinách.

Službu spustíme příkazem `sudo systemctl start securecockroachdb`, tím pouze spustíme daný node, celý cluster pak ještě musíme inicializovat příkazem `cockroach init --certs-dir=certs --host=< any node IP >`. Více konfiguračních detailů můžeme nalézt v [32].

Poté, co spustíme všechny nody a inicializujeme cluster, již můžeme přistoupit k databázi pomocí `cockroach sql --certs-dir=certs --host=<NLB IP>` příkazu. V konzolovém rozhraní databáze založíme uživatel noderoach, jak jsme zmiňovali výše v této sekci (tento uživatel bude používán online službami) a také mu vygenerujeme potřebné certifikáty, vytvoříme prázdnou databázi s názvem Bvision. Dále vytvoříme uživatele `petr.nohejl`, jehož způsob autentizace je skrze jméno a heslo. Přes tohoto uživatele budeme přistupovat do monitoringu a také administrovat databázi pomocí SQL klienta DBeaver, což je pohodlnější než z konzole. Pro nastavení DBeaver vizte obrázek 3.2.

Práce s databází je takřka k nerozeznání od PostgreSQL (lze používat specifické klausule pro Postgres jako například `WITH` nebo funkci `ROW_NUMBER()`),



Obrázek 3.2: Nastavení SQL DBeaver pro komunikaci s CockroachD.

více k propojení DBeaver a CockroachDB v [33]. Abychom mohli implementovat autentizační a registrační službu, musíme založit odpovídající tabulky v databázi. A to tabulku account a token, skripty pro vytvoření vizte ve výpisech kódu 3.6 a 3.7.

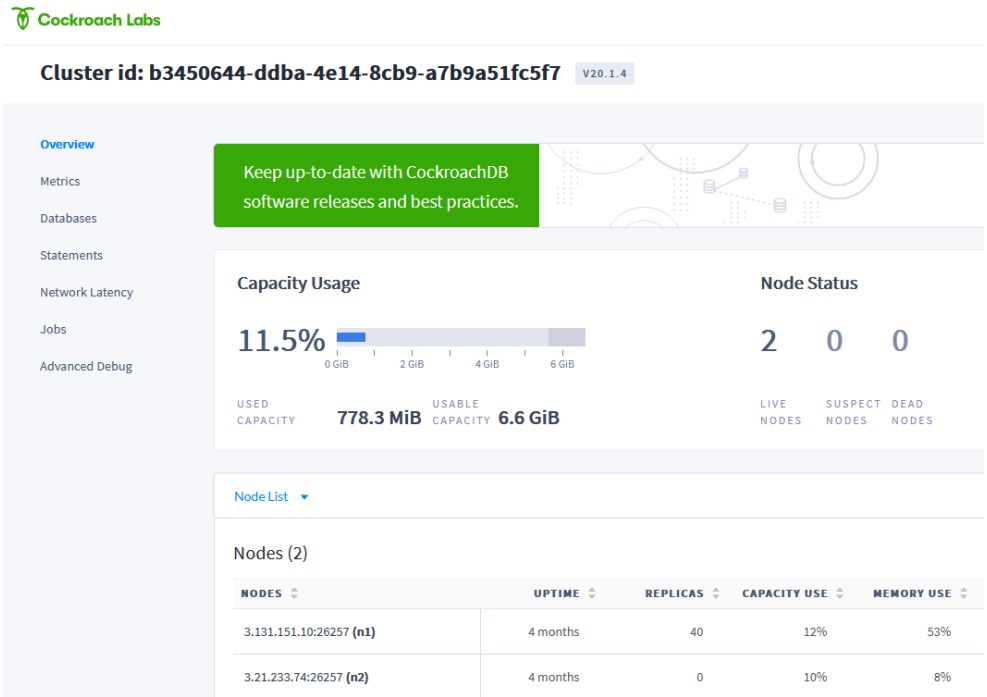
```
CREATE TABLE account (
  bvision_id UUID NOT NULL,
  username VARCHAR(20) NOT NULL,
  realname VARCHAR(100) NULL,
  email VARCHAR(100) NOT NULL,
  password VARCHAR(60) NOT NULL,
  about_account VARCHAR(150) NULL,
  visibility BOOL NOT NULL DEFAULT true,
  phone VARCHAR(20) NULL,
  created_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP,
  last_login_at TIMESTAMPTZ NULL,
  is_verified BOOL NOT NULL DEFAULT false,
  password_reset_token VARCHAR(64) NULL,
  CONSTRAINT account_pk PRIMARY KEY (bvision_id ASC),
  UNIQUE INDEX account_username_key (username ASC),
  UNIQUE INDEX account_email_key (email ASC),
  FAMILY "primary" (bvision_id, username, realname, email,
    password, about_account, visibility, phone, created_at,
    last_login_at, is_verified, password_reset_token)
);
```

Výpis kódu 3.6: Skript pro vytvoření tabulky account.

```
CREATE TABLE token (
  token_id UUID NOT NULL,
  account_id UUID NOT NULL,
  created_at TIMESTAMPTZ NOT NULL DEFAULT CURRENT_TIMESTAMP,
  CONSTRAINT token_pk PRIMARY KEY (token_id ASC),
  CONSTRAINT token_fk FOREIGN KEY (account_id) REFERENCES
    account(bvision_id),
  INDEX token_auto_index_token_fk (account_id ASC),
  FAMILY "primary" (token_id, account_id, created_at)
);
ALTER TABLE public."token" ADD CONSTRAINT token_fk FOREIGN
    KEY (account_id) REFERENCES account(bvision_id);
```

Výpis kódu 3.7: Skript pro vytvoření tabulky token.

Nyní máme funkční databázový cluster s vytvořeným uživatelem, který bude použit v jednotlivých službách. Máme také nastavený SQL klient pro případné úpravy v databázi. Založili jsme dva nody, každý na jiném serveru. Zatížení a zdraví nodů můžeme sledovat v monitoringu. Po přihlášení pomocí jména a hesla na IP adrese jednoho z nodů na portu 8080 můžeme sledovat odezvu jednotlivých nodů, vytížení celého clusteru, jednotlivé SQL dotazy, jejich dobu trvání a mnoho dalšího. Pro ukázku webového rozhraní monitoringu vizte obrázek 3.3.



Obrázek 3.3: Ukázka webového rozhraní CockroachDB monitoringu.

3.3 Realizace autentizační služby

Z počátku vývoje bylo plánováno, že všechny služby budou implementovány pomocí vlastního řešení, až později bylo rozhodnuto na základě skutečností uvedených v předchozí kapitole, že budeme implementovat pouze některé služby a na jiné použijeme sadu služeb EOS. Implementace čistě vlastního řešení by nám umožňovala autentizační službu realizovat prakticky jakýmkoli způsobem. Ze začátku jsme tedy implementovali pouze jednoduchou autentizaci založenou na JWT tokenech, až později jsme přešli na OAuth2.0 server, což při vývoji způsobilo značné zdržení, protože se tato služba musela od základu předělat.

Autentizační služba je webová aplikace napsaná v Node.js. Tato služba a registrační služba jsou jediné služby, které mají front-end. Ve hře Inpemo budeme autorizační server OAuth2.0 používat trochu jiným způsobem než většina webových aplikací. Po úspěšné autorizaci (vystavení autorizačního kódu) nebudeme uživatele nikam přesměrovávat, pouze chceme, aby uživatel mohl dále pokračovat do hry. Přesměrování do hry však není klasickým způsobem možné. Hra je totiž zjednodušeně řečeno nativní aplikace Windows. Hra tak bude kontrolovat redirect URI a, pokud v ní najde autorizační kód, požádá o access token a dále již může pokračovat do hry, kde je vyžadováno, aby byl hráč již přihlášen. Ukázky z integrace do hry uvedeme v následující sekci.

Tato služba používá, jak jsme již zmínili, server `node-oidc-provider`, který umí pracovat s frameworkem `express` [34], který jsme se rozhodli využít. Tento framework poskytuje směrování HTTP požadavků, dle metod a URL. Dále také umožnuje dynamické renderování HTML stránek, k tomu budeme používat Embedded JavaScript templates (EJS). Server node-oidc-provider je velmi dobře konfigurovatelný, konfiguraci si nyní projdeme a dále popíšeme další části autentizační služby.

3.3.1 Konfigurace node-oidc-provider

Pro vytvoření oidc serveru slouží následující řádky ve výpisu kódu 3.8.

```
1 const { Provider } = require('oidc-provider');
2 const oidc = new Provider(BASE_URL + PORT, configuration);
```

Výpis kódu 3.8: Vytvoření OIDC serveru.

Druhý parametr konstruktoru `Provider` budeme nyní zkoumat. Zmíníme pouze některá důležitá nastavení, jelikož celá konfigurace je poměrně rozsáhlá. Jako první si zmíníme pole klientů (objektů) reprezentující jejich metadata. Vizte výpis kódu 3.9.

```
1 clients: [{
2     client_id: process.env.CLIENT_ID,
3     client_secret: process.env.CLIENT_SECRET,
4     grant_types: ['authorization_code', 'refresh_token'],
5     response_types: ['code'],
6     redirect_uris: ['http://bgrgames.com/logincallbackinpemo']
7 }]
```

Výpis kódu 3.9: Konfigurace klientů.

Máme pouze jednoho klienta a tím je samotná hra, na řádcích 2 a 3 můžeme vidět, že hodnoty jsou brány z proměnných prostředí, které jsou pro testování uloženy v souboru, po nasazení na server na Amazonu jsou proměnné nastavovány přímo ve službě (administrované pomocí systemd). Na řádku 4 jsou podporované grant typy. Grant type `authorization_code` jsme již zmínili výše, druhý z nich, `refresh_token`, použijeme, pokud chceme prodloužit platnost access tokenu. Autorizační kód se tedy používá pro samotnou autentizaci a autorizaci (zjednodušeně řečeno přihlášení hráče do hry) a refresh token pro prodloužení access tokenu (to probíhá ve hře na pozadí, hra kontroluje platnost access tokenu a 10 minut před jeho vypršením požádá o nový token pomocí refresh tokenu).

Na řádku 6 vidíme pole `redirect_uris`, které v našem případě obsahuje pouze jeden URL. Tento URL v našem případě nemusí existovat (odkazovat na reálnou stránku), slouží pouze pro kontrolu hry, která URL dále zpracovává a získá z něj parametr `code`.

Další parametry, které si ukážeme, se pojí s access tokenem, jeho formátem a ověřováním. Vizte výpis kódu 3.10.

```
1 formats: {
2     AccessToken: 'jwt',
3 },
4 jwks: {
5     keys: bvision_jwks,
6 }
```

Výpis kódu 3.10: Konfigurace access tokenu a jwks.

Řádky 1 až 3 pouze specifikují formát access tokenu na JWT. V tomto formátu se token jednoduše ověřuje, toho budeme využívat u ostatních služeb, které předtím, než budou cokoli vykonávat, musí zkontrolovat, zda jim byl zaslán access token a zda je platný. Řádky 4 až 6 nastavují JWKs, což je sada kryptografických klíčů používaná k podepisování a ověřování JWT access tokenů. Objekt `bvision_jwks` obsahuje pole jednotlivých JWK (veřejná část klíče). Pro testovací účely používáme jeden JWK, jen uveďme, že v produkčním prostředí by se měly často generovat nové JWK. Server, který používáme, má také dostupný endpoint `/jwks`, ve kterém můžeme najít veřejnou část klíče, ta, jak uvidíme později, se nám bude hodit k ověření JWT tokenů.

Samotné ověřování JWT tokenu budeme podrobněji probírat v implementacích jednotlivých služeb, které ho používají.

Jako poslední si z konfigurace ukážeme nastavení time to live (TTL). Vizte výpis kódu 3.11.

```
ttl: {
    AccessToken: 1 * 60 * 60,        // 1 hour in seconds
    AuthorizationCode: 10 * 60,      // 10 minutes in seconds
    IdToken: 1 * 60 * 60,            // 1 hour in seconds
    DeviceCode: 10 * 60,             // 10 minutes in seconds
    RefreshToken: 1 * 24 * 60 * 60   // 1 day in seconds
}
```

Výpis kódu 3.11: TTL konfigurace.

TTL udává platnost jednotlivých kódů a tokenů. S tím musí počítat hra, musí tato nastavení znát, aby podle nich mohla například včas žádat o obnovení access tokenu.

3.3.2 Přístup do DB a ověření přihlášení

Pro přístup k databázovému clusteru budeme používat framework Sequelize, respektive jeho rozšíření sequelize-cockroachdb. Nastavení tohoto frameworku vizte ve výpisu kódu 3.12. Jedná se o framework, který umožňuje objektově-relační mapování.

```
1  var Sequelize = require('sequelize-cockroachdb');
2  var sequelize = new Sequelize('bvision', 'noderoach', '', {
3    dialect: 'postgres',
4    host: process.env.BVISION_NLB_URL,
5    port: 26257,
6    logging: false,
7    dialectOptions: {
8      ssl: {
9        ca: fs.readFileSync(process.env.HOME_DIR + 'certs/ca.crt').
       toString(),
10       key: fs.readFileSync(process.env.HOME_DIR + 'certs/client.
       noderoach.key').toString(),
11       cert: fs.readFileSync(process.env.HOME_DIR + 'certs/client.
       noderoach.crt').toString()
12     }
13   }
14 });
```

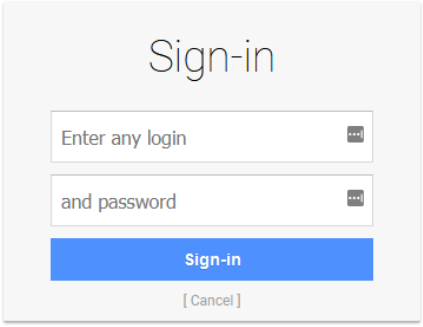
Výpis kódu 3.12: Konfigurace sequalize.

Použijeme zde uživatele `noderoach`, kterého jsme zmiňovali v sekci 3.2, tento uživatel má vygenerované příslušné certifikáty, jejichž použití můžete vidět na řádcích 9 až 11. Parametr host nastavíme na IP adresu network load balanceru, který je také zmiňován v sekci 3.2.

Dále přejdeme k samotnému přihlášení uživatele. Ve hře bude uživatel automaticky přesměrován na přihlášení `https://bvision.bgrgames.com/auth`.

V rámci tohoto požadavku musí být nastaveno několik povinných parametrů. Nastavíme tedy parametry `client_id`, `redirect_uri`, `response_type`, které jsme si již výše uváděli. Dále pak bezpečnostní parametry `nonce` a `state`, první z parametrů slouží k odhalení případného replay útoku a druhý k odhalení zranitelnosti CSRF. Dalším parametrem, který jsme nezmiňovali, je `scope`, ten se používá pro upřesnění, ke kterým informacím daná autorizace probíhá. My budeme používat scope `openid`, což je výchozí scope, dále `userdata`, což je námi definovaný scope, v rámci kterého vracíme například nickname uživatele, a `offline_access` scope, který nám umožnuje práci s refresh tokenem. Posledním parametrem je `prompt`, který nastavíme na hodnotu `consent`, což udává, že se vždy bude zobrazovat obrazovka s potvrzením souhlasu k dané autorizaci.

Nyní si ještě zmíníme technické detaily samotného přihlášení. Zde opravdu myslíme pouze samotnou autentizaci uživatele (obrazovku s přihlašovacím formulářem vizte na obrázku 3.4).



Obrázek 3.4: Ukázka webového rozhraní služby (Formulář pro přihlášení).

Nyní si ukážeme zajímavé části funkce pro autentizaci, kde využijeme framework sequelize a také knihovnu bcrypt [35], kterou budeme také používat v registrační službě pro správu hesel. Zde knihovnu bcrypt budeme používat k porovnání hesel. Funkce `compare` přečte sůl z hashe uloženého v databázi a poté ji použije k hashování hesla, které přišlo na vstup, a provede porovnání hashí.

```
1  app.post('/interaction/:uid/login', setNoCache, body, async (
       req, res, next) => {
2      ...
3      try {
4          let Account = await sequelize.define('account',
       account_table, ...);
5          ...
6          const rows = await Account.findAll({
7              attributes: ['bvision_id', 'username', 'password'
       ],
8              where: { username: req.body.login }
9          });
10         ...
11         if (!bcrypt.compareSync(req.body.password, rows[0].
       password)) {
12             throw new Error('Invalid username or password');
13         } ...
14     } catch (err) {...}
15 }
```

Výpis kódu 3.13: Části funkce pro ověření přihlášení.

Ve výpisu kódu 3.13, můžeme vidět některé zajímavé části funkce, na řádku 4 definujeme mapování mezi tabulkou z databáze s názvem `account` a mezi objektem v JS s názvem `account_table`. Dále v tabulce vyhledáme pomocí funkce `findAll` uživatele (dle dat zadaných do formuláře) a na řádku 11 můžeme vidět ověření hesla pomocí funkce `compareSync` z knihovny bcrypt. Všechny podobné funkce používané v této službě můžeme nalézt v souboru routes.js a objekty, na které mapujeme tabulky z databáze ve složce model.

3.3.3 Důležité endpointy

V této sekci jsme již zmínili dva důležité endpointy `/jwks` a `/auth`. Jsou však ještě další dva, které zde uvedeme.

První z nich je endpoint `/token`, který slouží k vygenerování a nebo obnovení access tokenu (záleží na zvolených parametrech). Pro vygenerování nového tokenu jsou zapotřebí dva parametry. Parametr `grant_type` nastavený na hodnotu `authorization_code` a parametr `code` nastavený na hodnotu autorizačního kódu, který hra získá z redirect URL po úspěšné autentizaci a autorizaci. Pro obnovení access tokenu je první parametr stejný, jen je nastaven na hodnotu `refresh_token` a název druhého parametru se shoduje s hodnotou prvního, tedy `refresh_token` a jeho hodnotou je refresh token samotný. Tento endpoint nám v případě správného zadání parametrů vrací mimojiné access token, refresh token a scopy, na které je token platný. Všechny požadavky, které nejsou provedeny přímo na front-endu, jsou vraceny v JSON formátu.

Druhým endpointem je `/token/introspection`, který je schopen ověřit platnost tokenu. Oba zmíněné endpointy také vyžadují hlavičky s base64

zakódovaným `client_secret` a `client_id`. Přesný formát pro zakódování můžete vidět ve výpisu kódu 3.14.

```
"Basic " + base64UrlEncode(client_id) + ":" + base64UrlEncode(
    client_secret)
```

Výpis kódu 3.14: Vytvoření hlavičky Authorization.

3.3.4 Nasazení autentizační služby na AWS

Pro nasazení této služby potřebujeme nejdříve na server nainstalovat Node.js a správce balíčku npm. Dále pak budeme pokračovat vytvořením služby, kterou budeme spravovat podobně, jako jsme mohli vidět u nodu CockroachDB. Založíme tedy novou službu. Ve výpisu kódu 3.15 můžeme vidět nastavení služby.

```
[Unit]
...
[Service]
...
Environment=HOME=/home/root
Environment=HOME_DIR=/home/pnohejl/oidc/
Environment=PORT=3000
Environment=CLIENT_SECRET=filtered
Environment=CLIENT_ID=filtered
Environment=COOKIE_KEY1=filtered

ExecStart=/usr/local/bin/node /home/pnohejl/oidc/index.js
Restart=always
RestartSec=5
...
```

Výpis kódu 3.15: Nastavení autentizační služby.

Službu spustíme pomocí příkazu `sudo systemctl start bvision-oidc-.service` musíme však ještě zajistit, aby byla služba přístupná z internetu. Jedná se o klasickou webovou aplikaci komunikující přes protokol HTTP, chtěli bychom tak s ohledem na bezpečnost používat zabezpečené spojení, tedy protokol HTTPS.

Nejprve však musíme nainstalovat nějaký HTTP server, pomocí kterého budeme schopni požadavky přicházející na server (EC2 instanci) rozposílat na jednotlivé služby. Vzhledem k předchozím zkušenostem autora této práce s administrací serveru Apache2.4 [36] jsme se rozhodli použít právě tento server, který nainstalujeme a zajistíme, aby byl na serveru (EC2 instanci) otevřený port pro HTTPS, tedy 443. Dále musíme na Apache serveru povolit moduly, které budeme potřebovat. Například proxy modul povolíme pomocí příkazu `sudo a2enmod proxy_http`. Nebudeme zde vyjmenovávat všechny povolené moduly, ještě jeden přesto zmíníme, a to SSL modul, který nám umožní používat protokol HTTPS. Dále pak vytvoříme konfiguraci pro danou službu. Vizte výpis kódu 3.16.

```
1  <VirtualHost *:443>
2      ServerName bvision.bgrgames.com
3      SSLEngine on
4      SSLCertificateFile      /etc/certificate/certificate.crt
5      SSLCertificateKeyFile   /etc/certificate/private.key
6      SSLCertificateChainFile /etc/certificate/ca_bundle.crt
7      ...
8      ProxyPass /          http://127.0.0.1:3000/
9      ProxyPassReverse /   http://127.0.0.1:3000/
10     ...
11 </VirtualHost>
```

Výpis kódu 3.16: Konfigurace Apache serveru pro autentizační službu.

Na řádku 2 můžeme vidět DNS záznam, který jsme napojili na naši EC2 instanci. Doménu `bgrgames.com` jsme již zmiňovali výše v textu, jedná se o zakoupenou doménu v rámci vývoje hry Inpemo. Zde ji používáme ze dvou důvodů. Na první pohled je zřejmé, ze které služby požadavek přišel, a tím druhým důvodem je, že chceme použít protokol HTTPS a, jelikož služba bude veřejná, neměli bychom používat self-signed certifikát. Na DNS záznam, který je přiřazený EC2 instanci od Amazonu, však nelze vytvořit SSL certifikát zdarma (Více v [37]). Certifikáty na řádcích 4 až 6 jsou tedy vydány na DNS záznam uvedený na řádku 2. Certifikát jsme nechali vygenerovat pomocí programu `certbot`, který také umožňuje automatické obnovování certifikátu (platnost certifikátu může kdokoliv ověřit přistoupením na danou subdoménu). Více v [38]. Dále v konfiguraci Apache na řádcích 8-10 máme nastavenou direktivu mapování na back-end (vizte port 3000, na kterém je autentizační služba spuštěna). Pokud provedeme všechna tato nastavení a máme spuštěný Apache server i samotnou autentizační službu, měli bychom k ní být schopni přistoupit z internetu.

Na závěr této sekce ještě zmiňme, že bylo nutné zasahovat do rozšíření frameworku sequelize-cockroachdb, jelikož framewrok není udržovaný a je závislý na modulu pg. Po stažení pomocí `npm i sequelize-cockroachdb` je balíček nefunkční a je třeba upravit závislosti[18].

3.3.5 Integrace autentizační služby do hry Inpemo

Integrace do hry byla prováděna Bc. Michalem Šveigerem za asistence autora této práce se znalostí implementovaných online služeb.

Naše autentizační služba je do hry Inpemo integrována přes plugin EOS Online Subsystem, který je nadstavbou EOS SDK. V rámci tohoto pluginu bylo třeba vytvořit vlastní modul (vlastní autentizační graf[19] , který zajišťuje

[18] Dle komentáře autorů (odpovídajícímu k založenému problému autorem této práce) se bude v budoucnu framework dále udržovat, proto jsme provedli toto dočasné řešení úpravy závislostí. Daný problém můžete najít v [39].

[19] Autentizační graf je struktura vytvořená v rámci tohoto pluginu, do které se přidávají jednotlivé požadavky zajišťující autentizaci.

volání jednotlivých endpointů autentizační služby). V rámci vývoje byl tento plugin pořízen za cenu 120 USD. Více o tomto pluginu zde.

```
1  FString Url = FString::Printf(TEXT("%s/token"), *FBVisionConfig::
       GetBaseUrl());
2  FString UrlParams = FString::Printf(TEXT("code=%s&grant_type=
       authorization_code"), *AuthSubsystem->AuthState->
       AuthorizationCode);
3  FString AuthorizationHeader = FString::Printf(TEXT("Basic %s"), *
       FBVisionConfig::GetBasicAuthorizationValue());
4  TSharedRef<IHttpRequest> Request = FHttpModule::Get().
       CreateRequest();
5  Request->SetURL(Url);
6  Request->SetVerb("POST");
7  Request->SetHeader(TEXT("User-Agent"), "X-UnrealEngine-Agent");
8  Request->SetHeader(TEXT("Content-Type"), TEXT("application/x-www-
       form-urlencoded"));
9  Request->SetHeader(TEXT("Authorization"), AuthorizationHeader);
10 Request->SetContentAsString(UrlParams);
11 Request->OnProcessRequestComplete().BindSP(this, &
       FBVisionGetTokenFromCodeNode::OnTokenReceived,State,OnDone);
12 Request->ProcessRequest();
```

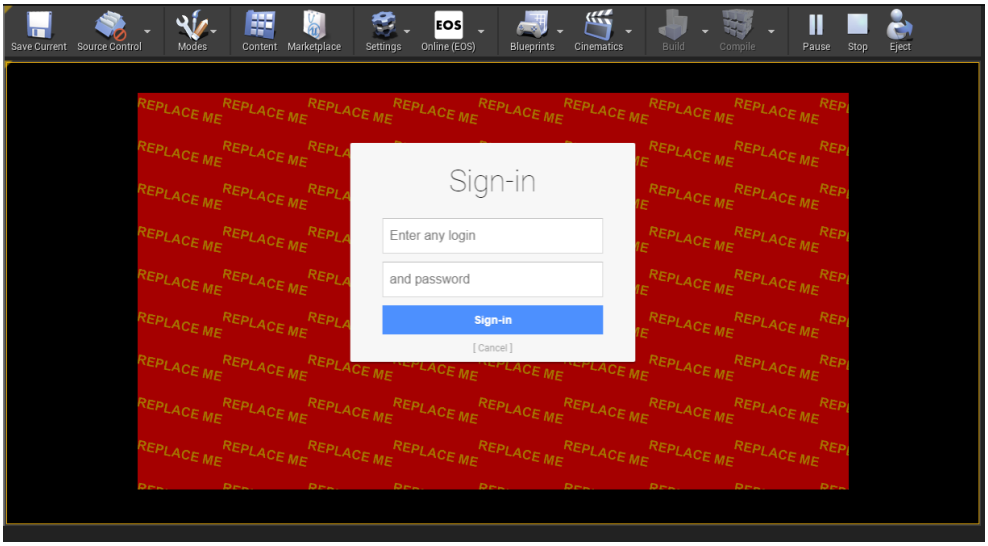
Výpis kódu 3.17: Požadavek na obdržení access tokenu ve hře (C++).

Ve výpisu kódu 3.17 můžeme vidět poslání požadavku na endpoint `/token` (URL vizte na řádku 1). Na řádcích 2 a 10 nastavíme parametry `grant_type` a `code`, jak jsme uváděli v předchozí sekci 3.3.2. Na řádcích 7 až 9 nastavujeme hlavičky požadavku.

Pro přihlášení přes naši autentizační službu, konkétně přes OAuth2.0 server, je třeba spustit prohlížeč ze hry. Můžeme buď otevřít systémový prohlížeč a hru dočasně minimalizovat, nebo využít plugin (zdarma), který umožňuje spouštět prohlížeč přímo ze hry. Tento přístup by šel obejít s využitím formuláře přímo ve hře, to by však neodpovídalo standardu OAuth2.0.

Výsledné řešení můžete vidět na obrázku 3.5. Hra Inpemo se stále vyvíjí a tedy i ve snímku obrazovky můžeme vidět texturu s nápisem *Replace me* a za ní černé pozadí. Černé pozadí je samotný level ve hře, který může obsahovat 3D objekty. Část obrazovky s texturou *Replace me* je součástí widgetu. Widget v UE4 je instance 3D komponenty, která se často používá pro vytvoření UI hry (tak je tomu i v našem případě)[20]. Můžeme vidět, že součástí obrazovky je pouze formulář z webového rozhraní autentizační služby, toho jsme docílili použitím Web browser pluginu, který umožňuje zobrazovat pouze obsah stránky a navíc zprůhlednit pozadí. Hráč tedy zdánlivě prohlížeč nespouští, ale vidí z něj pouze tento formulář. Více o pluginu v [40].

[20]Pozadí a popředí je zde tedy trochu zavádějící pojem. Ve skutečnosti jsou objekty na snímku přímo v levelu a toto zobrazení je dáno pozicí kamery. To odpovídá popisu výše, kde je uvedeno, že Widget je instance 3D komponenty.

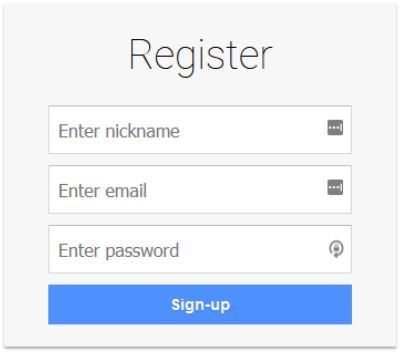


Obrázek 3.5: Formulář pro přihlášení v UE4 (ve hře Inpemo).

3.4 Realizace registrační služby

Registrační služba používá stejné technologie jako autentizační služba: pro komunikaci s databází nadstavbu frameworku sequelize-cockroachdb, framework `express` pro směrování HTTP požadavků a EJS pro dynamické vykreslování HTML. Tato služba má tedy také front-end, graficky stejně jednoduše stylovaný jako autentizační služba.

Nyní si popíšeme endpointy této služby. Pokud metodou GET přistoupíme na endpoint `/register`, otevře se nám formulář podobný tomu z předchozí sekce 3.3 (vizte obrázek 3.6).



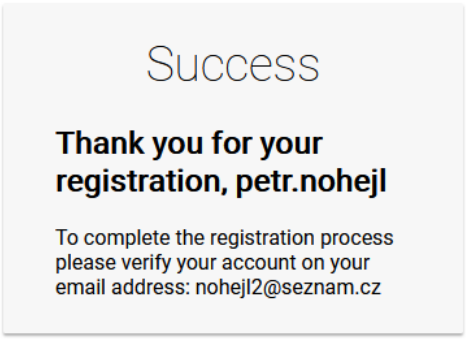
Obrázek 3.6: Formulář pro registraci.

Po vyplnění údajů a odeslání formuláře pošleme požadavek na vytvoření uživatele, kde je mimo jiné kontrolována minimální délka hesla, duplicita v přezdívce hráče (nickname) a duplicita v emailu. Pokud projdou údaje všemi kontrolami, použijeme funkci `hash` z knihovny bcrypt, která je běžně využívaná pro práci s hesly ve webových aplikacích. Poté použijeme funkci `sequelize.transaction`, ve které vytvoříme uživatele a zároveň ověřovací token. Dále už pouze odešleme ověřovací mail na danou emailovou adresu. Pro odesílání mailů jsme zvolili knihovnu nodemailer (vizte výpis kódu 3.18).

```
1  const transporter = nodemailer.createTransport({
2    host: 'smtp.forpsi.com',
3    port: 465,
4    secure: true,
5    auth: {
6      user: process.env.EMAIL_ADDRESS,
7      pass: process.env.EMAIL_PASSWORD
8    }
9  });
10 ...
11 transporter.sendMail(mailOptions, function (err) {...});
```

Výpis kódu 3.18: Použití knihovny nodemailer.

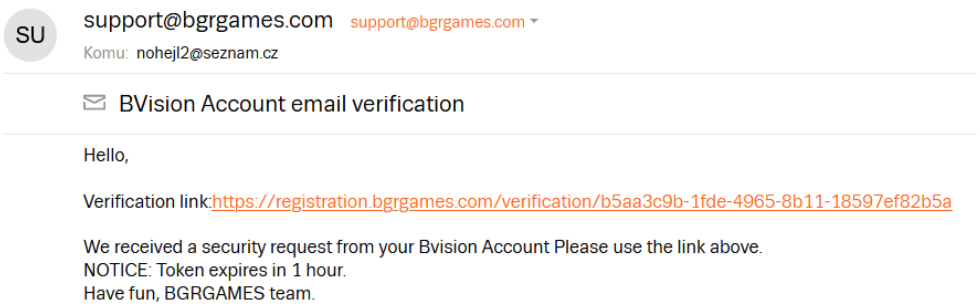
Jak můžete vidět na řádku 2, pro odesílání mailů používáme SMTP server poskytovatele domény, kde máme také emailové adresy. V produkčním prostředí by bylo vhodné použít vlastní SMTP server, abychom nebyli závislí na serveru třetí strany. Poté, co je celá registrace zpracována, by měl uživatel být úspěšně registrován (vizte obrázek 3.7).



Obrázek 3.7: Obrazovka úspěšné registrace.

V zaslaném emailu najdeme odkaz na `/verification` endpoint, který ověřuje, zda je daný token platný a zda náleží danému uživateli. Pokud ano, aktualizujeme u uživatele v databázi příznak `is_verified`. Na obrázku 3.8 můžeme vidět zaslaný email.

Po kliknutí na odkaz v emailu ještě není provedena verifikace, ale pouze vidíme obrazovku, kde je tlačítko *verify*, to je z toho důvodu, že samotné



Obrázek 3.8: Verifikační email zaslaný z registrační služby.

ověření mění stav aplikace (konkrétně záznam v databázi), dle správného návrhu by tento požadavek měl být typu POST [41]. Pro ověření vyžadujeme o jedno kliknutí navíc, avšak zachováme tím správný návrh systému.

Pokud by při posílání emailu došlo k problému, implementujeme také endpoint `/resend`, pomocí kterého můžeme přeposlat email na danou emailovou adresu.

3.4.1 Nasazení registrační služby na AWS

Zde bude nasazení probíhat téměř stejně jako u autentizační služby. Opět nejdřív založíme službu a spustíme ji. Na apache serveru založíme SSL konfiguraci pro tuto službu a nastavíme správný DNS záznam. Vizte výpis kódu 3.19.

```
1 <VirtualHost *:443>
2     ServerName registration.bgrgames.com
3     ...
4     ProxyPass /          http://127.0.0.1:3001/
5     ProxyPassReverse /   http://127.0.0.1:3001/
6     ...
7 </VirtualHost>
```

Výpis kódu 3.19: Konfigurace Apache serveru pro registrační službu.

Změníme tedy direktivu `ServerName`, upravíme port u direktiv `ProxyPass` a `ProxyPassReverse` a změníme cesty k certifikátům, které opět vygenerujeme pomocí certbota. Poté by měla být i tato služba přístupná z internetu.

3.5 Nasazení NATS serveru

NATS server budeme v této práci používat pouze pro komunikaci mezi službou matchmaking a notifikační službou, není tedy nutné mít službu vytvořenou jako cluster, samotné vytvoření clusteru je však velmi jednoduché. Stačí nám

při spouštění instance NATS serveru specifikovat dva přepínače: `-cluster` a `-routes`. První server (node), který bychom spustili, by byl takzvaný seed server, u kterého definujeme pouze přepínač `-cluster` s daným URL a portem. U ostatních pak specifikujeme oba přepínače, přičemž přepínač `-routes` směřuje vždy na seed server. Veškeré nastavení jinak probíhá stejně jako u serveru s pouze jednou instancí.

Pro NATS server vytvoříme opět systemd službu, která bude NATS server spouštět pod vlastním uživatelem (vizte výpis kódu 3.20).

```
1  [Unit]
2  Description=NATS messaging server
3
4  [Service]
5  ExecStart=/srv/nats/bin/gnatsd -c /srv/nats/gnatsd.config
6  User=nats
7  Restart=on-failure
8
9  [Install]
10 WantedBy=multi-user.target
```

Výpis kódu 3.20: Konfigurace služby spouštějící NATS.

Na řádku 5 vidíme, že se server spouští pomocí `gnatsd`[21]s přepínačem `-c` specifikující konfigurační soubor. Do toho se nyní také podíváme (vizte výpis kódu 3.21).

```
1  port: 4222
2  net: '127.0.0.1'
3  tls {
4    cert_file: "/srv/nats/priv/gnatsd.crt"
5    key_file: "/srv/nats/priv/gnatsd.key"
6    timeout: 1
7  }
8  authorization {
9    user: nats-server
10   password: filtered
11 }
```

Výpis kódu 3.21: Konfigurace NATS serveru.

V konfiguračním souboru vidíme standardně port, URL a nastavení `tls`, kde jsme opět vygenerovali certifikáty. Jedná se o interní server, který bude komunikovat pouze s ostatními službami, takže jsme pro zjednodušení použili self-signed certifikát. Dále zde ještě máme nastavení `authorization`, které vyžaduje autentizaci pomocí jména a hesla. Tímto máme server nastavený, klienti (jednotlivé služby) se budou připojovat skrze NATS Node.js framework, to si však uvedeme až u jednotlivých služeb.

[21]Binární soubor pomocí, kterého NATS server spouštíme.

3.6 Realizace notifikační služby

Tato služba již vyžaduje autentizaci, jinými slovy, aby měl hráč platný access token. Jako první tedy tento token musíme ověřit. K tomu v Node.js použijeme knihovny `node-jose` [42] a `jsonwebtoken` [43]. V autentizační službě jsme vygenerovali pro autorizační server JWK a ukázali jsme si, že k získání jejich veřejné části můžeme využít endpoint `/jwks`. Rozhodli jsme se však tento endpoint nepoužívat a veřejnou část klíče uložit přímo na server. Přístup klíče tak bude rychlejší, nebude závislý na autentizační službě a zároveň nebude autentizační službu zatěžovat. Uveďme příklad, kdy bychom měli deset služeb, které vyžadují autentizaci a nemají perzistentní spojení (ke komunikaci používají protokol HTTPS). Pak budeme posílat pro každý požadavek od každého hráče na takovou službu také požadavek na získání JWK. Pokud bychom endpoint přesto použili, výhodou by bylo, že JWK by byly vždy aktuální.

Pokud tedy JWKs přegenerujeme, budeme muset obnovit soubory na serveru tak, aby služby používaly aktuální JWK, to je však jen malá nevýhoda oproti ostatním výše zmíněným. Pojďme nyní na výpisu kódu 3.22 ukázat ověření access tokenu. Více informací o fungování a ověřování JWT tokenů pomocí JWK lze nalézt v [44].

```
1  const certificate = await jose.JWK.asKey(bvision_public_jwk);
2  const pemCertificate = certificate.toPEM(false);
3  // check data in headers
4  if(request.headers.authentication && request.headers['bvision-
       identifier']){
5      await jwt.verify(request.headers.authentication,
       pemCertificate, {algorithms: ['RS256']});
6      callback(null, request.headers['bvision-identifier']);
7      return;
8  }
9  // check data in url query
10 ...
```

Výpis kódu 3.22: Ověření access tokenu.

Objekt `bvision_public_jwk` obsahuje veřejnou část klíče. Funkce z knihovny jsonwebtoken `jwt.verify` přijímá jako druhý parametr certifikát v pem formátu. Proto musíme nejprve certifikát do tohoto tvaru zkonvertovat. Na řádku jedna z něj uděláme `jose.JWK.Key` instanci a poté na něj aplikujeme funkce `toPEM`. Celý výpis kódu je v try-catch bloku a pokud funkce verify vyhodí výjimku, autentizace selže.

Po autentizaci tedy můžeme pokračovat k další implementaci serveru, další dvě knihovny, které používáme, jsou ws (websocket, [45]) a nats klient [46]. Místo knihovny ws byla původně použita knihovna socket.io [47], což je nadstavba nad knihovnou ws, která umí zprávy ze socketů rozdělovat do topiců a zjednodušovat tak práci se sockety. Také umí využívat jak websockety, tak techniku polling pro klienty, kteří nejsou schopni websocket použít. Našimi klienty však bude vždy hra, takže víme, že využití websocketu je možné. Největší

problém, na který jsme narazili při použití socket.io, je však fakt, že používá vlastní komunikační protokol [48]. Je tedy nutné používat socket.io klienta (existuje klient v C++, který by šel integrovat do hry). Integrace tohoto klienta by však byla náročná. Do UE4 existuje i plugin, který daného klienta používá, jenže ten, jak bylo zjištěno po bližším zkoumání, má několik problémů (Implementuje funkcionality, které v rámci hry nepotřebujeme, a je tak pro naše potřeby zbytečně robustní. Integrace samotného pluginu je složitá.). Z tohoto důvodu používáme knihovnu ws, která komunikuje přes protokol WS nebo jeho zabezpečenou verzi WSS a k tomu není zapotřebí žádný dedikovaný klient.

Dále bylo v notifikačním serveru také implementováno použití protokolu Bvision, které jsme navrhli v sekci 2.5.6. Implementovali jsme třídy pro oba typy zpráv `BvisionEvent` a `BvisionMessage`, které zajišťují serializaci a deserializaci zpráv. Ve výpisu kódu 3.23 můžeme vidět příklad použití obou tříd.

```
1  const message = new BvisionMessage().fromString(msg);
2  if(message.messageCode === 101){
3      const event = new BvisionEvent();
4      event.serviceUrl  = 'https://notification.bgrgames.com';
5      event.serviceCode = 10;
6      event.timestamp   = new Date().getTime();
7      event.eventType   = 'pong';
8      event.eventCode   = 101;
9      ws.send(event.toString());
10 }
```

Výpis kódu 3.23: Použití deserializace zprávy typu BvisionMessage a serializace zprávy BvisionEvent.

Zprávu, která přijde na server, deserializujeme pomocí funkce `fromString`. Ze zprávy poté zjistíme `messageCode`. Pokud jím je 101 (odpovídá pingu od hráče), tak pošleme odpověď (pong). Nastavíme všechny parametry a zprávu na řádku 9 serializujeme a pošleme hráči po websocketu.

Dále v notifikačním serveru implementujeme jednu z klientských částí NATS serveru. Připojení k NATS serveru zajistíme funkcí `NATS.connect`, kde musíme dle konfigurace NATS serveru zadat certifikát a také heslo. Dále již můžeme pomocí metody subscribe brát zprávy z NATS serveru, které mohou přijít například od služby matchmaking. Ve výpisu kódu 3.24 můžeme vidět zjednodušené použití funkce subscribe.

```
nc.subscribe('match_solo', (players) => {
    for(const player of players){
        if(wsClients.has(player.userId)){
          const playerWebSocket = wsClients.get(player.userId
    );
          const event = new BvisionEvent();
          event.serviceUrl  = 'https://matchmaking.bgrgames.
    com';
          event.serviceCode = 11;
          event.timestamp   = new Date().getTime();
          event.eventType   = 'match_found';
          event.eventCode   = 201;
          event.payload     = players;
          playerWebSocket.send(event.toString());
        } ...
    } ...
});
```

Výpis kódu 3.24: Použití klientské části NATS serveru.

Zde je ukázka, kde čekáme na zprávu typu `match_solo`, poté projdeme všechny hráče, kteří v této zprávě figurovali, a pokud s nimi máme navázané spojení přes socket, pošleme jim zprávu o tom, že jejich hra byla nalezena.

3.6.1 Nasazení notifikační služby na AWS

Vytvoření služby (systemd) bude probíhat velmi podobně jako u předchozích služeb. V návrhu jsme tuto službu navrhovali clusterovanou a tak by taky měla jít do produkčního nasazení, v rámci implementace této služby jsme provedli implementaci prototypu jednoho nodu. Pokud bychom aplikaci chtěli rozšířit na cluster, použili bychom technologii k8s, která však vyžaduje minimálně 2 vCPU, jedna z instancí má pouze 1 vCPU. Na větší instanci jsme k8s nainstalovali, máme tedy připravené prostředí pro případné rozšíření této služby při dokoupení další EC2 instance.

Po nastavení a spuštění služby pro notifikace je opět nutné nastavit Apache server, zde bude nastavení jiné než u předchozích služeb. Musíme totiž počítat jak s HTTP požadavky, tak s připojením přes WSS. Povolíme tedy nový modul `mod_rewrite`. Ve výpisu kódu 3.25 můžeme vidět konfiguraci, která je v Apache navíc oproti původním službám.

```
RewriteCond %{HTTP:Connection} Upgrade [NC]
RewriteRule /(.*) ws://localhost:3002/$1 [P,L]

ProxyPass /socket wss://localhost:3002/socket
ProxyPassReverse /socket wss://localhost:3002/socket

ProxyPass /socket ws://localhost:3002/socket
ProxyPassReverse /socket ws://localhost:3002/socket
```

Výpis kódu 3.25: Část konfiguračního souboru (Apache) pro notifikační server.

3.6.2 Integrace notifikační služby do hry Inpemo

Již jsme zmínili výběr knihovny pro websockety, který musel být v návaznosti na integraci změněn. Ve hře musela být implementována stejně jako na serveru serializace a deserializace JSON zpráv protokolu Bvision. A v tuto chvíli je hra schopna také zjišťovat odezvu hráče na notifikační službu.

Ve výpisu kódu 3.26 si ukážeme připojení k notifikační službě přímo z UE4 (ze hry).

```
void UBVisionNotificationSubsystem::InitConnection()
{
    ...
  const FString ServerURL = TEXT("wss://notification.bgrgames.com
    /socket");
  const FString ServerProtocol = TEXT("wss");
  TMap<FString, FString> Headers;
  Headers.Add(TTuple<FString, FString>(TEXT("Authentication"),
    AuthSubsystem->GetAccessToken()));
  Headers.Add(TTuple<FString, FString>(TEXT("BVision-Identifier")
    , TEXT("464161351351313")));
  Socket = FWebSocketsModule::Get().CreateWebSocket(ServerURL,
    ServerProtocol, Headers);
    ...
  Socket->Connect();
}
```

Výpis kódu 3.26: Části funkce InitConnection integrující připojení k notifikační službě.

Služby jsme samozřejmě netestovali přímo ve hře, ale pomocí vlastních testů a jednoduchého Javascriptového klienta, více o testování si však ukážeme v následující kapitole.

3.7 Realizace služby matchmaking

Tuto službu jsme implementovali jako poslední, bude stejně jako notifikační služba implementovat ověřování access tokenu pomocí knihoven `node-jose` a `jsonwebtoken` a také bude používat stejného NATS klienta po připojení na NATS server. V této službě také používáme framework `express`, pro směrování HTTP metod a URL. Aplikace však nemá žádný front-end, veškeré odpovědi na požadavky jsou vraceny v JSON formátu.

Endpoint, který budou hráči (instance hry) používat, bude `/queue` a to metodu POST. Pro týž endpoint implementujeme i metodu GET (pro vypsání všech front) a metodu DELETE (pro promazání fronty). Tyto dvě metody (GET a DELETE) implementujeme jen pro testovací účely, pokud chceme vidět informace o aktuálním stavu serveru, případně pokud nastane problém, aby šla fronta jednoduše promazat a nemuseli jsme restartovat celou službu. Fronty pro matchmaking jsou ukládány v paměti služby, tedy nejedná se o perzistentně ukládaná data. Jako další, především testovací, endpoint je

`/capacity`, který lze použít s metodou PUT. Tímto požadavkem jsme schopni upravit kapacitu fronty, neboli maximální počet hráčů, který bude moci jedna hra pojmout. Zmíněné testovací metody budou použity pouze v testovacím prostředí, v následné produkční verzi nebudou metody k dispozici (zakázány pomocí nastavení proměnné prostředí).

Zdůrazněme, že hra Inpemo nemá v tuto chvíli navržený žádný systém rankování (ohodnocení) hráčů. Vzájemné seskupování hráčů se řídí jen dle požadavků na typ hry. Druhotným faktorem je odezva a možnost hráče hru hostovat. Hra Inpemo si neklade za cíl být kompetitivní, ale spíše spadá do kategorie oddechových/zábavných her a tomu také odpovídá tato služba.

Služba samozřejmě před zařazením do fronty kontroluje, zda je zvolený podporovaný mód a zda je v daném módu dodržen počet hráčů ve skupině, který daný mód může hrát. Pokud bychom zvolili mód duo, který dovoluje skupiny maximálně po dvou lidech, server skupiny o větším počtu do fronty nezařadí. Pokud vše splňuje zařazení do fronty, server jednoduchým algoritmem hledá ve všech hrách dané fronty, které nejsou obsazeny, a pokud je v dané hře dostatek volných míst, potom hráče z daného požadavku do hry přidá. Pojmem hra zde myslíme jen objekt v paměti, který drží informace o hráčích a počtu hráčů v dané hře. Pokud neexsituje hra vyhovující daným kritériím, založí se nová hra (nový objekt). Jako první se zkouší přidávat do her, které jsou vytvořeny nejdéle (dá se přirovnat k principu FIFO, některé hry se však přeskakují pokud nesplňují daná kritéria). Pokud se přidáním do hry naplní kapacita hry, je hra z fronty vyjmuta a pošle se zpráva na NATS server. Samotné poslání zprávy je velice jednoduché. Funkce k poslání zprávy může vypadat například takto `nc.publish('match_solo', parsedPlayersData);`.

Ještě před tím, než se daná hra vyjme z fronty a pošle se zpráva na NATS server, vybereme hosta. Zde uvádíme jednoduchou funkci na výběr hosta. Vizte výpis kódu 3.27

```
function calculateHost(session){
    let parsedPlayers = []; //pid and host flag
    let hostBestFitByLatency = session.players[0];
    let index = 0, bestFit = 0;
    for(const player of session.players){
        parsedPlayers.push({
            userId: player.userId,
            hostFlag: false });
        if(player.host &&
           player.latency < hostBestFitByLatency.latency){
            hostBestFitByLatency = player; bestFit = index;
        }; index++;
    }
    parsedPlayers[bestFit].hostFlag = true;
    return parsedPlayers;
}
```

Výpis kódu 3.27: Funkce pro výběr hosta hry.

Pro nasazení matchimaking služby je opět třeba vytvořit službu (systemd), spustit ji a vytvořit konfigurační soubor pro Apache server, vytvořit certifikáty na odpovídající DNS záznam, tedy `matchmaking.bgrgames.com`. Poté by měla být také tato služba dostupná z internetu. Vytvořená matchmaking služba je funkční a nasazená na serveru, tedy nic nebrání její integraci do hry. Byla testována Javascriptovým klientem na různé druhy požadavků, jak si ukážeme v následující kapitole.

Kapitola 4

Testování online služeb pro hru Inpemo

V této kapitole popíšeme, jakým způsobem jsme testovali jednotlivé služby a jaké typy testů jsme nad službami prováděli. Tyto testy si ukážeme a na závěr zkusíme navrhnout, jak by šlo testování těchto služeb ještě vylepšit.

4.1 Vývoj a nastavení testovacího prostředí

Všechny služby byly vyvíjeny a testovány nejprve lokálně mimo servery AWS na operačním systému Windows. Později se při implementaci ukázalo, že to nebylo ideální řešení, jelikož některé knihovny byly závislé na operačním systému a jejich přenositelnost nebyla bezproblémová (jedná se například o knihovnu `pg`). Jako vývojové prostředí jsme použili Visual Studio Code (VS Code), které umoňuje debuggovat kód Node.js serveru. Toho jsme při implementaci často využívali. CockroachDB cluster byla jediná technologie zprovozněná a nasazená na AWS serverech od začátku vývoje. Jeho lokální nasazení do testovacího prostředí na Windows by byla zbytečná práce navíc.

V rámci vývoje jsme také používali verzovací systém Perforce (P4)[22][49], který slouží také k vývoji samotné hry Inpemo. Do VS Code je možné nainstalovat plugin, který s P4 umí pracovat, je pak tedy jednoduché soubory přemisťovat z testovacího prostředí na AWS servery. Na AWS serveru stačí nakonfigurovat připojení k P4 serveru, pomocí konzolové utility `p4 sync` poté jen stahujeme potřebné verze souborů z p4 serveru.

Ve Windows jsme také spouštěli NATS server i in-memory databázi Redis. Druhá ze zmíněných má velice špatnou podporu na Windows a nejde pustit

[22] Verzovací systém P4 je běžně používaný při vývoji her a to především z důvodu přístupu práce s binárními soubory (v UE4 to jsou textury, assety, blueprinty a mnoho dalších). V rámci vývoje těchto služeb bychom tedy mohli stejně dobře zvolit například Git, jelikož s binárními soubory zde nepracujeme, avšak motivací použití P4 bylo mít online služby i hru samotnou v rámci jednoho verzovacího systému.

přímo jako Windows service, ale pouze z konzole a to ve velmi neaktuální verzi. S ostatními službami již nebyly žádné větší problémy. Samozřejmě jsme se snažili o to, mít co nejpodobnější verze samotného node.js i správce balíčku npm v obou prostředích (Windows i AWS servery).

4.2 Testování CockroachDB clusteru

Po nasazení clusteru jsme museli ověřit, zda cluster funguje, jak má. V rámci toho jsme tedy posílali požadavky nejprve přímo na jednotlivé nody a poté na NLB, který požadavky na nody přeposílal. V autentizační službě i registrační službě jsme při vývoji DB cluster používali a fungoval bez problémů, avšak množství požadavků, které na cluster bylo zasíláno v rámci vývoje, bylo relativně zanedbatelné.

Rozhodli jsme se proto udělat zátěžový test clusteru, kdy jsme vytvořili zkušební workload, který v rámci určité časové doby posílal požadavky na cluster. Během toho jsme mohli měřit, zda se nějak zvyšuje odezva (latence) na jednotlivé nody.

Jako první jsme si tedy vytvořili zkušební databázové schéma (workload), na kterém jsme testy prováděli. Vizte výpis kódu 4.1.

```
sudo cockroach workload init movr 'postgresql://root@Bvision-LB-0
    f6367f9fcbb246e.elb.us-east-2.amazonaws.com:26257?sslmode=
    verify-full&sslrootcert=ca.crt&sslcert=client.root.crt&sslkey
    =client.root.key'
```

Výpis kódu 4.1: Vytvoření testovacího workloadu.

Toto schéma se jmenuje `movr`, jedná se o zkušební databázové schéma, které poskytuje přímo CockroachDB právě k testovacím účelům. V příkazu můžeme vidět, že požadavek se posílá na NLB a jako parametry se mu samozřejmě musí dát certifikáty (v tomto případě jsme použili uživatele root).

Dále pak můžeme spustit samotný zátěžový test pomocí příkazu ve výpisu kódu 4.2.

```
sudo cockroach workload run movr --duration=5m 'postgresql://
    root@3.21.233.74:26257?sslmode=verify-full&sslrootcert=ca.crt
    &sslcert=client.root.crt&sslkey=client.root.key'
```

Výpis kódu 4.2: Spuštění testovacího workloadu.

Pomocí přepínače `--duration` můžeme nastavit, jak dlouho test poběží. Těchto přepínačů je celá řada, více o nich můžeme najít v [50]. Můžeme například nastavovat, kolik záznamů se bude vkládat do tabulek, či jestli má být nějak limitován počet zápisů/čtení a podobně.

Hlavním faktorem, který jsme při tomto testování zkoumali, bylo, zda nějaký node nebude mít výrazně zvýšenou latenci. Na to jsme i po změně různých parametrů nenarazili. Testování probíhalo na 24 různých workerech (lze nastavit pomocí přepínače –concurency) a to několikrát během několika

desítek minut se zvýšenými počty záznamů v databázi (v řádech tisíců záznamů). Musíme brát v potaz, že servery, na kterých je cluster spuštěn, jsou výkonostně jedny z nejhorších, které Amazon poskytuje. Proto jsme se nesnažili zkoušet extrémní hodnoty, které by servery mohly zahltit.

Jednotlivé parametry jsme mohly zkoumat přes monitoring databáze na jednom z nodů. Monitoring je přístupný například na adrese serveru Bvision1 `https://3.21.233.74:8080`, ovšem pouze po přihlášení.

V rámci testování vytížení databázového clusteru jsme tedy nezjistili, že by cluster fungoval špatně, či měl při mírné dlouhodobější zátěži nějaké výpadky či zvýšenou latenci.

4.3 Testování služeb

Začneme s autentizační službou, ve které jsme převážně používali `node-oidc-provider`, jak je několikrát zmíněno v předchozích kapitolách. V rámci použití této knihovny jsme prováděli mnoho úprav, většina z nich spočívala ve správné konfiguraci celého OAuth2.0 serveru a také přepisování chování některých endpointů. U toho bylo hojně využíváno zmíněné debugování v rámci VS Code a další manuální testování pomocí programu Postman[51], který umožňuje posílat HTTP požadavky s nastavením hlaviček, parametrů i těl požadavků.

Stejně jako registrační služba funguje autentizační služba především na vyplňování a posílání formulářů. Tuto funkcionalitu jsme testovali manuálně, ale také jsme si vyzkoušeli napsat funkcionální testy na jejich automatické zasílání. Jelikož je tato funkcionalita integrována do hry Inpemo, samotná hra může po dobu vývoje sloužit jako testovací prostředí pro tyto služby, zejména požadavky pro přihlášení uživatele.

K vytváření funkcionálních testů jsme použili knihovny `jest` [52] a `zombie` [53], první z knihoven nám umožnuje pouštět samotné testy, druhá knihovna vyplňovat automaticky formuláře a posílat HTTP požadavky. Ve výpisu kódu 4.3 si můžeme jeden funkcionální test pro registrační službu ukázat.

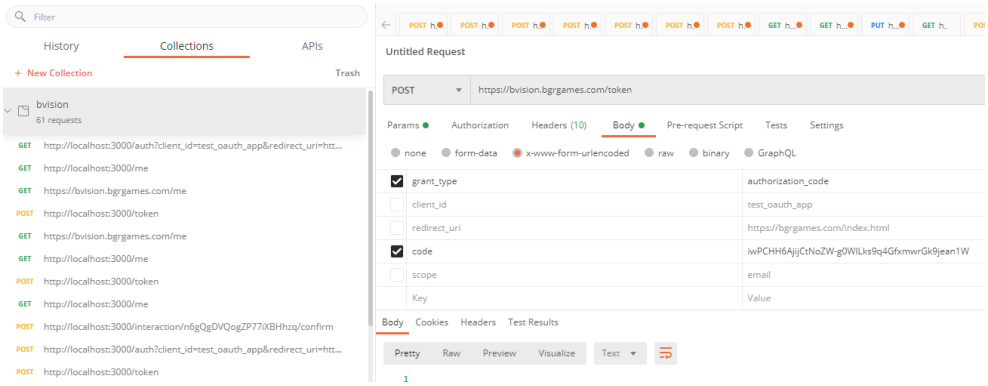
V tomto testu tedy zkoušíme přistupovat na endpoint `/register`, na kterém se posléze snažíme vyplňovat formulář (uživatelské jméno, email a heslo). Také kontrolujeme, zda byl uživatel registrován (v tomto funkcionálním testu není řešeno ověřování pomocí emailu).

Obecně služby, které jsme v této práci implementovali, jsou mezi sebou provázané a komunikují jednak mezi sebou, ale také se hrou Inpemo, pro služby matchmaking, notifikační službu a autentizační službu jsme tedy funkcionální testy nevytvářeli a spoléhali se především na integrační testování a na end-2-end testování a na klienta Postman, který umoňuje uchovávat HTTP požadavky s danými parametry, hlavičkami a dalšími nastaveními. Také pomocí něj lze spouštět celé sady testů seřazené jako test-cases, s vlastním api na testování odpovědí. Testovali jsme takto desítky endpointů ze všech imple-

mentovaných služeb (kromě registrační služby) Na obrázku 4.1 můžete vidět rozhraní klienta Postman. V levé části jsou některé testované endpointy a v pravé části je pak přímo jeden požadavek v tomto případě na endpoint `/token`.

```
const Browser = require('zombie');
Browser.localhost('localhost', 3001);
describe('User visits signup page', function() {
  const b = new Browser();
  beforeAll(function() {
    return b.visit('/register');
  });
  describe('submits form', function() {
    beforeAll(function() {
      b.fill('email', Math.random().toString(36) + '@test.cz')
      .then(() => b.fill('username', Math.random().toString(36)))
      .then(() => b.fill('password', 'ThisIsGr8Password'))
      .then(() => b.pressButton('Sign-up', done));
    });
    it('should be successful', function() {
      b.assert.success();
    });
    it('should see success page', function() {
      b.assert.text('title', 'Registration success | BGR Games');
    });
  });
});
```

Výpis kódu 4.3: Funkcionální test pro registrační službu.



Obrázek 4.1: Ukázka Postman klienta.

Klienta Postman však nemůžeme použít u všech služeb. Hráči s notifikační službou, jak víme z předchozích kapitol, komunikují pomocí socketů. Klient Postman však nabízí pouze komunikaci pomocí HTTP. Pro testování notifikační a matchmaking služby jsme se tak rozhodli využít jednoduchého javascriptového klienta. Na službu matchmaking bychom samozřejmě HTTP dotazy posílat mohli, a tak bychom také mohli používat klienta Postman.

Pomocí samotné matchmaking služby, kdy nebudeme mít otevřené spojení na notifikační službu, však nedostaneme notifikace o tom, že byla hra nalezena. Proto tedy do klienta implementujeme také požadavky na matchmaking službu.

Zmiňovaný JS klient je ve skutečnosti pouze HTML stránka s javascriptovým kódem, kterou můžeme otevřít v prohlížeči. Pokud tak učiníme, automaticky se nám naváže spojení na notifikační službu (pokud máme platný access token z autentizační služby). Poté můžeme posílat požadavky na službu matchmaking a pokud je nějaká hra vytvořena, obdržíme odpovídající notifikaci. Pro účely testování jsme zrušili kontrolu unikátnosti hráče ve službě matchmaking, pokud tedy hráč zašle vícekrát požadavek na službu matchmaking, všechny jeho požadavky jsou přijaty tak, jako by se jednalo o jiného hráče (access token musí být stále platný). To nám umožňuje službu testovat pouze s jedním autentizovaným a autorizovaným hráčem.

Prohlížeče nám přímo umožňují používat WebSocket API. Připojení k notifikačnímu serveru pomocí socketu je velmi jednoduché. Vizte výpis kódu 4.4.

```
1 const socket = new WebSocket('wss://notification.bgrgames.com/
    socket?accessToken=' + testAccessToken);
```

Výpis kódu 4.4: Připojení socketu v prohlížeči.

Můžeme zde vidět použití protokolu `wss`. Zároveň musíme v parametrech poslat platný token. Dále jsme, podobně jako přímo ve hře, schopni zjišťovat odezvu hráče. Vizte výpis kódu 4.5.

```
1  socket.addEventListener('open', function (event) {
2      setInterval(function() {
3        startTime = Date.now();
4        socket.send(JSON.stringify({msg_type: "ping", msg_code:
       101, timestamp: startTime}));
5      }, 2000);
6  });
7  socket.addEventListener('message', function (event) {
8      console.log('Message from server:', event.data);
9      let jsonData = JSON.parse(event.data);
10     if(jsonData && jsonData.event_type === "pong"){
11       latency = Date.now() - startTime;
12       if(latencies.length > 9){
13         latencies.shift();
14       }
15       latencies.push(latency);
16       console.log(latencies);
17     }
18 });
```

Výpis kódu 4.5: JS klient (získávání odezvy).

Na řádcích 1 až 6 můžeme vidět event listener `open`, tedy pokud je navázáno spojení pomocí secketu se serverem, každé dvě vteřiny pošleme ping na

server, pomocí kterého zjišťujeme odezvu. Na zbylých řádcích máme event listener `message`, který přijímá zprávy přicházející po socketu. Zde konkrétně ukládáme odezvu (latenci), pokud nám přichází zpráva typu `pong`. Můžeme si také povšimnout, že zprávy jsou posílány pomocí Bvision protokolu, který jsme implementovali. U zprávy typu `ping` není atribut `payload`, jedná se totiž o nepovinný atribut. Takto jednoduše jsme tedy schopni zachytávat veškeré zprávy procházející po socketu mezi notifikační službou a klietem.

Stránka obsahuje několik tlačítek, každé odkazující na nalezení hry v jiném módu. Implementaci odesílání jednoho požadavku (mód duo) můžeme vidět ve výpisu kódu 4.6.

```
1  $("#findMatchDuo").click(function(event){
2  ...
3  $.ajax({
4      url: "https://matchmaking.bgrgames.com/queue",
5      type: "POST",
6      data: {
7        accessToken: AccessToken,
8        mode: "duo",
9        players: [
10         {
11           userId: playerId1,
12           latency: latencies.reduce(getSum, 0)/latencies.length,
13           host: true
14         },
15         {
16           userId: playerId2,
17           latency: latencies.reduce(getSum, 0)/latencies.length,
18           host: true
19         }
20       ]
21     },
22     dataType: "json",
23     ...
24   });
25 });
```

Výpis kódu 4.6: JS klient (matchmaking požadavek).

Tento požadavek se provede po kliknutí na tlačítko s id `findMatchDuo`. Jak si můžeme povšimnout, používáme zde knihovnu jQuery [54] a z ní funkci `ajax`, pomocí které jsme schopni (jednodušeji než pomocí samotného JS) posílat HTTP požadavky. Zde posíláme požadavek na připojení do hry typu *duo* a to pro skupinu dvou hráčů. Jak můžeme vidět například na řádcích 11 až 13, posíláme informace o jednotlivých hráčích (id, odezvu a zda je hráč schopen hostovat).

Tento JS klient byl pro testování nutný, samotná integrace do hry byla zdlouhavá a navíc v UE4 je obecně propojení s webovými technologiemi problematičtější a složitější. To jsme si vyzkoušeli při integraci, kdy jsme použili socket.io a následně museli službu přepisovat, aby podporovala místo socket.io

knihovnu ws.

4.4 Shrnutí testování

Všechny navržené služby a i další použité technologie (CockroachDB) byly v rámci práce otestovány různými způsoby (funkcionální testy, integrační testy, end-2-end testy). V rámci dalšího vývoje by bylo dobré dále rozvíjet JS klienta, kterého jsme uváděli v předchozí sekci. Pokud bychom později uváděli do produkce notifikační službu clusterovanou, bylo by dobré udělat výkonnostní testy této služby, abychom veděli, kolik spojení je služba schopná pojmout.

Závěr

Uveďme na závěr, že byly navrženy a implementovány čtyři online služby pro hru Inpemo. Služba autentizační, registrační, notifikační a služba matchmaking. Udělali jsme tak základ systému pro všechny ostatní služby, které se budou moci k těmto službám připojit, ať už se jedná například o friends službu, nebo službu pro statistiky. Autor této práce se v rámci jejího plnění zdokonalil v administračních činnostech spojených s konfigurací AWS a Ubuntu, dále se pak seznámil s databázovým clusterem CockroachDB. Velmi důkladně také musel porozumět fungování OAuth2.0 serveru a OIDC. V rámci jednotlivých služeb se pak měl možnost seznámit s mnoha knihovnami, jako například `jsonwebtoken`, `jest`, `ws` a mnoha dalšími.

Do budoucna je plánováno pokračovat s vývojem jednotlivých služeb, které zde byly zmíněny, a dále je integrovat do hry Inpemo, která by na nich v budoucnu měla být závislá. Může se zdát, že nasazení CockroachDB clusteru byl relativně jednoduchý úkon, z počátku vývoje však zabral velké množství času. Dalším velkým problémem bylo rozhodování, zda bude ve hře Inpemo použit EOS či Steamworks, pokud by byl použit Steamworks nebylo by nutné implementovat autentizační a registrační službu. Další implementace služeb již probíhala relativně bez problémů až na uvedenou změnu z knihovny `socket.io` na knihovnu `ws`.

Literatura

[1] Datapath.io. The History of Online Gaming. [online], Leden 2020, [cit. 2020-10-12]. Available from: https://medium.com/@datapath_io/the-history-of-online-gaming-2e70d51ab437

[2] Corporation Valve. Joining The Steamworks Distribution Program. [online], [cit. 2020-10-15]. Available from: https://partner.steamgames.com/steamdirect

[3] Statt, N. Valve's new Steam revenue agreement gives more money to game developers. [online], Listopad 2018, [cit. 2020-11-05]. Available from: https://www.theverge.com/2018/11/30/18120577/valve-steam-game-marketplace-revenue-split-new-rules-competition

[4] Epic Games. Online Subsystem Steam. [online], [cit. 2020-11-05]. Available from: https://docs.unrealengine.com/en-US/ProgrammingAndScripting/Online/Steam/index.html

[5] Byford, S. Steam hit its all-time concurrent user peak over the weekend. [online], Březen 2020, [cit. 2020-11-06]. Available from: https://partner.steamgames.com/steamdirect

[6] Grubb, J. Epic Games Store is seeing near-Steam-like engagement levels. [online], Červen 2020, [cit. 2020-11-06]. Available from: https://venturebeat.com/2020/06/23/epic-games-store-engagement/

[7] Epic Games. Epic game store revenue split. [online], [cit. 2020-11-06]. Available from: https://www.epicgames.com/store/en-US/about

[8] Epic Games. Lobby Interface. [online], [cit. 2020-11-06]. Available from: https://dev.epicgames.com/docs/services/en-US/Interfaces/Lobby/index.html

[9] Epic Games. Sessions Interface. [online], [cit. 2020-11-06]. Available from: https://dev.epicgames.com/docs/services/en-US/Interfaces/Sessions/index.html

[10] Heroic Labs. Unreal client guide. [online], [cit. 2020-11-08]. Available from: https://heroiclabs.com/docs/unreal-client-guide/

[11] Kyro. Battle.net Chat Server Protocol Overview. [online], Červen 2013, [cit. 2020-11-08]. Available from: https://bnetdocs.org/document/10/battle-net-chat-server-protocol-overview

[12] Matulef, J. Battle.net has been hacked. [online], Srpen 2012, [cit. 2020-11-10]. Available from: https://www.eurogamer.net/articles/2012-08-10-battle-net-has-been-hacked-e-mails-compromised

[13] Epic Games. Identity Provider Management. [online], [cit. 2020-11-10]. Available from: https://dev.epicgames.com/docs/services/en-US/DevPortal/IdentityProviderManagement/index.html

[14] OpenJS Foundation. About Node.js. [online], [cit. 2020-12-31]. Available from: https://nodejs.org/en/about/

[15] Hardt, D. The OAuth 2.0 Authorization Framework. [online], October 2012, [cit. 2020-11-10]. Available from: https://tools.ietf.org/html/rfc6749

[16] Bradley, J. OpenID Connect Core 1.0 incorporating errata set 1. [online], 2014, [cit. 2020-11-10]. Available from: https://openid.net/specs/openid-connect-core-1_0.html

[17] Skokan, F. oidc-provider. [online], [cit. 2020-12-31]. Available from: https://github.com/panva/node-oidc-provider

[18] Amazon Web Services. Amazon RDS for PostgreSQL features. [online], 2020, [cit. 2020-11-25]. Available from: https://aws.amazon.com/rds/postgresql/features/

[19] Amazon Web Services. Amazon RDS for PostgreSQL Pricing. [online], 2020, [cit. 2020-11-25]. Available from: https://aws.amazon.com/rds/postgresql/pricing/

[20] Amazon Web Services. DB instance billing for Amazon RDS. [online], 2020, [cit. 2020-11-25]. Available from: https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/User_DBInstanceBilling.html

[21] MongoDB. Sharded Cluster Components. [online], [cit. 2020-11-28]. Available from: https://docs.mongodb.com/manual/core/sharded-cluster-components/

[22] Cockroach Labs. Manual Deployment. [online], [cit. 2020-12-02]. Available from: https://www.cockroachlabs.com/docs/stable/manual-deployment.html

[23] KeyCDN. TLS 1.2 vs TLS 1.1. [online], Říjen 2018, [cit. 2020-12-22]. Available from: https://www.keycdn.com/support/tls-1-2-vs-tls-1-1

[24] Cockroach Labs. CockroachDB Security. [online], [cit. 2020-12-02]. Available from: https://www.cockroachlabs.com/docs/stable/security-overview.html

[25] Reeder, T. Asynchronous Messaging Patterns. [online], Říjen 2019, [cit. 2020-12-05]. Available from: https://blogs.mulesoft.com/dev-guides/api-design/asynchronous-messaging-patterns/

[26] Collison, G. Explore NATS Pub/Sub. [online], 2020, [cit. 2020-12-07]. Available from: https://docs.nats.io/developing-with-nats/tutorials/pubsub

[27] RabbitMQ. Publish/Subscribe (using the amqp.node client). [online], [cit. 2020-12-08]. Available from: https://www.rabbitmq.com/tutorials/tutorial-three-javascript.html

[28] DataFlair. Advantages and Disadvantages of Kafka. [online], [cit. 2020-12-08]. Available from: https://data-flair.training/blogs/advantages-and-disadvantages-of-kafka/

[29] Johansson, L. Apache Kafka for beginners - What is Apache Kafka? [online], Březen 2020, [cit. 2020-12-08]. Available from: https://www.cloudkarafka.com/blog/2016-11-30-part1-kafka-for-beginners-what-is-apache-kafka.html

[30] Amazon Web Services. Amazon GameLift Pricing. [online], 2020, [cit. 2020-12-12]. Available from: https://aws.amazon.com/gamelift/pricing/

[31] Amazon Web Services. Getting started with Network Load Balancers. [online], 2020, [cit. 2020-12-12]. Available from: https://docs.aws.amazon.com/elasticloadbalancing/latest/network/network-load-balancer-getting-started.html

[32] Cockroach Labs. Deploy CockroachDB on AWS EC2. [online], [cit. 2020-12-25]. Available from: https://www.cockroachlabs.com/docs/stable/deploy-cockroachdb-on-aws.html

[33] Cockroach Labs. DBeaver. [online], 2020, [cit. 2020-12-12]. Available from: https://www.cockroachlabs.com/docs/stable/dbeaver.html

[34] OpenJS Foundation. Express. [online], [cit. 2020-12-31]. Available from: https://expressjs.com/

[35] Kelektiv. Bcrypt. [online], [cit. 2020-12-31]. Available from: https://github.com/kelektiv/node.bcrypt.js

[36] The Apache Software Foundation. Apache 2.4. [online], [cit. 2020-12-31]. Available from: https://httpd.apache.org/

[37] GeoffRussell. Policy forbids issuing name for AWS? [online], Červen 2019, [cit. 2020-12-12]. Available from: https://community.letsencrypt.org/t/policy-forbids-issuing-name-for-aws/95246

[38] Electronic frontier foundation. Certbot instructions. [online], [cit. 2020-12-12]. Available from: https://certbot.eff.org/lets-encrypt/ubuntubionic-apache

[39] Nohejl, P. Latest npm version of this package is 1.1.0. [online], Listopad 2020, [cit. 2020-12-28]. Available from: https://github.com/cockroachdb/sequelize-cockroachdb/issues/49

[40] Epic Games. Web Browser. [online], [cit. 2020-12-31]. Available from: https://docs.unrealengine.com/en-US/InteractiveExperiences/UMG/UserGuide/WidgetTypeReference/WebBrowser/index.html

[41] Jacobs, I. URIs, Addressability, and the use of HTTP GET and POST. [online], Březen 2004, [cit. 2020-12-28]. Available from: https://www.w3.org/2001/tag/doc/whenToUseGet.html

[42] Cisco Systems. node-jose. [online], [cit. 2020-12-31]. Available from: https://github.com/cisco/node-jose

[43] Auth0. jsonwebtoken. [online], [cit. 2020-12-31]. Available from: https://github.com/auth0/node-jsonwebtoken

[44] Meyer, S. Navigating RS256 and JWKS. [online], Červen 2020, [cit. 2020-12-28]. Available from: https://auth0.com/blog/navigating-rs256-and-jwks/

[45] WebSockets. ws: a Node.js WebSocket library. [online], [cit. 2020-12-31]. Available from: https://github.com/websockets/ws

[46] nats-io. NATS.js - Node.js Client. [online], [cit. 2020-12-31]. Available from: https://github.com/nats-io/nats.js

[47] socket.io. socket.io library. [online], [cit. 2020-12-28]. Available from: https://socket.io/

[48] socket.io. socket.io-protocol. [online], [cit. 2020-12-28]. Available from: https://github.com/socketio/socket.io-protocol

[49] Perforce Software. Helix Core. [online], [cit. 2020-12-31]. Available from: https://www.perforce.com/products/helix-core

[50] Cockroach Labs. Cockroach workload. [online], 2020, [cit. 2020-12-29]. Available from: https://www.cockroachlabs.com/docs/stable/cockroach-workload.html

[51] Postman. Postman. [online], [cit. 2020-12-31]. Available from: https://www.postman.com/

[52] Facebook open source. Delightful JavaScript Testing Framework. [online], [cit. 2020-12-31]. Available from: https://jestjs.io/

[53] Arkin, A. Zombie.js - Insanely fast, headless full-stack testing using Node.js. [online], [cit. 2020-12-31]. Available from: http://zombie.js.org/

[54] OpenJS Foundation. jQuery. [online], [cit. 2020-12-31]. Available from: https://jquery.com/

Příloha A

Seznam použitých zkratek

AES Advanced Encryption Standard

API Application programming interface

AWS Amazon Web Services

CA Certificate Authority

CPU Central processing unit

CSRF Cross-site Request Forgery

DB Database

DNS Domain Name System

DTLS Datagram Transport Layer Security

DWANGO Dial-up Wide-Area Network Game Operation

EC2 Elastic Compute Cloud

ECC Elliptic Curve Cryptography

EJS Embedded JavaScript templates

EOS Epic Online Services

FIFO First-in, first-out

HTML Hypertext Markup Language

HTTPS Hypertext Transfer Protocol Secure

HW Hardware

IP Internet Protocol

JSON JavaScript Object Notation

JWK JSON Web Key

JWT JSON Web Token

LAN Local area network

MDTC Microsoft Distributed Transaction Coordinator

NAT Network address translation

NATS Neural Autonomic Transport System

NLB Network Load Balancer

OIDC OpenID Connect

P4 Perforce

P2P Peer-to-peer

RDS Relational Database Service

rUDP Reliable User Datagram Protocol

SaaS Software as a service

SDK Software development kit

SMTP Simple Mail Transfer Protocol

SQL Structured Query Language

SSL Secure Sockets Layer

TTL Time to live

TLS Transport Layer Security

UE4 Unreal Engine 4

URI Uniform Resource Identifier

URL Uniform Resource Locator

USD United States dollar

vCPU virtual Centralized processing unit

VPC Amazon Virtual Private Cloud

VPN Virtual Private Network

VS Visual Studio

WS WebSocket

WSS WebSocket Secure

WebRTC Web Real-Time Communication

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF