

Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Bot na platformě Discord



2023

Vedoucí práce:
Mgr. Roman Vyjídaček

Lukáš Netřeba

Studijní program: Informatika, prezenční
forma

Bibliografické údaje

Autor:	Lukáš Netřeba
Název práce:	Bot na platformě Discord
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2023
Studijní program:	Informatika, prezenční forma
Vedoucí práce:	Mgr. Roman Vyjídáček
Počet stran:	55
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliographic info

Author:	Lukáš Netřeba
Title:	Bot on Discord platform
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2023
Study program:	Computer Science, full-time form
Supervisor:	Mgr. Roman Vyjídáček
Page count:	55
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

Obsahem práce je vytvoření Bota pro uživatele a komunity nacházející se na komunikační platformě Discord. Tato aplikace umožní bota používat pro správu komunit, nastavením jejich pravidel, pravomocí a moderace chatu. Bot dále umožní vytvářet a organizovat události, na které se uživatelé z komunit budou moci přihlásit a být upozorněni před jejich začátkem. Dále bude zprostředkovávat audio přehrávač skrze hlasové kanály na serverech.

Synopsis

Content of this theses aims to create a Bot for users and communities on Discord platform. Application will provide management for those communities with their own set of rules, permissions and chat moderation. The bot will take care of creating and organizing scheduled events for community users that can participate in and be notified just before the event starts. The bot will also mediate an audio player in voice channels on the servers.

Klíčová slova: Discord, nástroj pro správu, Python, Bot, audio přehrávač, události

Keywords: Discord, administration tool, Python, Bot, audio player, scheduled events

Chtěl bych poděkovat svému vedoucímu práce, panu Mgr. Romanu Vyjídáčkovi, za to, že mi umožnil zpracovat toto téma a za jeho odbornou pomoc při jejím vypracování. Dále bych chtěl poděkovat svým blízkým a kolegům, kteří byli se mnou trpěliví a motivovali mě k vypracování práce.

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce

podpis autora

Obsah

1	Úvod	9
2	Platforma Discord	9
2.1	Definice a terminologie	10
2.2	Servery a komunity	11
2.3	Kanály hlasové a textové	11
2.4	Uživatelé, role a pravomoce	12
2.5	Limitace a předplatné	12
3	Discord Bot	14
3.1	Ovládání a práce s botem	15
3.1.1	Prefixované zprávy	15
3.1.2	Lomítkové příkazy	16
3.2	Technologie pro vývoj	17
3.2.1	Discord API	17
3.2.1.1	WebSocket API	17
3.2.1.2	REST API	18
3.2.2	Asyncio	18
3.2.3	FFmpeg	19
3.3	Knihovny pro práci s Discord API	20
3.3.1	Knihovna discord.py	20
3.4	Možnosti hostování aplikace	21
4	Návrh funkcionalit	21
4.1	Automatické vlastnosti	21
4.1.1	Událost: nový člen	22
4.1.1.1	Zaslání pravidel	23
4.1.1.2	Přiřazení role, pravomocí	23
4.1.1.3	Kontrola záznamů	23
4.1.2	Událost: zpráva	23
4.1.2.1	Kontrola odeslaných a upravených zpráv	24
4.2	Manuální vlastnosti	24
4.2.1	Správa uživatelů	25
4.2.2	Správa událostí	25
4.2.3	Přehrávání audio stop	25
5	Implementace	27
5.1	Souborová struktura	27
5.2	Spouštěcí soubor	28
5.3	Vytvoření aplikace a účtu jako bot	32
5.4	Ukládání dat	33
5.5	Struktura příkazů v kódu	34
5.6	Pravidla komunit	37

5.7	Kontrola zpráv	38
5.8	Kontrola uživatelů, přidělení pravomocí	39
5.9	Události a jejich spravování	40
5.10	Přehrávač	42
5.11	Hosting aplikace bota	43
Závěr		51
Conclusions		52
A Obsah přiloženého datového média		53
Seznam zkratk		54
Literatura		55

Seznam obrázků

1	Systémový bot platformy Discord	15
2	Příkaz vyvolaný prefixovanou zprávou	16
3	Kontextové menu lomítkových příkazů	16
4	Způsob komunikace bota a platformy [7]	17
5	Asynchronní programování s jedním procesem [12]	19
6	Obsloužení události při příchodu nového uživatele	22
7	Obsloužení události nové (upravené) zprávy	24
8	Souborový systém aplikace bota	27
9	Vytvoření aplikace	32
10	Vytvoření aplikace jako bot	32
11	Získání autorizačního tokenu bota	33
12	Identifikátor aplikace	33
13	Ukázka struktury příkazu v kódu, příkaz <i>clear</i>	36
14	Chybová funkce k příkazu <i>clear</i>	37
15	Ukázka vytvořené události příkazem <i>events echat</i>	48

Seznam tabulek

1	Limity pro nepředplacené uživatele a komunity	13
2	Úrovně limitů pro předplatitelské účty	13
3	Úrovně limitů pro předplacené servery	14
4	Knihovny pro práci s boty [11]	20

Seznam zdrojových kódů

1	Nastavení oprávnění bota v Discord API	28
2	Funkce určující prefix	29
3	Vytvoření bota	29
4	Ukázka callback funkce websocket eventu	29
5	Synchronizace příkazů	30
6	Instalace modulů	30
7	Spuštění bota	31
8	Otevření databázového souboru	34
9	Zápis do databázového souboru	34
10	Čtení v databázovém souboru	35
11	Aktualizace dat v databázovém souboru	35
12	Event (websocket) připojený člen	38
13	Zaslání pravidel komunity	39
14	Zobrazení pravidel	40
15	Přidání pravidla	41
16	Resetování pravidel	42

17	Smazání pravidla	43
18	Event (websocket) nové zprávy	44
19	Kontrola obsahu zprávy	44
20	Event (websocket) upravené zprávy	45
21	Zjištění existence záznamu uživatele	45
22	Oznámení o existenci záznamů nového uživatele	45
23	Část funkce pro vytvoření události	46
24	Funkce o oznámení události	47
25	Přehrání audia příkmem <i>play</i>	49
26	Zprocesování a přehrání audia	50

1 Úvod

Tato práce se zabývá vytvořením bota pomocí knihovny discord.py v programovacím jazyce Python pro komunikační platformu Discord, který bude sloužit jako nástroj pro komunity ke zlepšení správy nad danou komunitou, jejími pravidly, nastavením pravomocí pro nově připojené uživatele a jejich následnou kontrolou vůči prohřeškům z jiných komunit, kde bot mohl problémového uživatele již spatřit a případně tak závčas varovat moderátory. Bot umožní vytvoření události s názvem, popisem a datem konání a evidencí uživatelů přihlášených (odmítnutých, nerozhodných) na tuto událost a těsně před započítáním události je upozorní. Dále zprostředkuje možnost posílat audio stopu z YouTube přímo do hlasových kanálů na serveru. V práci je také rozebráno jsou možnosti, kde lze takovou aplikaci nasadit, vlastnosti každé z možností a implementace jedné z možností.

V teoretické části jsou popsány požadavky k pochopení a vypracování práce. Tedy způsob, kterým platforma Discord funguje po uživatelské a technické stránce, funkcí vlastních aplikací v kontextu s touto platformou a technologiemi používanými při jejich vývoji.

Praktická část se zabývá efektivním návrhem a následnou implementací bota.

Výsledkem práce je funkční aplikace realizující bota, který může být nasazen pro jakoukoliv komunitu na Discordu.

2 Platforma Discord

Discord je [VoIP](#) a sociální komunikační platforma. Její uživatelé zde mají možnost komunikovat skrze audiohovor, videohovor, média, soubory a zprávy a to buď prostřednictvím soukromých chatů a nebo jako součástí komunit (veřejné, soukromé), kterým se také někdy říká „servery“ nebo „guildy“. Discord je multiplatformní a běží na operačních systémech Windows, macOS, Linux, Android, iOS a ve webových prohlížečích. V roce 2021 bylo na Discordu registrováno 350 milionů uživatelů a evidováno 150 milionů aktivních uživatelů denně.

Platforma vznikla jako nápad dvou studentů pro odvětví herního průmyslu, kteří cítili nedostatky v komunikaci již existujících [VoIP](#) software se spoluhráči v taktických hrách jako je Final Fantasy XIV. [1] První release Discordu tak přišel začátkem roku 2015, dnes po letech vývoje se používá nejen v odvětví herního průmyslu ačkoliv ten stále dominuje. [2]

Discord je řazen mezi podobné platformy jako jsou např. Microsoft Teams, Microsoft Skype, Slack, Teamspeak, Reddit, Facebook Groups, které všechny patří ke komunikačním platformám s různými rozdíly podle uživatelů na které jsou cíleny.

2.1 Definice a terminologie

Discord používá svoji vlastní terminologii, kterou respektuje i tento text a je používán ve zbytku této práce. Následující seznam vysvětlí nejpodstatnější termíny:

Server

Server neboli komunitní server, někdy také označován anglicky slovem *guild* v kontextu kódu, je v uživatelském rozhraní platformy uskupení sdružující uživatele, hlasové a textové kanály.

Kanál

Kanál je místo, skrze které mohou uživatelé serveru komunikovat podle jeho typu. Existují pouze dva typy kanálů a to *hlasové*, nebo *textové*. Textové přenášejí zprávy, média a soubory, zatímco hlasové přenášejí navíc audio a video.

Kategorie

Kategorie je pojmenované uskupení kanálů, které jsou v uživatelském rozhraní platformy zobrazeny jako složky. Kategorie mohou obsahovat další kategorie a kanály.

Role

Role je pojmenovaná skupina uživatelů, které mohou být v uživatelském rozhraní platformy zobrazeny jako složky. Každá role s sebou nese svoji množinu systémových oprávnění a barvu k zobrazení. Role jsou postaveny v lineární hierarchii, role tak bývá nadřazena/podřazena jiné roli.

Emoji

Discord emoji je spojením *Unicode* emoji jako jsou smajlíci a personalizovanými čtvercovými obrázky definované v rámci konkrétního discord serveru.

Reakce

Reakce je discord emoji, které pomocí interakcí s tlačítkem u libovolné zprávy mohl uživatel nebo bot přidat.

Uživatel

Uživatel je osoba, která má vytvořený účet na platformě a může se tak připojit k libovolnému serveru. Uživatelé se dělí na dva druhy: *člověk* nebo *bot*. Když je řečeno slovo uživatel, je tím myšlen druh účtu, který není automatizovaný a stojí za ním skutečný člověk.

Bot

Bot je speciální druh uživatele, jehož účet je řízený programem, který komunikuje s *Discord API* více viz kapitola 4. Má přístup k více funkcím než obyčejný uživatel, které může používat k různým úkonům. Navíc bývá méně limitován než běžný uživatel co se do počtu zaslaných požadavků do API týče.

Příkaz

Příkaz je textová zpráva napsaná v uživatelském rozhraní v poli pro odesílání zpráv. Příkazy se dělí na dva typy, a to *prefixové* nebo *lomítkové*. Prefixované mají serverem určený speciální symbol, který rozlišuje normální zprávu od příkazové k vyhodnocení botem. Tato zpráva je odeslána do chatu jako běžná zpráva. Lomítkový příkaz se neodesílá do chatu jako prefixový příkaz, ale je odchycen v Discord API a předaný botu k obslužení.

2.2 Servery a komunity

Server na platformě je v uživatelském rozhraní jako kulatá ikonka v jeho levé části. Není zde myšleno, že server je výkonný počítač, který obsluhuje požadavky. Discord server, synonymem komunita či guilda, je poskytován a hostován přímo službou discordu jako takovou, která však běží na fyzických strojích společnosti, které obsluhují několik instancí discord komunit zároveň.

Na server se může uživatel dostat pouze pomocí pozvánky (vloženou v [UI](#)), která může být krátký textový řetězec připomínající hash sloužící jako identifikátor pozvánky. Popřípadě [URL](#) adresa, kde je identifikátor obsažen v parametru. Pozvánka může být implicitně parametrizována navíc o dobu expirace popřípadě limitem počtu užití. Neomezené pozvánky se zpravidla používají pro veřejné servery.

Při vytváření vlastního discord serveru lze v průvodci nastavením najít i šablony pro různé účely, které pomohou vytvořit a základně nastavit discord server podle specifických potřeb komunity.

Na serveru se mohou schopni najít kategorie, textové a hlasové kanály a samotní uživatelé serveru a případné integrace aplikací, které mají programem řízený účet odlišený za jménem pomocí štítku s nápisem „BOT“.

2.3 Kanály hlasové a textové

Kanály obecně mají společnou vlastnost, kterou je nastavení oprávnění podle rolí nebo „per user“.

Textové kanály mohou být dle oprávnění pro uživatele neviditelné, zamčené pro psaní, zakázané v jiných směrech např. posílání odkazů, přidávání reakcí, možnost smazat vlastní nebo cizí zprávu, nebo nastavené na určitý časový limit pro odesílání zpráv.

Hlasové kanály opět dle oprávnění mohou být neviditelné, zaheslované, nastavené maximálním limitem současně připojených uživatelů, nastavené kvality přenosu audia a videa.

Speciálním případem je komunikace uživatel s jiným uživatelem mimo server přes soukromý chat nebo hovor. Zde neplatí žádné oprávnění a uživatelé nejsou limitováni oprávněními, které lze aplikovat na role. Jedinou výjimkou jsou limity nastavené samotnou platformou, mezi které patří mazání zpráv druhého

uživatele, komunikovat s druhým uživatelem mimo seznam přátel. Komunikace mimo alespoň jeden server, který spolu sdílí, a uživatel druhý, příjemce, nevypl základní bezpečnostní funkci v prostředí svého účtu.

2.4 Uživatelé, role a pravomoce

Uživatel je kdokoliv kdo provedl registraci na platformě Discord a zavazuje se k dodržení podmínek služby.

Role je skupina oprávnění vztahující se pouze na serveru na němž byla vytvořena a může být přiřazena pouze uživatelům tohoto serveru.

K základní roli, kterou má každý server je role *everyone* s výchozím nastavením oprávnění, které se zpravidla ponechává a vytvářejí se role nové. Pro zpřísnění oprávnění než je role *everyone* bývá zvykem vytvořit novou roli, kterou některý z botů na serveru nastaví nově příchozím. Tato základní role však slouží v chatové zprávě pro hromadné označení všech členů serveru pomocí *@everyone*.

Existuje jedna role implicitní a tou je *server owner*, který se dá poznat, že v seznamu uživatelů serveru má uživatel za svým jménem ikonku královské koruny. Role není zobrazena v seznamu rolí serveru a není možné s ní nijak zacházet, má nejvyšší administrátorské oprávnění nade všemi.

Role na discord serverech jsou vytvářeny jako pořadový seznam v nastavení serveru, toto pořadí určuje jejich lineární hierarchii. Což znamená, že role nadřazená může dělat úkony na roli podřazené.

Pravomoce jsou elementární úkony, které mohou být nastaveny konkrétní roli. Mezi takové úkony patří: právo smazat zprávu, vytvořit pozvánku, vyhodit uživatele, upravit název kanálu, upravit nastavení serveru, vytvořit nové role. Všechny můžeme vidět v uživatelském rozhraní serveru pro správu rolí.

2.5 Limitace a předplatné

Myšleny jsou limity, které jsou kladeny na uživatele a komunitní servery na platformě s možností jejich výběru na základě předplatného, limitem aplikací pak rozumíme omezení, které jsou kladeny na účet bota a jeho přístup k informacím platformy.

Některé limity je schopen si uživatel nebo komunitní server navýšit díky předplatitelům. Pro uživatele se druh předplatného nazývá *Nitro Basic* a vylepšená varianta *Nitro*. Předplacený server se nazývá dle její úrovně, které jsou *Tier 1*, *Tier 2* a *Tier 3*.

Limity bez předplacení, podle tabulky 1, platné pro všechny uživatele a komunitní servery.

Limity s předplatným, se v případě předplatného pro uživatele řadí do dvou úrovní, pro komunitu se předplatné řadí do úrovní tří. Tabulka 2 ukazuje úrovně předplatného pro uživatele, tabulka 3 ukazuje předplatné úrovně pro servery.

K limitům patří také „rate limit“, což je maximální počet požadavků, které uživatelský účet nebo účet bota může platformě zasílat, bot má tento limit vyšší. Požadavek je např. odeslání zprávy, vytvoření pozvánky serveru, probíhající audio nebo videohovor. Při rychlém vyčerpání přiděleného maximálního počtu požadavků pro určitý časový úsek požadavky nemají žádný efekt dokud neuplyne určitý čas po kterém se předešlé požadavky automaticky pokusí sami zopakovat. Nadměrným a úmyslným čerpáním tohoto maximálního počtu požadavků je proti ToS a může dojít k pozastavení nebo odstranění účtu jedince nebo komunitního serveru.

Typ	Limit
Počet uživatelů na serveru	250,000
Počet online uživatelů na serveru	5,000
Počet serverů uživatel smí být členem	100
Počet kategorií na serveru	50
Počet kanálů na serveru	500
Počet rolí na serveru	250
Počet neanimovaných emoji na serveru	50
Počet anomovaných emoji na serveru	50

Tabulka 1: Limity pro nepředplacené uživatele a komunity

Typ	Úroveň 1	Úroveň 2
Vlastní emoji kdekoliv	Ano	Ano
Vlastní samolepky kdekoliv	Ano	Ano
Velikost souborů	50MB	500MB
Vysoké rozlišení videa	Ne	Ano, až 4k@60
Personalizovaný profil účtu	Ne	Ano
Personalizovaný profil „per server“	Ne	Ano
Odznak předplatitele	Ano	Ano
Vlastní pozadí ve videohovoru	Ano	Ano
Počet serverů uživatel smí být členem	100	200
Delší zprávy	do 2000 znaků	do 4000 znaků

Tabulka 2: Úrovně limitů pro předplatitelské účty

Typ	Úroveň 1	Úroveň 2	Úroveň 3
Vlastních emoji serveru	až 100	až 150	až 250
Audio kvalita	až 128Kbps	až 256Kbps	až 384Kbps
Vlastní pozadí v pozvánce	Ano	Ano	Ano
Vlastních samolepek serveru	až 15	až 30	až 60
Animovaná ikona serveru	Ano	Ano	Ano
Kvalita 1080p@30 videa všem	Ano	Ano	Ano
Vlastní ikony rolí	Ne	Ano	Ano
Vlastní banner serveru	Ne	Ano	Ano
Kvalita 1080@60 videa všem	Ne	Ano	Ano
Až 50MB velikost souborů všem	Ne	Ano	Ano
Až 100MB velikost souborů všem	Ne	Ne	Ano
Animovaný vlastní banner serveru	Ne	Ne	Ano
Vlastní odkaz pozvánky	Ne	Ne	Ano

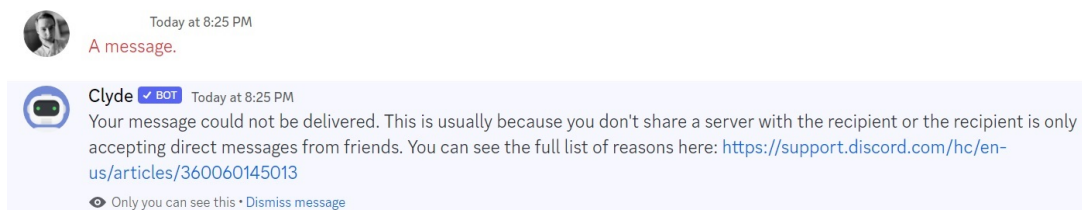
Tabulka 3: Úrovně limitů pro předplacené servery

3 Discord Bot

Discord bot je uživatelský účet, který je kompletně automatizovaný programem, tento program pak komunikuje prostřednictvím Discord API [5] s platformou na které provádí automatické úkony nebo úkony vyvolané uživatelem či prostředím za účelem vylepšení uživatelské zkušenosti. Platforma komunikuje s botem skrze WebSocket API [6] a předává tak v „realtime“ informace bez nutnosti navazovat neustále nová spojení, bot si tyto odpovědi ukládá do mezipaměti, jedná se tak o rychlý přenos informace mezi platformou a botem. Druhým směrem bot odpovídá platformě skrze její REST API [7], kde místo odesílání celých objektů odesílá pouze identifikátory objektů, které má ve své mezipaměti.

Bota si může zřídit jakýkoliv registrovaný uživatel discordu, což může učinit na oficiálních webových stránkách platformy, kde svoji aplikaci pojmenuje a vytvoří. Vytvořením dostane autorizační token, který poté následně použije ve zdrojovém kódu podle patřičné implementace Discord API knihovny pro zvolený programovací jazyk.

Clyde bot je bot jenž nepatří žádnému registrovanému uživateli, ale jedná se o systémového bota platformy, který je zde pro vylepšení a používání této platformy. Můžeme ho spatřit na obr. 1, když se pokusíme poslat zprávu uživateli, který nás zablokoval.



Obrázek 1: Systémový bot platformy Discord

3.1 Ovládání a práce s botem

Ovládání bota je buď automatické, nebo manuální. Automatické fungují tak, že přes websocket dostává bot informaci o eventech, které se na platformě stali, třeba uživatel odeslal zprávu a botu přijde event typu **on-message** pro který ve zdrojovém kódu bude vytvořena funkce, která funguje jako *callback* a bude spuštěna při tomto eventu z websocketu. Tyto eventy jsou předdefinované a je na programátorovi, které funkce jako callbacky vytvoří podle toho, kterým eventům chce jeho aplikace naslouchat. Manuální jsou funkce, které definují příkaz prostřednictvím uživatelského rozhraní v chatovacím okně dvěma způsoby: prefixovanou zprávou, nebo příkaz za lomítkem.

Pokud bot obsahuje funkce, které jsou v příslušném jazyce knihovny Discord API označeny jako event a dodržují definované pojmenování, pak budou automaticky skrze knihovnu spuštěny jako callback.

Uživatel bota nemusí v případě úkonů, které reagují na websocketem zasláné eventy, nic dělat. Pro úkony, které vyžadují zásah nebo doplnění informací k jejich provedení, musí uživatel tyto úkony vyvolat manuálně sám přes příkazy prefixové, nebo lomítkové.

3.1.1 Prefixované zprávy

Je způsob komunikace mezi uživatelem a botem, kdy uživatel napíše zprávu do chatu jako obvykle s rozdílem, že před začátek zprávy dá speciální symbol, který bot poté rozpozná, která zpráva obsahuje příkaz a která zpráva byla mezi probíhající konverzací na komunitním serveru. Mezi speciální symboly, které se často používají jsou: `?`, `!`, `&`, `*`. Tyto symboly jsou vždy na začátku zprávy a následuje za nimi řetězec znaků, který je definován jako příkaz. Příkaz je pak rozdělen na jeden či více argumentů pomocí mezer, před první mezerou je tak symbolické jméno příkazu.

Na obr. 2 můžeme vidět příkaz *echo* s jedním argumentem následující za první mezerou po slově *echo*. Při implementaci funkce příkazu je vždy možnost pro poslední argument příkazu posbírat celý řetězec včetně mezer a chápat jej jako jeden argument jako tomu je v tomto případě.

Pokud máme příkaz s více argumenty, kdy do jednoho argumentu potřebujeme dát řetězec obsahující i mezery je potřeba tento řetězec obalit do uvozovek.

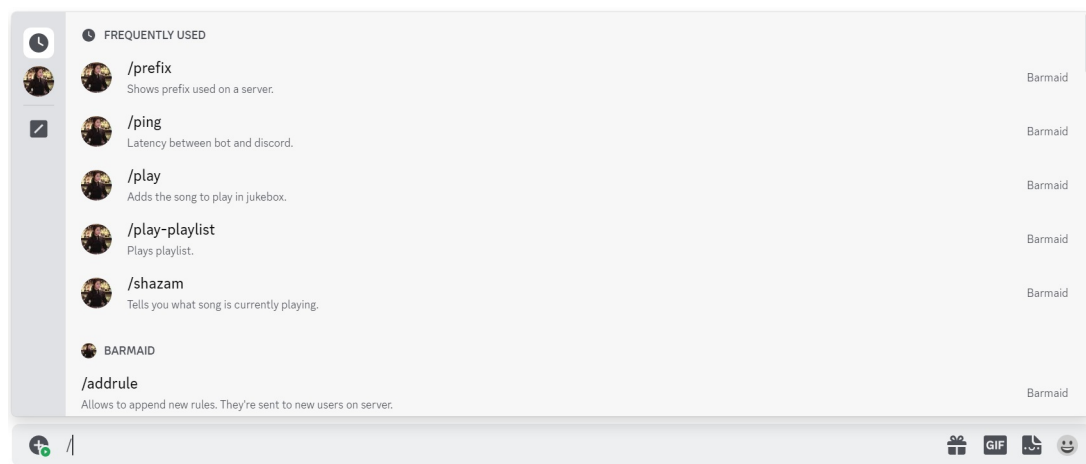


Obrázek 2: Příkaz vyvolaný prefixovanou zprávou

3.1.2 Lomítkové příkazy

Je novějším způsobem komunikace uživatel a bot, kdy uživatel předepíše do textového pole pro odeslání nové zprávy lomítko.

V uživatelském rozhraní se rozbalí menu s možnostmi příkazů, které jsou definovány v kódu bota. Uživatel vybere příkaz a je mu našeptáváno jaké argumenty musí použít a co do nich patří pro úspěšné vykonání příkazu. Poprvé byly lomítkové příkazy přidány do uživatelského prostředí a do odpovídajících API koncem roku 2020, v roce 2022 se stali jako povinnosti pro boty, kteří musejí být verifikováni za účelem nakládání s citlivými daty uživatelů platformy. Dokud bot nepřesáhne jeho počet 100 komunitních serverů ke kterým se připojil není potřeba žádného ověření.



Obrázek 3: Kontextové menu lomítkových příkazů

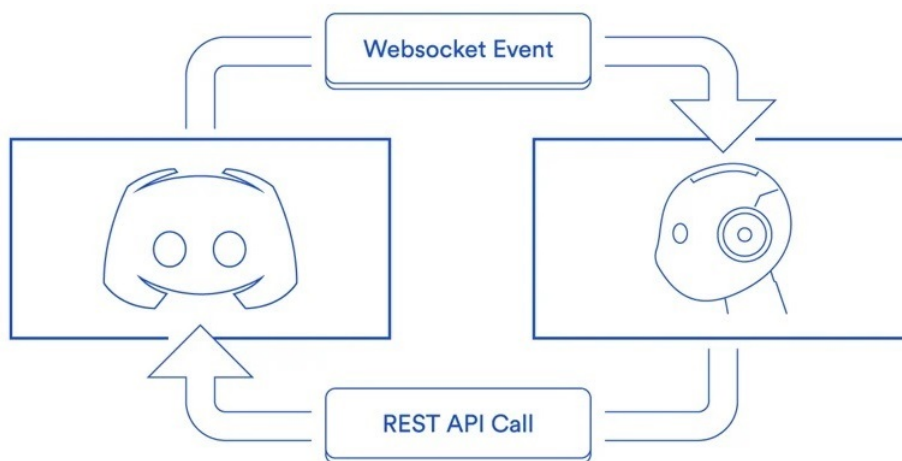
Na obr. 3 vpravo vidíme sloupec ve kterém nalezneme všechny boty na daném komunitním serveru, uprostřed nabídku se všemi příkazy kde shora jako první se zobrazují nejčastěji používané příkazy a na konci řádku příkazu vidíme název bota, kterému příkaz patří. Více botů může mít stejně pojmenovaný příkaz a uživatel je schopen je rozlišit podle profilové fotky bota, nebo jeho jména.

3.2 Technologie pro vývoj

Pro komunikaci s Discordem má platforma své vlastní [API](#), k počátku roku 2023 se datuje už 10. verze. Přičemž se dá používat ještě verze předešlá, verze 6-8 jsou nyní zastaralé a verze 1-5 jsou již zrušené [5]. Komunikovat s touto API napřímo je náchylné na chyby programátora při implementaci, které mohou způsobit nechtěné přetěžování API, které může vyústit k zablokování této komunikace platformou nebo dokonce účtu bota či samotného uživatele vlastního tohoto bota. Od těchto strastí s verzí a správnou komunikací nás abstrahují pryč knihovny pro práci s API platformy Discord. Knihovny jsou napsány jako „wrapper“ v patřičném programovacím jazyce usnadňující práci s API a ošetřují jeho nebezpečného zacházení. Další knihovny pro vybraný programovací jazyk mohou pomoci při komunikaci bota s webovými stránkami na internetu, databází a dalšími službami pro specifickou činnost bota.

3.2.1 Discord API

Je preferovaný způsob komunikace mezi botem a platformou Discord, umožňuje snadné, rychlé a bezpečné vytvoření bota pro komunikaci s platformou přes dva základní prvky: **WebSocket API** a **REST API**. Na obr. 4 převzatého z článku [7], je vidět průběh komunikace kdy platforma vlevo komunikuje ve směru bota pomocí WebSocket API, a bot vpravo ve směru platformy pomocí volání REST API.



Obrázek 4: Způsob komunikace bota a platformy [7]

3.2.1.1 WebSocket API

WebSocket API je používán při přijímání eventů, o změně stavu na platformě, ze serverových strojů platformy Discord. Komunikace je navázána mezi platformou a botem skrze websocket, který umožňuje vytvořit přes informace v „realtime“,

eliminuje se tím potřeba pro každý přenos informace vytvořit nové spojení a vzniká tak velmi rychlý způsob přenosu informací.

Mezi eventy, které se přenášejí z platformy botu jsou například: uživatel zaslal zprávu, uživatel upravil zprávu, uživatel smazal zprávu, uživatel se připojil ke komunitě, uživatel připl zprávu, uživatel vytvořil vlákno ve zprávě, uživatel se připojil k hovoru, uživatel si ztlumil mikrofon v hovoru.

Během přenosu informace o jakém eventu se stal je spolu s ním předávány objekty *zpráv*, *uživatelů*, *místností*, *komunitního serveru* obsahující úplné informace. Po úspěšném přenosu informace o tom, že nastal event si bot ukládá všechny informace o objektech do **cache** se kterými je poté schopen zpracovat pro úkony co bot nad nimi provede a odesílá odpovědi s jeho reakcí do REST API.

3.2.1.2 REST API

Akce, kterých je bot schopen provést musí provést přes volání REST API, bot je také schopen si skrze REST API vyžádat získání informací, ale to není zpravidla kvůli výkonu používáno, neboť většinu těchto informací již dostal z websocketu a má je uložené ve své cache.

Bot z této cache vytáhne objekty se kterými potřebuje pracovat a při provedení změn nad nimi odesílá do REST API pouze **identifikátor** objektu a změnu, kterou provedl a neodesílá tak znovu celý objekt z důvodu rychlosti a případnému zahlcení API.

Identifikátor objektu je velké celé přirozené číslo, pod kterým si platforma je schopna najít objekt, kterému patří. V případě bota si i bot je schopen pomocí identifikátoru dohledat objekt kterému patří avšak s omezením, že objekt se musí nacházet v jeho cache. Nemůže tak například získat objekt komunitního serveru jenž bot není členem i kdyby znal jeho identifikátor, protože takový objekt mu websocket s tímto identifikátorem nikdy nezaslal.

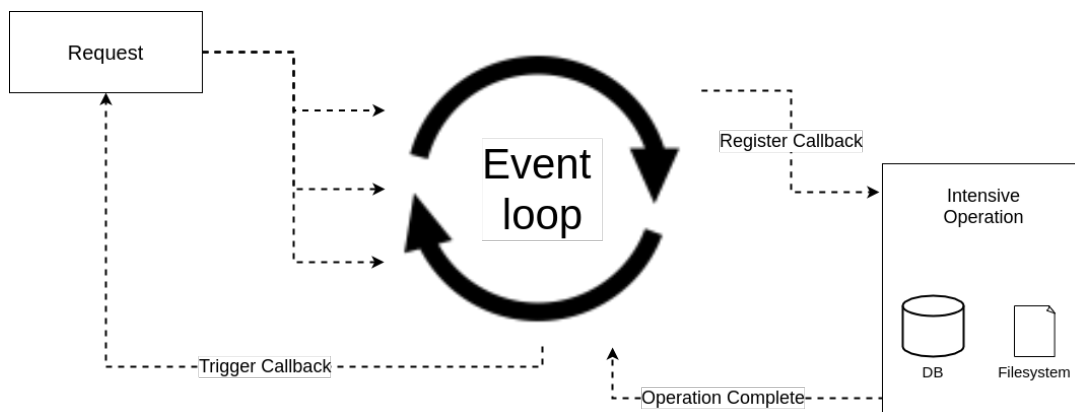
3.2.2 Asyncio

Asynchronní programování používané při komunikaci klient-server trvá nějaký čas než se informace přenesou a dostaví se odpovědi, pro využití procesorového času efektivně namísto čekání se vykonává něco jiného zatímco se čeká než odpověď dorazí. Asyncio, přesněji asyncio je zkratka za asynchronous input/output, technologie která se stará o efektivní hospodaření s procesorovým časem při operacích, jenž mají delší efekt jako právě komunikace po síti.

Tato technologie dává možnost tzv. „single threaded“ procesům zdánlivě vykonávat více operací současně, jako jsou například samotné webové prohlížeče s *JavaScriptem*. Dlouho trvající operace nejprve všechny naráz jednu po druhé spustí a po vyčkání vrací v žádaném pořadí odpovědi po provedení operace, než aby se sekvenčně spustila jedna operace a vyčkalo se na dokončení než se spustí v sekvenci další dlouhotrvající operace.

Program má pouze jedno vlákno označované též jako **main thread** a tedy jeden **execution stack**, kde může vykonávat pouze jednu operaci. Při asynchron-

ním programování požadavek jako je získání dat z webové stránky by blokoval execution stack po celou dobu čekání odpovědi namísto toho předává tento svůj požadavek do **event loopu** a z execution stacku se tím funkce odstraní, ukládá se místo kam se po callbacku navrátí, a přenechá to na event loopu. Event loop se postará o obsluhu tohoto požadavku a spustí jej, po dokončení požadavku se skrze callback vrací odpovědi požadavku zpátky do execution stacku main threadu.



Obrázek 5: Asynchronní programování s jedním procesem [12]

Na obr. 5 vlevo vidíme request z main threadu programu předaný event loop smyčce starající se o časově náročné input/output operace vpravo, po jejich dokončení se vrací zpět přes callback do main threadu programu.

3.2.3 FFmpeg

Je open-source software **CLI** nástroj pro práci s audiem a videem a dalšími multimediálními soubory a datovými proudy. [13] [14] **Fast Forward moving picture experts group**, zkráceně FFmpeg byl vytvořen již v roce 2000 a dnes je používán ve spoustě aplikací jako je *Google Chrome*, *blender* a platforme jako *YouTube*, *Discord* a *Vimeo*. Nástroj je schopen dekódovat, enkódovat, transkódovat, multiplexovat, demultiplexovat, streamovat, filtrovat a přehrávat jakýkoliv multimediální soubor na světě. Podporuje přes 100 video kodeků.

FFmpeg funguje tak, že vezme jakýkoliv multimediální soubor, který pomocí demultiplexeru rozdělí do několika audio a video stop jako separátní data pakety. Pakety jsou dále dekódovány do jednotlivých snímků, které poté mohou být zpracovány či filtrovány. Zpracováním jako například úpravou jasu, kontrastu, přidáním titulků. Poté tyto snímky jsou zpátky zakódovány a multiplexerem složeny nazpět do cíleného formátu.

Součástí nástroje je také nástroj *ffplay* pro přehrávání multimediálního souboru a *ffprobe* pro získání metadat ze souboru.

3.3 Knihovny pro práci s Discord API

K funkcionalitám platformy Discord sice jde přistupovat napřímo pomocí API v různých verzích a použitím [HTTP](#) protokolu a zasíláním *requestů*, ale knihovny mohou tuto činnost podstatně zjednodušit a usnadnit a použít ji je výhodnější. Každá knihovna zapouzdřuje všechna volání do Discord API a stará se o limity a chybové stavy.

Limit o který se stará je ratelimit a chrání uživatele knihovny před jeho překročením, pokud kód provádí spoustu požadavků rychle tak předchází zahlcením API pozdržejím požadavků překračujících určitý limit a odesílá je postupně v pořadí co nejdříve je to možné.

Zamezuje odeslání informací, které by vedli k chybovým stavům API a tyto špatné informace propaguje knihovna svými chybovými stavy v příslušném programovacím jazyce.

Discord sám některé z knihoven třetích stran [\[11\]](#) schválil a označil, že splňují požadavky pro jejich API, výrazně také doporučil použití takových knihoven pro komunikaci s platformou než přístupem napřímo. Nejčastěji používané knihovny vidíme v tabulce [4](#) níže:

Název knihovny	Programovací jazyk
Discord.Net	C#
discord.js	JavaScript
discord.py	Python
Discordia	Lua

Tabulka 4: Knihovny pro práci s boty [\[11\]](#)

3.3.1 Knihovna discord.py

Jedná se o knihovnu pro Discord API v programovacím jazyce Python, vytvořenou jako open-source projekt se zdrojovým kódem na platformě Github. [\[15\]](#) Team, který knihovnu vytvořil ji udržuje aktuální, funkční a snaží se pokrýt vše, co nabízí celé Discord API a vytvořit tak jednoduchý, rychlý a bezpečný nástroj při vývoji a implementaci botů. Uživatele knihovny zcela abstrahuje od volání do zmíněného API, který v kódu pracuje pouze objektově orientovaným přístupem. Tento způsob dělá kód jednodušší, čitelnější a přenositelnějším.

Nově přidané funkce platformou Discord a Discord API bývají implementované do knihovny v řádu dnů, neboť se na jejím vývoji podílí přes 300 kontributorů.

Knihovna je ve dvou verzích, jedna bez podpory audia, druhá s podporou audia. Při její instalaci je tak nutné specifikovat, kterou nainstalovat, bez explicitního uvedení o audio verzi je nainstalována základní obsahující eventy (zasílané websocketem) a podpora pro vytváření příkazů.

3.4 Možnosti hostování aplikace

Spuštění může být realizováno na jakémkoliv hardware běžící s libovolným operačním systémem a nainstalovanými potřebnými programy na osobním počítači, osobním mikropočítači, nebo ve službě cloudu na pronajatém vzdáleném počítači.

Použití osobního počítače je způsob při vývoji a testování aplikace bota a to z několika důvodů. Je vhodné a rychlé vyzkoušet si všechny implementované změny ihned bez nutnosti přenosu souborů na jiný hardware. Avšak bývá nežádoucí ponechávat neustále zapnutý a připojený k síti osobní počítač, proto se na ostrý provoz vybírá plnohodnotný hosting.

Použitím osobního mikropočítače jako je Raspberry Pi [9] je plnohodnotná varianta k hostování aplikace bota. Raspberry Pi navíc dává velikou flexibilitu v operačních systémech, hardwarových specifikacích osobního mikropočítače a jeho provozování je levné.

Cloudová služba AWS Cloud Computing [10] dává možnost se u jejich služby zaregistrovat a pronajmout si vzdálený stroj na kterém bot následně poběží. Nabídka vzdálených strojů a jejich hardwarovou konfigurací je velká, pro nenáročné aplikace postačující základní výpočetní výkon jsou tyto stroje i bezplatné.

4 Návrh funkcionalit

Tato kapitola se zabývá návrhem funkcionalit automaticky prováděných botem, případně funkcionalit, které manuálně vyvolá uživatel, ještě před jejich implementací v kódu je detailně probrán popis činnosti těchto funkcionalit.

K automaticky prováděným úkonům, což jsou úkony, které reagují na event z websocketu, neboli změnu stavu platformy, zaslané skrze websocket. K těmto úkonům patří při připojení nového uživatele kontrola jeho záznamů, zaslání pravidel komunitního serveru a přiřazení komunitou předem vybrané role s pravomocemi.

K manuálním úkonům slouží příkazy, které uživatel může vyvolat pomocí textového pole pro odeslání zprávy do chatu. K těmto příkazům se řadí správa uživatelů – vyhoštění a zablokování s přidáním záznamu o uživateli do databáze, přesouvání skupiny uživatelů mezi hlasovými kanály. Správa událostí – vytvoření události s názvem, popisem, datem konání a evidencí přihlášených osob, kterým se těsně před počátkem události zašle upozornění. Přehrávání audio stop – přehrání jednoho multimediálního obsahu, přehrání více multimediálního obsahu dané playlistem, úpravou hlasitosti, zobrazení fronty, přeskočení aktuálně přehrávaného audia.

4.1 Automatické vlastnosti

Jedná se o takové vlastnosti bota, které jsou prováděny automaticky bez zásahu uživatele, uživatel může pouze nadefinovat jejich chování pro daný komunitní server. K takovým vlastnostem se řadí informace o změně stavu na platformě

jako je připojení uživatele *on_member_join*, připojení klienta ke komunitnímu serveru *on_guild_join*, při zaregistrování nové zprávy *on_message*, při zaregistrování upravené zprávy *on_message_edit*, při zaregistrování úspěšné inicializace klienta do platformy *on_ready*.

Pro typ *on_member_join* se provede získání pravidel komunitního serveru a zaslání nově připojenému uživateli, získání přidělené role s pravomocemi a následné přidělení uživateli, zjištění existence záznamů o trestech uživatele a následném upozornění moderátorů komunitního serveru.

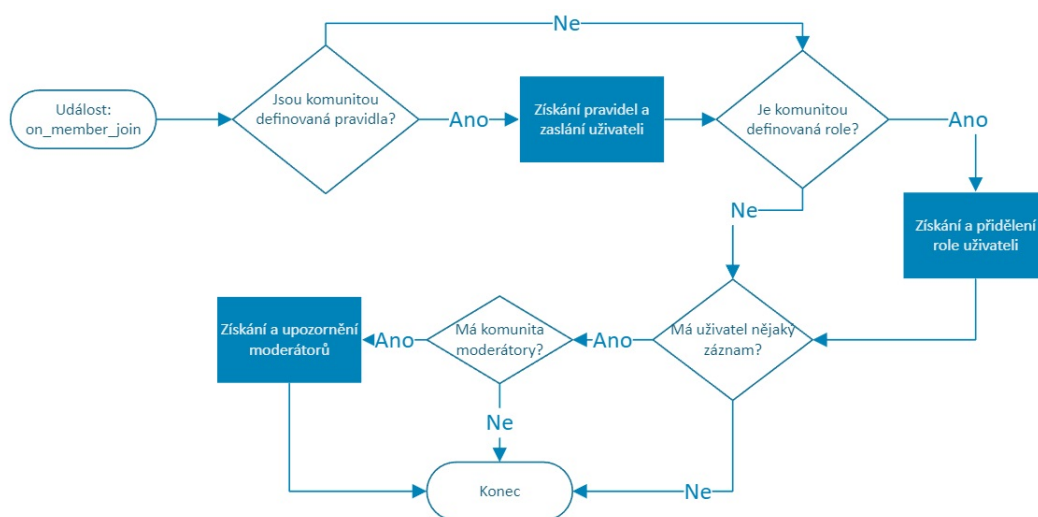
Pro typ *on_message* a *on_message_edit* provede získání slovníku zakázaných slov a frází, zjištění, zda zpráva obsahuje některé z těchto slov a frází, pokud ano je zpráva okamžitě odstraněna.

Typy *on_guild_join* a *on_ready* slouží k uvedení bota do provozu pro daný komunitní server, přidává záznam o serveru do databáze. Tuto automatickou vlastnost nemůže uživatel komunity nijak definovat podle svých vlastních potřeb.

4.1.1 Událost: nový člen

Při získání eventu z websocketu týkající se připojení nového člena obsluhujeme všechny tři vlastnosti zároveň: zaslat pravidla, přiřadit role a pravomoce, zkontrolovat záznamy.

Reakce o změně tohoto stavu je rozdělena na každou činnost zvlášť a každá může být vykonána individuálně, je-li v komunitním serveru zadefinována. Člen s příslušným moderátorským oprávněním si musí v komunitním serveru definovat své pravidla, které chce, aby bot odesílal. Definovat roli, chce-li, aby ji bot přiřadil a jmenovat členy, ke kterým se bot bude chovat jako moderátory komunity a upozorní je na uživatele se záznamy.



Obrázek 6: Obsloužení události při příchodu nového uživatele

4.1.1.1 Zaslání pravidel

Pro automatické zasílání pravidel se alespoň jedno pravidlo musí definovat na komunitním serveru pomocí přidávacího příkazu *rules add*. Příkazem *rules reset* se smažou existující pravidla komunity. Pro zobrazení všech platných pravidel poslouží příkaz *rules*. Pravidla se budou automaticky číslovat, v pořadí ve kterém byly přidány a budou novému uživateli zaslány při připojení do soukromé zprávy.

4.1.1.2 Přiřazení role, pravomocí

K přiřazování role novým členům bude potřeba, aby již na komunitním serveru přes uživatelské rozhraní byla role nadefinována. Poté přes příkaz jako *autorole set* bude moci moderátor přiřazovanou roli s pravomocemi nastavit. Příkazy *autorole show*, *autorole remove* ji zobrazí nebo smažou. Takto definovaná role bude automaticky přiřazena novému uživateli po jeho připojení na komunitní server, za podmínky že je v hierarchii pod rolí bota.

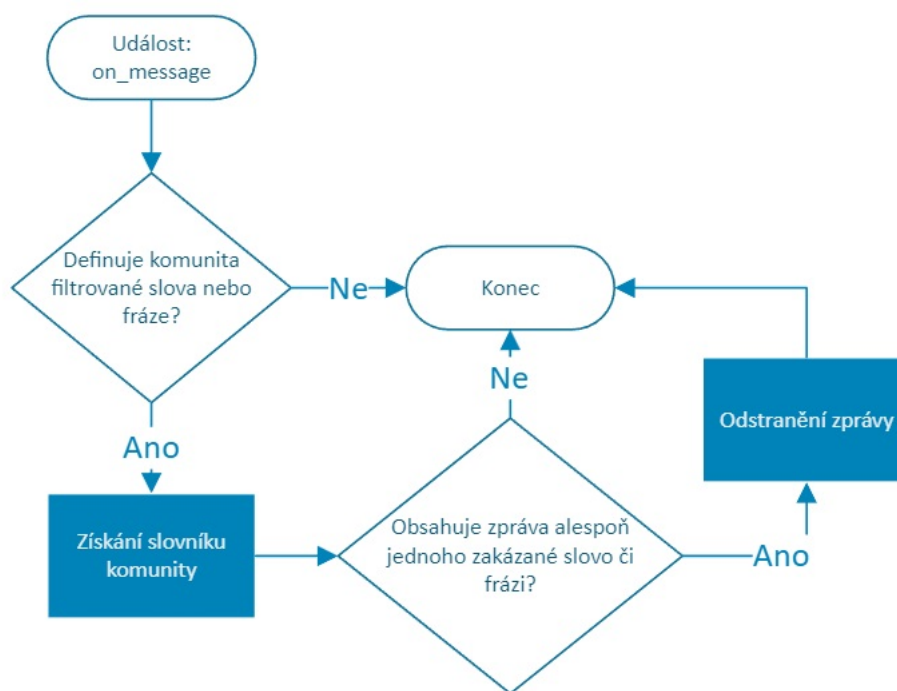
4.1.1.3 Kontrola záznamů

Kontrola záznamu se provede automaticky poté co uživatel, který má záznam v rejstříku se připojí na server. Upozornění dostanou pouze předem jmenovaní moderátoři, které lze přidat, odstranit a zobrazit příkazy *moderation add*, *moderation reset*, *moderation show*. Bude také možné zkontrolovat jestli již stávající člen nemá nový záznam pomocí příkazu *records show*. Záznamy do rejstříku budou dělat příkazy pro vyhoštění či zablokování uživatele v libovolné komunitě (i mimo stávající), kterými budou *kick* a *ban*.

4.1.2 Událost: zpráva

Zaslaný event o změně stavu, že byla zaslána či upravena zpráva, bot bude muset zkontrolovat obsah zprávy a v případě závadné zprávy takovou zprávu nechat odstranit.

Následná reakce na změnu tohoto stavu obsahuje jednu činnost, zkontrolovat slovník slov a frází v příslušném komunitním serveru nově příchozí (upravené) zprávy, její zpracování.



Obrázek 7: Obsloužení události nové (upravené) zprávy

Na obr. 7 je vidět událost při zjištění stavu o zaslání nové zprávy, kdy se získá „blacklist“ slovník komunity a pokud ve zprávě existuje alespoň jeden výskyt ze slovníku, je taková zpráva odstraněna. Pro událost při upravené zprávě by vypadal diagram stejně jako na obr. 7, s jediným rozdílem a to spouštěcí událostí.

4.1.2.1 Kontrola odeslaných a upravených zpráv

Pro automatickou moderaci chatu botem je potřeba přes příkaz *filter add* definovat alespoň jedno zakázané slovo (frázi) v komunitním serveru. Příkazem *filter remove* se bude moci odstranit konkrétní napsané slovo či fráze, *filter show* zase zobrazí veškerý slovník. Není žádoucí, aby uživatel bez oprávnění mohl příkaz pro zobrazení spustit a nalézt případné chyby k obcházení. Též se nebude rozlišovat zda slovo či fráze má malá nebo velká písmena.

4.2 Manuální vlastnosti

U manuálních vlastností bota se jedná o takové akce, které pro jejich vykonání musejí být vykonány uživatelem, tedy uživatel musí spustit příkaz přes kontextové menu obr. 3 lomítkového příkazu nebo napsáním prefixované zprávy obsahující validní název příkazu obr. 2. Vlastnosti, které tyto manuální příkazy mají zastat jsou pro správu uživatelů (vyhoštění, zablokování), správou událostí (vytvoření, smazání, editace), přehráváním audio stop (přehrání, přeskočení, zobrazení fronty, úprava hlasitosti).

Tyto příkazy musejí být manuální neboť potřebují extra vstup od uživatele k doplnění či upřesnění některých informací potřebných k jejich provedení.

4.2.1 Správa uživatelů

Pro správu uživatelů bude zavedena možnost vyhostit jednoho uživatele či více zároveň pomocí příkazu *kick* s nepovinným odůvodněním, jejich vyhoštění povede k vytvoření záznamu v databázi bota pro případné varování uživatelů v jiných komunitách o jejich incidentu. Podobně na tom bude příkaz zablokování *ban* s rozdílem, že umožní ještě volbu smazání jejich výskytu v historii chatu za posledních několik předchozích dní. Zprostředkuje pro majitele komunity s nejvyšším oprávněním možnost zasílat hromadně všem uživatelům jeho komunitního serveru zprávu do soukromé zprávy příkazem *admin message* (či *embedded*), bude se dbát na potencionální zneužití při napadení účtu majitele a bude nastaven limit pro tuto funkcionalitu, aby nemohla být v případě velkých komunit zneužita např. pro scam či phishing. Pro správu nad hlasovými kanály bude moderátor s oprávněním mít možnost příkazem *move* vybrat „mentions“ připojených uživatelů a označením cílového hlasového kanálu do kterého budou následně přesunuti.

4.2.2 Správa událostí

Zprostředkování a správa naplánovaných událostí je uděláno pomocí zapouzdřené zprávy a interakcí příslušnými tlačítky (*Accept*, *Decline*, *Tentative*, *Cancel*). Tlačítko *accept* půjde použít dokud se nepřekročí nastavený limit přihlášených, je-li stanoven. *Decline* odmítne událost, *tentative* pro nerozhodného uživatele a tlačítko *cancel* použitelné pouze pro uživatele jenž událost vytvořil pro její zrušení. Zapouzdřená zpráva umožní formátovat jednotlivé položky zprávy do řad a sloupců s podporou odkazů a obrázků. Vytvoření události se provede příkazem *event echat*, který stojí za zkratkou *event external echat*, při jeho vytváření bude uživatel vyzván k doplnění názvu, popisku, datu, volitelnému hornímu limitu počtu přihlášených a výběru zda bude použito číslo počtu přihlášených nebo jejich celé jména (vhodné pro menší událost) a místa buď libovolného textu či hlasového kanálu. Vedle toho existuje příkaz *events ilocation* (*ivoice*) pro tvorbu události zaintegrované přímo v Discordu v komunitě. Tyto integrované události jdou upravovat pomocí *events edit_location* (*events edit_voice*) a smazat *events idelete* za pomoci vyhledání první shody názvu události se nalezená událost odstraní. Pro odstranění *events echat* není nutné mít vlastní příkaz, neboť stačí zprávu moderátorem odstranit.

4.2.3 Přehrávání audio stop

Přehrání audio stop je uděláno příkazem *play*, který bota připojí do stejného hlasového kanálu jako uživatel jenž příkaz vyvolal a začne přehrávat hudbu danou

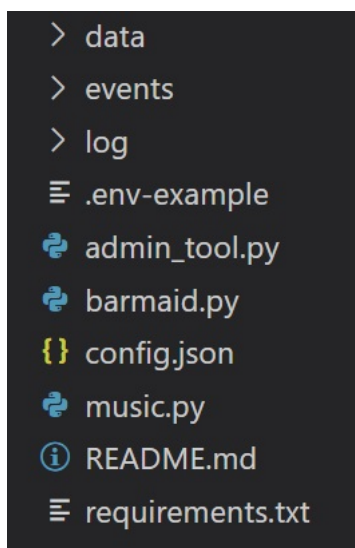
odkazem, v odkazu může být pouze video z platformy YouTube. Při opakovaném použití příkazu *play* se bude další hudba zařazovat do fronty, která se dá zobrazit příkazem *queue*. Pro přehrání playlistu slouží příkaz *play_playlist*, který zařadí každou přístupnou písničku z playlistu do fronty. Příkaz *volume* umožní upravit hlasitost přehrávaného audia a upraví hlasitost i pro veškerý další přehrávaný obsah. Příkaz *pause* pozastaví přehrávání hudby, příkaz *resume* opět spustí přehrávání pozastavené hudby, *stop* zastaví hudbu a odpojí bota od hlasového kanálu. Příkazy jako *next* přeskočí aktuálně přehrávající audio a začne přehrávat další v pořadí, *shazam* určí název aktuálně přehrávané skladby.

5 Implementace

Tato kapitola se zabývá tím, jakým způsobem je bot vyvinut a implementován.

Při implementaci bota byla zvolena knihovna discord.py, která zajistí komunikaci bota s Discord API, knihovna yt_dlp která umožní komunikovat s platformou YouTube a knihovna pro manipulaci s audiem FFmpeg a k tomu knihovna asyncio umožňující asynchronně kód vykonávat. Jako další knihovny jsou použity aiohttp pro potřeby komunikace s REST API, aiofiles pro asynchronní přístup k souborům a knihovna python-dotenv sloužící jen pro uložení autorizačního tokenu mimo zdrojový kód za účelem bezpečnosti. Je také implementována vlastní databáze za pomoci JSON formátu, neboť není nutné ukládat velké množství dat a vedle aplikace bota mít databázový systém by bylo paměťově větší než samotná aplikace bota.

5.1 Souborová struktura



Obrázek 8: Souborový systém aplikace bota

Na obr. 8 je souborová struktura rozdělena následujícím způsobem:

data

Složka obsahuje veškerou práci s daty se kterými bot nakládá, jako je načtení informací z konfiguračního souboru, hlavní databáze bota a databáze záznamů uživatelů, modul pro správu nad zmíněnými databázemi.

events

Složka obsahuje moduly související s plánovanými událostmi.

log

Složka obsahuje modul pro vytvoření a obsluhu logů jenž bot využívá a soubor do kterého se log zapisuje.

.env-example

V souboru .env-example se nachází placeholder pro autorizační token bota, který je potřeba doplnit při zprovoznění vlastního účtu bota. Soubor je poté nutné přejmenovat jako .env a ponechat.

admin_tool.py

Je modul obsahující všechny příkazy týkající se správy komunity, uživatelů.

barmaid.py

Je hlavní spouštěcí soubor bota, který připojí ostatní moduly a bota přihlásí do Discordu.

config.json

Je základní konfigurační soubor ve kterém se v případě vlastního bota musí vhodně nastavit údaje s identifikátorem bota, dále zde lze upravovat chování bota k příkazům.

music.py

Je modul obsahující všechny příkazy týkající se přehrávání audia.

README.md

Je soubor obsahující popis projektu a instalaci, určený pro Github repozitář.

requirements.txt

Je soubor obsahující popis všech potřebných knihoven v příslušných verzích pro instalaci skrze *Python Installer Package*, zkráceně *pip*.

5.2 Spouštěcí soubor

Jako hlavní a také jako jediný spouštěcí soubor je modul s názvem „barmaid.py“, který při spuštění interpretem připojí ostatní moduly, nastaví eventy na které bot bude reagovat z websocketu, připojí se k Discordu, bota autorizuje a spustí, spouštěcí modul rovněž definuje funkci pro správný chod prefixových příkazů.

```
1 # Intents manages some level of permissions bot can do
2 INTENTS = Intents.default()
3 INTENTS.members, INTENTS.presences = True, True
4 INTENTS.message_content, INTENTS.reactions = True, True
```

Zdrojový kód 1: Nastavení oprávnění bota v Discord API

Ve zdrojovém kódu 1 definujeme botu přístup ke členům komunitního serveru a jejich aktivit, obsahu zpráv a jejich reakcí.

Zdrojový kód 2 je funkce, která zajistí zjištění příslušného prefixu používaném v konkrétním komunitním serveru je-li nastaven, jinak se použije symbol „.“ jako výchozí.

```

1  async def get_prefix(client:commands.Bot, message:Message):
2      if not message.guild:
3          return commands.when_mentioned_or(S.DEFAULT_SERVER_PREFIX) (
4              client, message)
5
6      prefix = await read_db(DATABASE, message.guild.id, 'prefix')
7      if not prefix:
8          is_okay = await insert_db(DATABASE, message.guild.id, 'prefix'
9              , S.DEFAULT_SERVER_PREFIX)
10         if not is_okay:
11             return
12         prefix = S.DEFAULT_SERVER_PREFIX
13     return commands.when_mentioned_or(prefix) (client, message)

```

Zdrojový kód 2: Funkce určující prefix

```

1  # Client has to be top level variable because of @ decorator for
2      events
3  CLIENT = commands.Bot(command_prefix=get_prefix, intents=INTENTS)
4
5  # Create of log handle
6  log_handle=logging.Logger = setup_logging()

```

Zdrojový kód 3: Vytvoření bota

Kód 10 zajišťuje vytvoření instance třídy *commands.Bot* za použitím prefixové funkce a oprávnění, které bot bude mít definované v předchozích zdrojových kódech 1 a 2. Také vytváří proměnnou *log_handle*, pomocí které lze zapisovat log při běhu aplikace.

```

1  @CLIENT.event
2  async def on_command_error(ctx:Context, error:commands.
3      CommandError):
4      if isinstance(error, commands.CommandNotFound):
5          await ctx.send(error, ephemeral=True)
6      return

```

Zdrojový kód 4: Ukázka callback funkce websocket eventu

Kód 4 definuje callback funkci, která bude reagovat na události z websocketu, pro ukázkou je zde uvedena pouze jedna, ale spouštěcí soubor obsahuje těchto funkcí více: *on_ready*, *on_guild_join*, *on_member_join*, *on_message*, *on_message_edit*, *on_message_error*. Tyto funkce budou ještě popsány v implementaci dále.

Zdrojový kód 7 spustí bota pomocí autorizačního tokenu, který nalezne v souboru *.env* a následně spustí funkci 6 pro načtení ostatních modulů do instance třídy bota v proměnné *CLIENT*. Kód 5 je spuštěn poté co je bot úspěšně nastar-

```

1  @CLIENT.event
2  async def setup_hook():
3      """Syncs slash commands in all connected guilds and makes
4          buttons restart persistent.
5      """
6      synced_commands = await CLIENT.tree.sync()
7      print(f"In-app commands have been synchronized.")
8      synced_commands = [sc.name for sc in synced_commands]
9      print(f"Following were affected: {synced_commands}")
10     CLIENT.add_view(EventView())

```

Zdrojový kód 5: Synchronizace příkazů

```

1  # Listed modules to be loaded into client
2  EXTENSIONS = [
3      "admin_tool",
4      "events.event",
5      "music",
6  ]
7
8  async def install_extensions(target: commands.Bot):
9      """Install all the extentions in the other files to the client.
10
11     Args:
12         client (commands.Bot): Instance of the bot itself.
13     """
14     for ext in EXTENSIONS:
15         await target.load_extension(ext)

```

Zdrojový kód 6: Instalace modulů

tován a dorazí jeden z prvních eventů websocketu k synchronizaci lomítkových příkazů, slouží k tomu funkce *setup_hook*, zároveň s tím nastavuje třídu *EventView*, o té později, jako persistentní vůči restartu, aby tlačítka jenž bot vytváří fungovali i po jeho vypnutí a následném zapnutí.

```

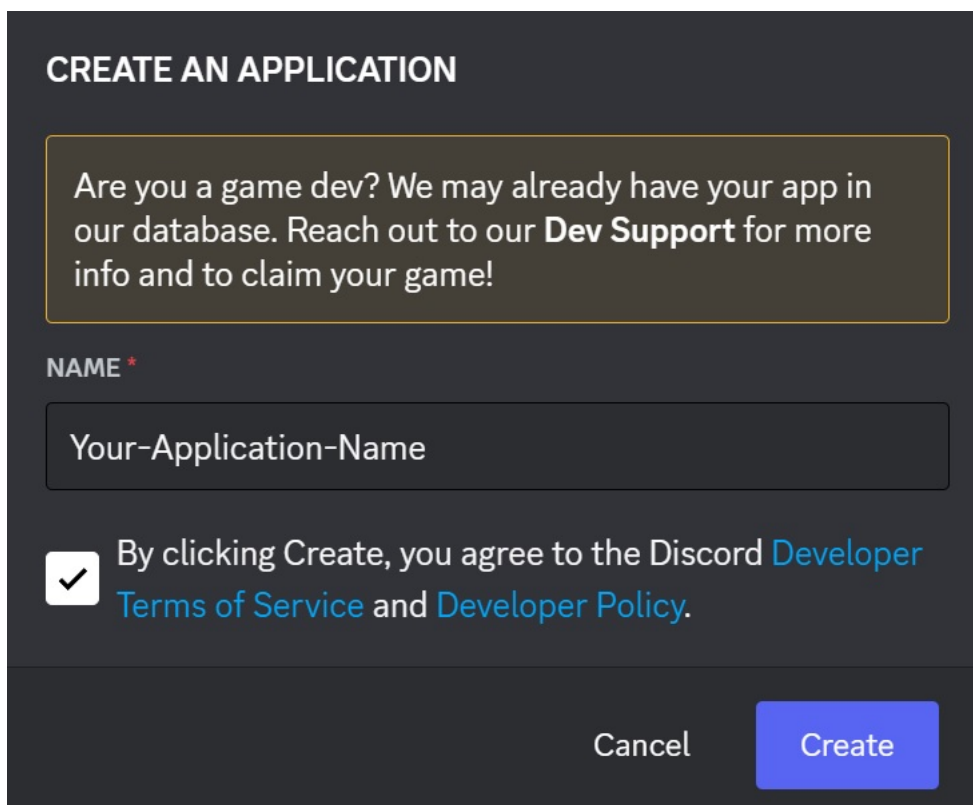
1  if __name__ == "__main__":
2      """This is main module and only one to be executed."""
3      # Loads .env file
4      load_dotenv()
5      CONNECTION_TOKEN = os.getenv("TOKEN")
6
7      # Gets the event loop
8      loop = asyncio.new_event_loop()
9
10     # Puts the load of dependency files into loop
11     loop.run_until_complete(install_extensions(CLIENT))
12     # Starts the bot client
13     CLIENT.run(CONNECTION_TOKEN)

```

Zdrojový kód 7: Spuštění bota

5.3 Vytvoření aplikace a účtu jako bot

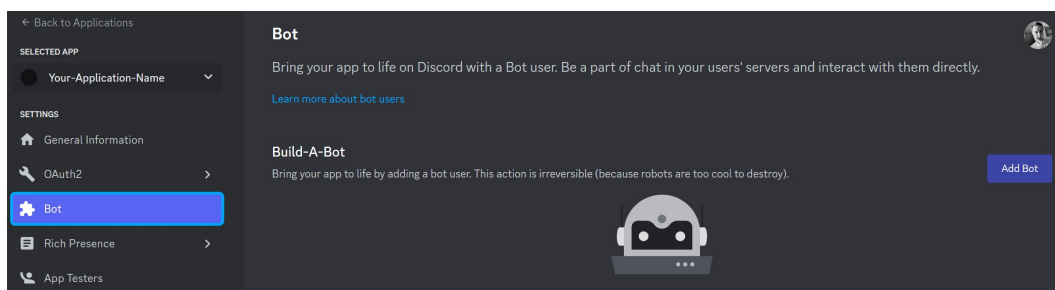
Vytvoření aplikace probíhá po přihlášení hlavním účtem uživatele na webové stránce platformy a navštívením webové adresy *discord.com/developers/applications*, zde je v pravém horním rohu tlačítko „Create Application“ a vyzvání k výběru jména aplikace viz obr. 9, toto jméno bude jménem účtu bota.



The screenshot shows a dark-themed form titled "CREATE AN APPLICATION". At the top, a yellow-bordered box contains the text: "Are you a game dev? We may already have your app in our database. Reach out to our **Dev Support** for more info and to claim your game!". Below this is a text input field labeled "NAME *" with the placeholder text "Your-Application-Name". Under the input field, there is a checked checkbox followed by the text: "By clicking Create, you agree to the Discord [Developer Terms of Service](#) and [Developer Policy](#)". At the bottom right, there are two buttons: "Cancel" and "Create".

Obrázek 9: Vytvoření aplikace

Po vytvoření aplikace je nutné přejít do sekce „Bot“ a vytvořit nového bota tlačítkem „Create Bot“ viz obr. 10.

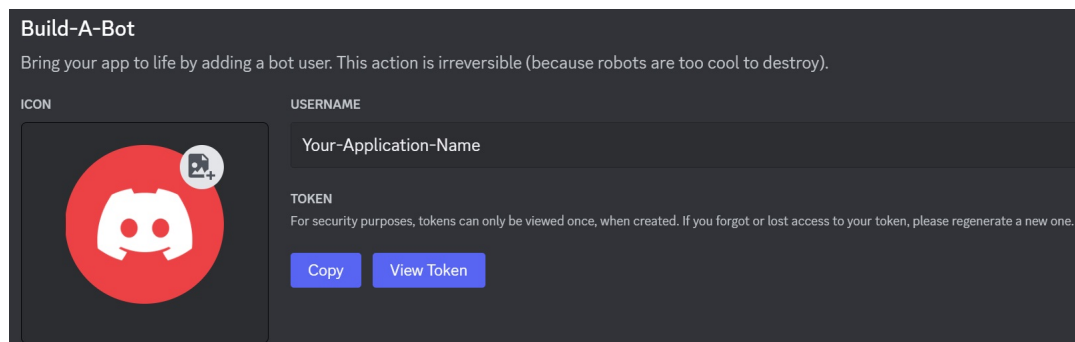


The screenshot shows the "Bot" settings page in the Discord Developer Portal. On the left is a sidebar with a "Back to Applications" link at the top. Below it is a "SELECTED APP" section showing "Your-Application-Name". Under "SETTINGS", there are links for "General Information", "OAuth2", "Bot" (which is highlighted in blue), "Rich Presence", and "App Testers". The main content area is titled "Bot" and contains the text: "Bring your app to life on Discord with a Bot user. Be a part of chat in your users' servers and interact with them directly." Below this is a link "Learn more about bot users". Further down is the "Build-A-Bot" section with the text: "Bring your app to life by adding a bot user. This action is irreversible (because robots are too cool to destroy)." and an "Add Bot" button. At the bottom of the main content area is a robot icon.

Obrázek 10: Vytvoření aplikace jako bot

Po vytvoření bota se zpřístupní účet bota a je možné zobrazit jeho autorizační token jednorázovým tlačítkem „View Token“ viz obr. 11. Tento token slouží ke

spuštění bota hlavním spouštěcím souborem, token je nutné vložit do souboru *.env-example* na příslušné místo a přejmenovat soubor na *.env*.



Build-A-Bot
Bring your app to life by adding a bot user. This action is irreversible (because robots are too cool to destroy).

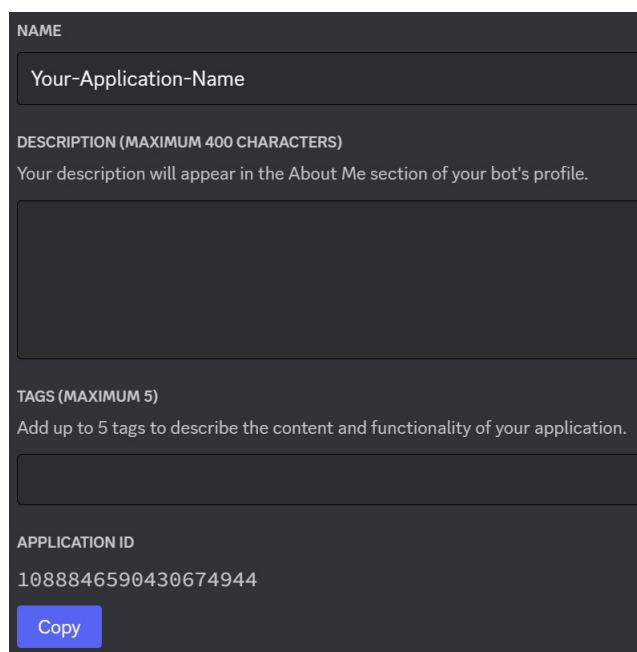
ICON

USERNAME
Your-Application-Name

TOKEN
For security purposes, tokens can only be viewed once, when created. If you forgot or lost access to your token, please regenerate a new one.

[Copy](#) [View Token](#)

Obrázek 11: Získání autorizačního tokenu bota



NAME
Your-Application-Name

DESCRIPTION (MAXIMUM 400 CHARACTERS)
Your description will appear in the About Me section of your bot's profile.

TAGS (MAXIMUM 5)
Add up to 5 tags to describe the content and functionality of your application.

APPLICATION ID
1088846590430674944
[Copy](#)

Obrázek 12: Identifikátor aplikace

V sekci „General Information“ na webové stránce lze, viz obr. 12, nalézt *application id*, kterým je nutné příslušně vyplnit soubor *config.json*.

5.4 Ukládání dat

Ukládání informací je nutné, aby bot mohl rozlišovat data týkající se pouze konkrétního komunitního serveru. Ukládá tedy pro každý komunitní server jeho prefix, pravidla, roli pro přidělení a veškerou organizaci naplánovaných událostí. Dále používá globální databázi pro záznamy uživatelů, jenž byly z některého serveru vyhoštěni či z nějakého důvodu zablokováni.

Pro dva nezávislé soubory s daty *user_records.json* a *data.json* je zaveden univerzální přístup modulem *jsonified_database.py*, který umožňuje čtení a zápis do těchto souborů na základě výběru.

```
1  async def open_file(file_name:str) -> dict:
2      try:
3          async with aiofiles.open(file_name, READ_FLAG) as f:
4              contents = await f.read()
5              json_as_dict = json.loads(contents)
6      except OSError as e:
7          log.critical(f"Opening file failed: {file_name} due {e}")
8          raise OSError(e)
9      return json_as_dict
```

Zdrojový kód 8: Otevření databázového souboru

K otevření cíleného souboru slouží kód 8, který zajišťuje asynchronní přístup k souboru a vrátí jeho celý obsah ve formátu [JSON](#).

```
1  async def flush_file(file_name:str, data:dict) -> str:
2      try:
3          async with aiofiles.open(file_name, WRITE_FLAG) as f:
4              await f.write(data)
5              return data
6      except OSError as e:
7          log.critical(f"Writing to file failed: {file_name} due {e}")
8          raise OSError(e)
```

Zdrojový kód 9: Zápis do databázového souboru

Pro zápis do cíleného souboru slouží funkce *flush_file* ze zdrojového kódu 9, který zajišťuje asynchronní přístup k souboru a zapisování do něj.

Za pomoci těchto dvou základních univerzálních funkcí jako je čtení a zápis jsme schopni vytvořit funkci, která přečte danou hodnotu podle klíče v databázi, slouží k tomu funkce *read_db* viz zdrojový kód 10.

Základní funkcí aktualizace dat v databázových souborech slouží funkce *update_db* viz zdrojový kód 11, dále modul *jsonified_database.py* obsahuje i další funkce jako *insert_db*, *delete_from_db*, *add_id* a *id_lookup*.

5.5 Struktura příkazů v kódu

Na obr. 13 je vidět struktura příkazu v kódu. Řádek č. 57 určuje pomocí dekorátoru o jaký příkaz se jedná *@commands.command()* definuje prefixovaný příkaz, *@commands.hybrid_command(with_app_invoke=True)* definuje prefixový příkaz a k němu stejnojmenný lomítkový příkaz dle pravdivostní hodnoty *with_app_invoke*, *@commands.hybrid_group()* pak podobně jako *hybrid_command* definuje hlavní

```

1  async def read_db(file_name:str, id:int, key:str):
2      id_str = str(id)
3
4      data = await open_file(file_name)
5      try:
6          value_by_key = data[id_str][key]
7          return value_by_key
8      except KeyError as e:
9          return None

```

Zdrojový kód 10: Čtení v databázovém souboru

```

1  async def update_db(file_name:str, id:int, key:str, new_value):
2      id_str = str(id)
3
4      data = await open_file(file_name)
5      try:
6          data[id_str][key] = new_value
7      except KeyError:
8          return None
9
10     new_data = json.dumps(data, indent=2)
11
12     result = await flush_file(file_name, new_data)
13     return result

```

Zdrojový kód 11: Aktualizace dat v databázovém souboru

příkaz a k němu podpříkazy, lze vidět dále v implementaci plánovaných událostí v podkapitole. Ve stejném dekorátoru lze definovat jméno příkazu, není-li definován pak je použito jako jméno název definované funkce na řádce č. 62, a případné „aliasy“ pouze pro prefixované příkazy.

Řádek č. 60 určuje rozsah platnosti příkazu, dekorátor *@commands.guild_only()* určí příkaz pouze pro použití uvnitř komunitního serveru, není-li definováno pak příkaz je platný i ve soukromé zprávě s botem.

Dekorátor *@commands.has_guild_permissions()* na řádce č. 61 ověřuje zda uživatel jenž příkaz vyvolal má dostatečná oprávnění, pokud nemá je programem vyvolána výjimka, která může být buď obsloužena pomocnou funkcí se stejnojmenným dekorátorem jako je název příkazu a vlastností error, pro tuto funkci *clear* by to bylo například *@clear.error* a funkce s libovolným jménem např. *clear_error*, pro příkad jak taková chybová funkce může vypadat viz zdrojový kód 14. Není-li taková funkce definována, je pak odchycena eventem websocketu s názvem *on_command_error*.

Řádek č. 62 definuje funkci, která se vykoná při zavolání příkazu za pomoci prefixu/lomítka dle příslušného dekorátoru u funkce, funkce musí mít vždy jeden argument *ctx* třídy *commands.Context*, který obsahuje všechny informace z cache

```

57 @commands.hybrid_command(name="clear",
58 | | | | | aliases=["clr", "delmsgs", "delmsg"],
59 | | | | | with_app_command=True)
60 @commands.guild_only()
61 @commands.has_guild_permissions(manage_messages=True)
62 async def clear(ctx: commands.Context, amount: int=1):
63 > """Removes any number of messages in chat history. ...
69     await ctx.defer(ephemeral=True)
70     # If invoked via slash commands, no need to purge invoke itself.
71     if not ctx.interaction:
72         amount += 1
73
74     if amount > 0:
75         await ctx.channel.purge(limit=amount)
76         amount = amount - 1 if not ctx.interaction else amount
77         await ctx.send(content=f"Cleared `{amount}` message(s)",
78 | | | | | delete_after=S.DELETE_COMMAND_INVOKE)
79     else:
80         await ctx.send(f"Amount of messages to be deleted has" \
81 | | | | | " to be a positive number",
82 | | | | | delete_after=S.DELETE_COMMAND_ERROR)

```

Obrázek 13: Ukázka struktury příkazu v kódu, příkaz *clear*

bota a lze pomocí tohoto objektu zjistit autora příkazu, komunitní server a na základě toho rozlišovat chování příkazu. Zbytek argumentů je na vývojáři daného příkazu, při použití statického typování pak u lomítkových příkazů dochází ke správné nápovědě, co může do argumentu vstupovat, je-li uvedena výchozí hodnota pak se tento argument stává nepovinným pro příkaz.

Zabalené řádky č. 63-68 obsahují dokumentaci funkce, jenž je pak následně použita pro popisky příkazů a jejich argumentů je-li příkaz volán jako lomítkový.

První řádek v těle funkce č. 69 slouží k obslužení příkazu hybridně, buď jako interakcí skrze lomítkový příkaz nebo prefixovanou zprávu, pokud se jedná o lomítkový příkaz je nutné odpovědět pomocí *ctx.send()* do tří vteřin, trval by výpočet déle, pak vyprší timeout interakce a *ctx.send()* vyvolá výjimku, tento řádek zvýší timeout interakce z 10ti vteřin na 15 minut. Pokud příkaz byl vyvolaný pomocí prefixové zprávy, pak tento kód řádku č. 69 nemá žádný efekt a neprovádí nic s navýšením timeoutu, pouze v obou případech může definovat skrze *ephemeral=True* viditelnost odpovědi pouze autorovi, co příkaz vyvolal.

Řádky č. 70 až 82 obsahují zbytek těla funkce a chování celého příkazu, v tomto příkladu smazání určitého počtu zpráv v textovém kanále, jenž byl příkaz spuštěn.

```

84 @clear.error
85 async def clear_error(ctx: commands.Context, error: errors):
86     # Bot missing permissions
87     await ctx.defer(ephemeral=True)
88     if isinstance(error, discord.Forbidden):
89         await ctx.send(f"{CLIENT} missing permissions `Manage Messages`",
90                       delete_after=S.DELETE_COMMAND_ERROR)
91     # User missing permission
92     elif isinstance(error, commands.MissingPermissions):
93         await ctx.send(error, delete_after=S.DELETE_COMMAND_ERROR)
94
95     if not ctx.interaction:
96         await delete_command_user_invoke(ctx, S.DELETE_COMMAND_ERROR)

```

Obrázek 14: Chybová funkce k příkazu *clear*

5.6 Pravidla komunit

Pravidla komunit spadají pod správu a nacházejí se v modulu *admin_tool.py*, zde existují příkazy *rules* pro zobrazení pravidel, *addrule* pro přidání nového pravidla ke stávajícím, *delrule* pro smazání konkrétního pravidla, *rules_reset* pro odstranění všech platných pravidel. Pro zprovoznění automatického zasílání pravidel botem musí existovat alespoň jedno pravidlo zadané uživatelem s potřebnými pravomocemi, pokud role uživatele nesplňuje příslušná oprávnění jsou mu některé z těchto příkazů nedovoleny a pouze napíší, které pravomoce jsou potřeba k jejich spuštění. K automatickému zaslání slouží callback funkce *on_member_join* reagující na event websocketu v hlavním spouštěcím souboru *barmaid.py*, tuto callback funkci také uvidíme v kapitole s názvem Kontrola uživatelů. Zde se podíváme na její část přenechání na funkci *send_guild_rules*.

Event websocketu *on_member_join* popsaný callback funkcí 12, je spuštěn při připojení nového člena do serveru a funkce přiděluje pravomoce, o tom v pozdější kapitole, a přenechává na funkci *send_guild_rules* zaslání připojenému členovi příslušná pravidla serveru. Nejsou-li stanovena žádná pravidla předem, není zasláno nic.

Funkce 13 obstarává zaslání uživateli, který se připojil příslušná pravidla serveru, která se pokusí nalézt v databázi, naformátuje jejich výstup a zapouzdří je do zprávy, kterou mu následně zašle do soukromého chatu.

Zdrojový kód 14 slouží k zobrazení pravidel příslušné komunity, pravidla získává z databáze bota a formátuje každá pravidlo na vlastní řádek s řadovou číslovkou, pokud žádná pravidla ještě neexistují informuje o tom chybovou hláškou, taktéž odstraní prefixovou zprávu pokud ta příkaz vyvolala.

K přidání nového pravidla slouží zdrojový kód 15, který prvně zkontroluje dostatečná oprávnění uživatele, získá stávající pravidla a k nim připojí pravidlo nové, poté aktualizuje záznam v databázi a oznamí o úspěšném zaznamenání pravidla, pokud něco selže, je o tom uživatel informován chybovou hláškou. Pokud

```

1  @CLIENT.event
2  async def on_member_join(member:Member):
3      # Auto-assign role given by guild
4      role_assigned_by_guild:int = await read_db(DATABASE, member.
          guild.id, "auto-role")
5      member_guild:Guild = member.guild
6      if role_assigned_by_guild:
7          role_to_give:Role = member_guild.get_role(
              role_assigned_by_guild)
8          await member.add_roles(role_to_give)
9
10     # Auto-rules given by guild
11     await send_guild_rules(member, member_guild)
12     # Aware moderators of naughty member join
13     if await check_records(member):
14         await aware_of_records(member, member_guild)

```

Zdrojový kód 12: Event (websocket) připojený člen

byla k vyvolání použita prefixová zpráva, bude odstraněna.

Výresetování všech pravidel, o to se postará zdrojový kód 16, který zkontroluje potřebné pravomoce, odstraní prefixovanou zprávu jenž příkaz vyvolala (pokud byl použit), odstraní záznam z databáze a informuje o úspěšném či neúspěšném vykonání.

Zdrojový kód 17 smaže pravidlo na příslušném indexu, příkaz zkontroluje potřebné pravomoce uživatele jenž příkaz vyvolal, pokud byla použita prefixová zpráva k jejímu vyvolání dojde k jejímu smazání, odstraní se konkrétní pravidlo na daném indexu v databázi a dojde k informování uživatele o provedení, ať úspěšném nebo neúspěšném.

5.7 Kontrola zpráv

O kontrolu zpráv se stará hlavní spouštěcí modul, neboť v něm jsou definované všechny websocket eventy, na které bot reaguje. Těmito eventy pro kontrolu zpráv poslouží *on_message* a *on_message_edit*, kde v těle těchto callback funkcí je zavolána funkce *check_blacklist*, která se stará o zkontrolování zablokovaných slov v příslušné komunitě a dojde-li ke shodě je příslušná zpráva odstraněna. Z modulu ke správě *admin_tool.py* pomáhají k definování zakázaných slov příkazy *filter show*, *filter add*, *filter remove* pro zobrazení všech zablokovaných slov nebo frází, přidání/odebrání nového slova či fráze.

Ve zdrojovém kódu 18, dojde ke i ke kontrole vlastní zprávy zaslané botem, tuto skutečnost bot ignoruje a má pro sebe výjimku. Dochází ke kontrole zda zpráva byla poslaná v komunitním serveru, pokud ano předává se zpráva ke kontrole příslušnému serveru funkcí *check_blacklist* viz 19, dále posílá zprávu ke zpracování prefix příkazů, které se mohou potencionálně ve zprávě nacházet. V neposlední řadě informuje uživatele, že bot nebude reagovat na interakce skrze

```

1  async def send_guild_rules(member:Member, guild_joined:Guild):
2      guild_rules:dict = await read_db(DATABASE, guild_joined.id, "
        guild-rules")
3      if guild_rules:
4          result:list = []
5          for idx, rule in guild_rules.items():
6              result.append(f"{int(idx)+1}. " + rule)
7          formatted_output = "\n".join(result)
8          emb = Embed(title=f"[{guild_joined.name}] server's rules:",
9                      description=formatted_output,
10                     color=Colour.red())
11         emb.set_footer(text="Being part of that server means you do
            respect these rules.")
12         await member.send(embed=emb)

```

Zdrojový kód 13: Zaslání pravidel komunity

soukromý chat s botem.

Kód 19 provádí smazání zprávy pokud zpráva obsahuje alespoň jednoho slovo či frázi definovanou příslušnou komunitou jako nevhodnou, bot dává výjimku skrze *is_blacklist_exception* funkci pouze takové zprávě, která je zamýšlena k použití při odstraňování zablokovaného slova či fráze neboť tato zpráva bude takové slovo/frázi obsahovat.

Aby nebylo možné napsat nežádoucí slova „na podruhé“ obcházením funkcionality úpravy zpráv, tak je potřeba kontrolovat nežádoucí obsah i v nich, od toho slouží zdrojový kód 20, který je zjednodušenou verzí kódu při zaslání první zprávy *on_message* 18.

5.8 Kontrola uživatelů, přidělení pravomocí

Kontrola uživatelů je zajištěna automaticky pomocí eventů websocketu *on_member_join* v hlavním spouštěcím souboru *barmaid.py*, pro nastavení této funkcionality slouží příkazy z modulu *admin_tool.py* pro nastavení moderátorů komunity, příkaz příkaz pro zobrazení záznamu konkrétního uživatele. Souvisejícími příkazy s touto funkcionalitou jsou *moderation show*, *moderation add*, *moderation reset*, *records*.

V předchozí kapitole *Pravidla komunit* již byl zdrojový kód 12, kde jsme vysvětlili jednu část funkcionality týkající se pravidel, nyní bude vysvětlena druhá část týkající se následné kontroly připojeného uživatele a přidělení pravomocí.

Zmíněný zdrojový kód obsahuje funkci *check_records*, která zkontroluje zda uživatel má záznam v databázi, pokud má je zavolána funkce *aware_of_records*, která zajistí zaslání informace všem moderátorům příslušné komunity o existenci záznamů nového člena.

Kód 21 vrací pravdivostní ohodnocení, zda uživatel má záznam v databázi, záznam má v databázi pouze pokud je u něho evidován alespoň jeden z prohřešků.

Zdrojový kód funkce 22 se postará o zjištění a nalezení definovaných mode-


```

1  @commands.hybrid_command(with_app_command=True)
2  @commands.guild_only()
3  async def rules(ctx: commands.Context):
4      await ctx.defer(ephemeral=True)
5
6      if not ctx.interaction:
7          await delete_command_user_invoke(ctx, S.
              DELETE_COMMAND_INVOKE)
8
9      guild_rules:dict = await read_db(DATABASE, ctx.guild.id,
10                                     "guild-rules")
11
12      if guild_rules:
13          result:list = []
14          for idx, rule in guild_rules.items():
15              result.append(f"{int(idx)+1}. " + rule)
16          formatted_output = "\n".join(result)
17          await ctx.send(formatted_output, delete_after=S.DELETE_MINUTE
18                          )
19      return
16      await ctx.send("No rules have been set yet.",
17                      delete_after=S.DELETE_COMMAND_INVOKE)

```

Zdrojový kód 14: Zobrazení pravidel

rátorů komunity a jejich následném upozornění a sdělené celkovém počtu evidovaných prohřešků, odkáže je jiný příkaz pomocí kterého si mohou detailněji prohlédnout o jaké prohřešky se jedná.

Zdrojový kód 12 obsahuje ještě třetí funkcionalitu, přidělení role s právomocemi novému členu, o které se nestará vlastní definovaná funkce, ale pouze nalezení takové definované role pro přidělení v komunitě a přidělením skrze metodu *add_roles()* třídy *discord.Member* z *discord.py* knihovny. Souvisejícími příkazy pro definování takové role k přidělení jsou: *autorole show*, *autorole set*, *autorole remove*.

5.9 Události a jejich spravování

Naplánování události je zajištěno pomocí příkazu *events echat*, který je zkratkou za „events external chat“, tento příkaz vytvoří událost s definovaným jménem, popiskem, místem a datem konání a možnostmi uvádět celá jména uživatelů při jejich evidenci, nebo limit počet uživatelů, kteří se mohou přihlásit. Zároveň s tím, pokud je definované datum ve správném formátu je automaticky vytvořeno oznámení 15 minut před počátkem události, nerozpoznání data dává autorovi možnost mít zde libovolný text bez funkcionality oznámení předem. Příkaz je definovaný v modulu *event.py* ve složce *events*, která zároveň definuje pohled *EventView*, kterým lze přidat tlačítka ke zprávě a zaslat ji.

Zdrojový kód 23, na kterém je vidět mezi řádky č. 10 až 18, které definují zapouzdřenou zprávu a formátují její položky (datum, místo, název apod.) do


```

1  @commands.hybrid_command(with_app_command=True)
2  @commands.guild_only()
3  async def addrule(ctx: commands.Context, *, new_rule: str):
4      await ctx.defer(ephemeral=True)
5      if not ctx.author.guild_permissions.administrator:
6          await ctx.send("Missing `administrator` permission",
7                          delete_after=S.DELETE_COMMAND_ERROR, ephemeral=
8                              True)
9      return
10     if not ctx.interaction:
11         await delete_command_user_invoke(ctx, S.
12             DELETE_COMMAND_INVOKE)
13
14     exists = await read_db(DATABASE, ctx.guild.id, "guild-rules")
15     if not exists:
16         rules:dict = {}
17     else:
18         rules = exists
19     rules[f"{len(rules)}"] = new_rule
20
21     ok_update = await update_db(DATABASE, ctx.guild.id, "guild-
22         rules", rules)
23     if ok_update:
24         await ctx.send("New rule applied.")
25     return
26     await database_fail(ctx)

```

Zdrojový kód 15: Přidání pravidla

sloupců a řádků, řádky č. 20-24 se starají u uložení do databáze. Nejpodstatnější je řádek č. 25, který odesílá formátovou událost zapouzdřenou ve zprávě jako objekt třídy *discord.Embed* a vlastními definovanými tlačítky třídou *EventView*. Na řádku č. 27 je obsluha a nastavení oznámení o události, je-li příkaz vhodně spuštěn s argumentem *start_time*, který respektuje určitý formát data a času.

Funkce *setup_notification* ze zdrojového kódu 24 je rozdělena dvou částí, první část se pokusí rozpoznat uživatelem stanovený čas, pokud se jedná o validní formát a aktuální čas je více než 15 minut do zahájení události, pozastaví se vykonávání této funkce o příslušný čas, po jejím opětovném probuzení si zkontroluje obsah zprávy bota, obsahující událost, pokud existuje a nebyla zrušena, informuje do příslušného textového kanálu o tom, že se událost blíží, rovněž vezme všechny uživatele jenž se mezitím na událost přihlásili a upozorní je v soukromém chatu.

Vytvořenou událost lze vidět na obr. 15, která je jako zapouzdřená zpráva formátovaná do sloupců a řádků pro přehlednost, pod samotnou zprávou jsou také tlačítka, které k ní přísluší a intereagují. Tato tlačítka doplňují uživatele, které jej zmáčkne do příslušného seznamu události, tlačítko „Cancel“ slouží ke zrušení události a může ji využít pouze autor.

```

1  @commands.hybrid_command(with_app_command=True,
2                             name="reset-rules",
3                             aliases=["rules_reset", "reset_rules"])
4  @commands.guild_only()
5  async def rules_reset(ctx: commands.Context):
6      await ctx.defer(ephemeral=True)
7      if not ctx.author.guild_permissions.administrator:
8          await ctx.send("Missing `administrator` permission",
9                          delete_after=S.DELETE_COMMAND_ERROR, ephemeral=
10                             True)
11      return
12      if not ctx.interaction:
13          await delete_command_user_invoke(ctx, S.
14                                           DELETE_COMMAND_INVOKE)
15
16      successful_remove = await delete_from_db(DATABASE, ctx.guild.id
17                                              , "guild-rules")
18      if successful_remove:
19          await ctx.send("Rules no longer apply.",
20                          delete_after=S.DELETE_COMMAND_INVOKE)
21      return
22      await database_fail(ctx)

```

Zdrojový kód 16: Resetování pravidel

5.10 Přehrávač

O přehrávání audio stop se stará modul *music.py*, který zajišťuje kontrolu nad audiem a samotné příkazy týkající se přehrávání audia. Modul obsahuje příkazy *stop*, *pause*, *resume* pro zastavení audia a odpojení bota z hlasového kanálu, pozastavení přehrávané skladby a opětovném pokračování v přehrávání. Příkazy *next*, *volume*, *queue*, *shazam* slouží k přeskočení hudby, nastavení hlasitosti bota, zobrazení fronty audio obsahu v pořadí a zjištění názvu audia. Pro zpracování playlistu slouží funkce *play_playlist*, která dá do fronty každý přístupný obsah z playlistu. Příkaz *play* přidá pouze jedno audio, pokud se stane, že je některé audio právě přehrávané, tak bude zařazeno do stejné fronty jako používá příkaz *play_playlist*.

Zdrojový kód 25 ukazuje na řádcích č. 6-10 připojení bota do kanálu, kde se nachází autor příkazu, řádky č. 13-17 obsluhu přehrávací fronty, mezi řádky č. 20-34 dochází k získání informací o audiu potřebné například pro seznam fronty, řádky č. 36-43 se postarají o spuštění audia pomocí funkce *play_next*.

Funkce ve zdrojového kódu 26 zprostředkovává informace o aktuálně přehrávané hudbě na řádcích č. 2-6 a řádku 27, mezi řádky č. 7-8 dochází k ukončení přehrávání je-li fronta vyčerpána, jinak se z fronty získává odkaz audia (z platformy YouTube), který je následně zpracován na řádcích 11-13. Před odesláním k přehrávání se provádí transformace audia do příslušné hlasitosti a formátu na řádcích č. 15-21. O samotné přehrávání audia se poté starají řádky č. 23-28, které

```

1  @commands.hybrid_command(with_app_command=True)
2  @commands.guild_only()
3  async def delrule(ctx:commands.Context, index:int):
4      await ctx.defer(ephemeral=True)
5      if not ctx.author.guild_permissions.administrator:
6          await ctx.send("Missing `administrator` permission",
7                          delete_after=S.DELETE_COMMAND_ERROR, ephemeral=
8                              True)
9      return
10     if not ctx.interaction:
11         await delete_command_user_invoke(ctx, S.
12             DELETE_COMMAND_INVOKE)
13
14     guild_rules:dict = await read_db(DATABASE, ctx.guild.id, "guild
15         -rules")
16     if not guild_rules:
17         await ctx.send("There are no rules therefore nothing to
18             delete.")
19     return
20     del guild_rules[str(index-1)]
21     idx = 0
22     new_dict = {}
23     for _, value in guild_rules.items():
24         new_dict[str(idx)] = value
25         idx += 1
26
27     ok_update = await update_db(DATABASE, ctx.guild.id, "guild-
28         rules", new_dict)
29     if ok_update:
30         await ctx.send(f"Rule {index} deleted.")
31     return
32     await database_fail(ctx)

```

Zdrojový kód 17: Smazání pravidla

audio spustí v příslušném kanále komunitního serveru a nastaví se po jeho ukončení opět na zavolání této funkce, kde právě přehrané audio už bylo odstraněno z fronty.

5.11 Hosting aplikace bota

Pro nasazení aplikace byl zvolena hostingová služba nazývajícím se Amazon Cloud Web Services [10], se zvoleným hardwarem pronajatého stroje s jedním virtuálním procesorem a 1GB operační paměti běžící na operačním systému Ubuntu 22.04 LTS, architektura 64-bit (x86). Tento způsob byl zvolen na základě dostatečného výkonu pro aplikaci bota a použití bezplatného účtu pro takto výpočetně nenáročný stroj. Se vzdáleným přístupem k tomuto stroji došlo k přesunutí celé aplikace bota na tento stroj, kde byly doinstalovány všechny potřebné prerekvi-

```

1  @CLIENT.event
2  async def on_message(msg:Message):
3      if msg.author == CLIENT.user:
4          return
5      if msg.guild:
6          await check_blacklist(msg.guild, msg)
7          await CLIENT.process_commands(msg)
8          return
9      await msg.channel.send("I don't serve any drinks here.")

```

Zdrojový kód 18: Event (websocket) nové zprávy

```

1  async def check_blacklist(guild:Guild, msg:Message):
2      if await is_blacklist_exception(msg):
3          return
4
5      bl = await read_db(DATABASE, guild.id, "blacklist")
6      if not bl:
7          return
8
9      msg_content = msg.content
10
11     for w in bl:
12         if w in msg_content:
13             await msg.delete()
14             await msg.author.send(f"Word \"{w}\" is restricted to use
                                     in \"{guild.name}\"")

```

Zdrojový kód 19: Kontrola obsahu zprávy

zity podle přiloženého souboru *readme.txt* u této aplikace, následně aplikace byla spuštěna.

```

1  @CLIENT.event
2  async def on_message_edit(before:Message, after:Message):
3      if after.author == CLIENT.user:
4          return
5      if after.guild:
6          await check_blacklist(after.guild, after)

```

Zdrojový kód 20: Event (websocket) upravené zprávy

```

1  async def check_records(member:Member)->bool:
2      recorded = await id_lookup(RECORDS_DB, member.id)
3      if recorded:
4          return True
5      return False

```

Zdrojový kód 21: Zjištění existence záznamu uživatele

```

1  async def aware_of_records(member:Member, guild:Guild):
2      mods_id = await read_db(DATABASE, guild.id, "mods_to_notify")
3      mods = [guild.get_member(id) for id in mods_id]
4      naughty = await read_id(RECORDS_DB, member.id)
5      naughty_items = naughty.items()
6
7      for m in mods:
8          await m.send(f"{member} joined {guild.name} with `{len(
9              naughty_items)` " +
10             "naughty records.\nUse `/records @mention` on your server
11             " +
12             "to see more information.")

```

Zdrojový kód 22: Oznámení o existenci záznamů nového uživatele

```

1  @events.command()
2  async def echat(ctx: commands.Context, include_names: bool,
3              title: str, description: str,
4              start_time: str, voice: VoiceChannel,
5              limit: int = 0):
6      # ... head of the function here
7
8      buttons = EventView()
9
10     emb = Embed(color=int("0x2f3136", 0))
11     emb.set_author(name=f"by: {ctx.author.display_name}", icon_url=
12         ctx.author.avatar.url)
13     emb.add_field(name="Name", value=title, inline=True)
14     emb.add_field(name="Date", value=original_time, inline=True)
15     emb.add_field(name="Description", value=description, inline=
16         True)
17     emb.add_field(name="Voice", value=f"<#{voice.id}>", inline=
18         False)
19     emb.add_field(name=sign_up_string, value=default_unknown_value,
20         inline=True)
21     emb.add_field(name="Declined", value=default_unknown_value,
22         inline=True)
23     emb.add_field(name="Tentative", value=default_unknown_value,
24         inline=True)
25
26     ok = await insert_db(DATABASE, ctx.guild.id, hash, {"author":
27         ctx.author.id})
28     if not ok:
29         await ctx.send("Something has failed. Try again later.",
30             delete_after=S.DELETE_COMMAND_ERROR, ephemeral=
31                 True)
32
33     return
34
35     event_message = await ctx.send(embed=emb, view=buttons)
36     try:
37         await setup_notification(ctx, emb, event_message.id,
38             start_time)
39         log.info(f"Notification set: {ctx.guild.name} at
40             {start_time}")
41     except ValueError:
42         error_message = f"Date not recognized\n" \
43             f"Notifications not applied.\nstart_time: `{original_time}`\n"
44
45         await ctx.send(content=error_message,
46             delete_after=S.DELETE_COMMAND_ERROR, ephemeral=True)

```

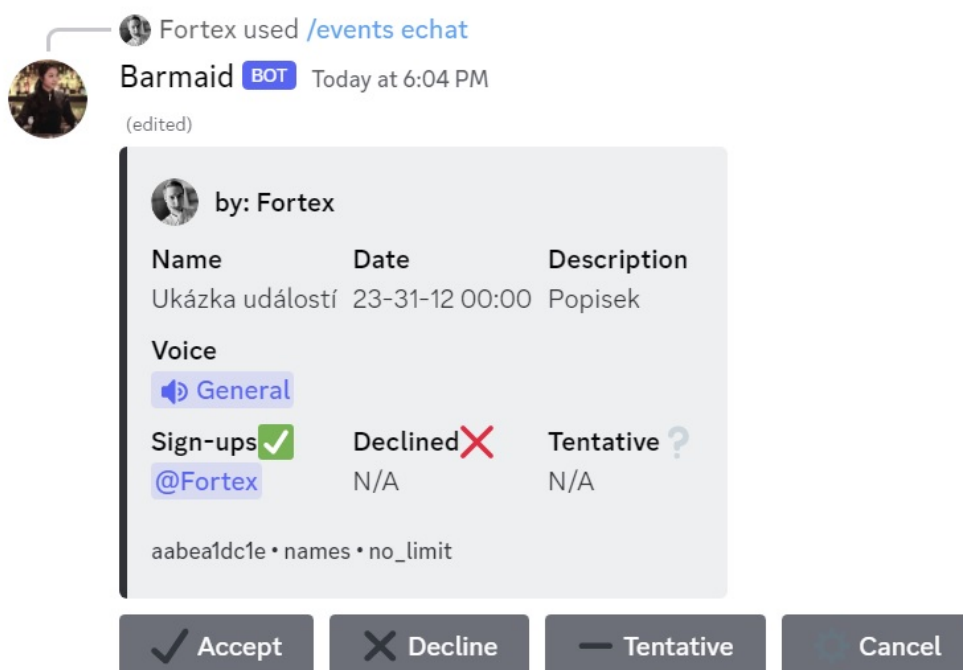
Zdrojový kód 23: Část funkce pro vytvoření události

```

1  async def setup_notification(ctx:Context, emb:Embed, message_id:int,
    time:str):
2      NAME_FIELD_POSITION = 0
3      ISO_FORMAT = "%Y-%m-%dT%H:%M:%S"
4
5      # Determine time to wait before notification
6      try:
7          iso_time_format = await format_time(time)
8      except ValueError:
9          raise ValueError("Format could have not been recognized.")
10     event_name = emb.fields[NAME_FIELD_POSITION].value
11     original_time = datetime.strptime(iso_time_format, ISO_FORMAT)
12     fifteen_mins_before = original_time - timedelta(minutes=15)
13     # CZ is UTC+1
14     time_now = datetime.utcnow() + timedelta(hours=1)
15     to_wait = fifteen_mins_before - time_now
16     if time_now > fifteen_mins_before:
17         return
18     await ctx.send(f"Notifications was set 15 minutes ahead
        successfully!",
19                 delete_after=S.DELETE_COMMAND_INVOKE)
20
21     # wait the desired time
22     await asyncio.sleep(to_wait.seconds)
23
24     # only notify if the event wasnt cancelled
25     new_updated_message = await ctx.fetch_message(message_id)
26     new_updated_embed = new_updated_message.embeds[0]
27     event_name = new_updated_embed.fields[0]
28     if not "Cancelled: " in event_name.value:
29
30         # Notify in the channel
31         await ctx.send(f"@here Event `{event_name.value}` starting
            soon!",
32                 delete_after=S.DELETE_COMMAND_INVOKE)
33
34         # Notify the signed up members
35         name_field = new_updated_embed.fields[4]
36         mentions = name_field.value
37         trick_to_get_mentions_in_list = await ctx.send(content=
            mentions)
38         await trick_to_get_mentions_in_list.delete()
39         user_mentions = trick_to_get_mentions_in_list.mentions
40         for u in user_mentions:
41             await u.send(f"Event `{event_name.value}` on `{ctx.guild.
                name}` is starting soon!")
42     log.info(f"Notification runned: {ctx.guild.name} on embed
        {embed_hash(emb)}")

```

Zdrojový kód 24: Funkce o oznámení události



Obrázek 15: Ukázka vytvořené události příkazem `events echat`


```

1  @commands.hybrid_command(with_app_command=True)
2  @commands.guild_only()
3  async def play(ctx: commands.Context, url: str):
4      await ctx.defer(ephemeral=True)
5      # connect to voice channel if not already connected
6      voice_client = _voice_clients.get(ctx.guild.id)
7      if not voice_client:
8          voice_client = await ctx.author.voice.channel.connect(
9              timeout=30, reconnect=True)
10         _voice_clients[ctx.guild.id] = voice_client
11         log.info(f"Connecting success: to {ctx.author.voice.channel.
12             name} in {ctx.guild.name}")
13
14     # add the new URL to the queue
15     queue = _queues.get(ctx.guild.id)
16     if not queue:
17         queue = asyncio.Queue()
18         _queues[ctx.guild.id] = queue
19     await queue.put(url)
20
21     # add info
22     names = _list_names.get(ctx.guild.id)
23     loop = asyncio.get_event_loop()
24     try:
25         data = await loop.run_in_executor(None, lambda: _ytdl.
26             extract_info(url, download=False))
27     except yt_dlp.DownloadError as e:
28         await ctx.send(e, S.DELETE_COMMAND_ERROR)
29         return
30     if not names:
31         name = list(data["title"])
32         name="".join([str(i) for i in name])
33         _list_names[ctx.guild.id] = [name]
34     else:
35         name = list(data["title"])
36         name="".join([str(i) for i in name])
37         _list_names[ctx.guild.id].append(name)
38
39     # start playing the next song if not already playing
40     if not voice_client.is_playing():
41         await play_next(ctx)
42         await ctx.send("Jukebox started!",
43             delete_after=S.DELETE_EPHEMERAL)
44     return
45     await ctx.send("Added to queue!",
46         delete_after=S.DELETE_EPHEMERAL)

```

Zdrojový kód 25: Přehrání audia příkmem *play*

```

1  async def play_next(ctx: commands.Context):
2      try:
3          now_playing = _list_names[ctx.guild.id].pop(0)
4      except:
5          pass
6      queue: asyncio.Queue = _queues[ctx.guild.id]
7      if queue.empty():
8          return
9      url = await queue.get()
10
11     loop = asyncio.get_event_loop()
12     data = await loop.run_in_executor(None, lambda: _ytdl.
13         extract_info(url, download=False))
14     music = data["url"]
15
16     player = discord.FFmpegPCMAudio(music, **_ffmpeg_options)
17     if not player.is_opus():
18         try:
19             current_volume: float = _volumes[ctx.guild.id]
20         except KeyError:
21             current_volume: float = 1.0
22         player = discord.PCMVolumeTransformer(player, volume=
23             current_volume)
24
25     voice_client = _voice_clients[ctx.guild.id]
26     voice_client.play(player, after=lambda e:
27         asyncio.run_coroutine_threadsafe(play_next(ctx), loop=loop))
28     voice_client.player = player
29     voice_client.now_playing = now_playing if now_playing is not None
30     else ""
31     log.info(f"Playing success: {now_playing}, at {ctx.guild.name}")

```

Zdrojový kód 26: Zpracování a přehrávání audia

Závěr

Cílem práce bylo vytvořit aplikaci bota pro platformu Discord, která umožní zlepšení uživatelské zkušenosti na platformě správou komunity, správou uživatelů, uskutečnění naplánovat události a zprostředkovat audio přehrávač.

Teoretická část práce obsahuje seznámení čtenáře s platformou Discord její funkcionalitou, popisuje technologie jenž platforma nabízí, jak fungují a dá se s nimi zacházet. Popisuje také technologie se kterými aplikace bota pracuje navíc od samotné technologie platformy.

V praktické části je popsán návrh aplikace bota, které by měl umět ke splnění zadaných požadavků a detailněji popisuje jejich vlastnosti, poté popisuje způsoby jakými byl návrh implementován a použit.

Výstupem práce je bot, který umožňuje moderátorům komunity na platformě Discord spravovat jejich komunitu způsoby jako je přiřazení pravomocí pomocí vlastní definované role, kontrolou nově připojených uživatelů a jejich záznamů trestů, které má bot dostupné a sám si je vytvořil, umožňuje stanovit pravidla platná pro komunitu a tyto pravidla jsou automaticky zaslány nově příchozímu uživateli. Bot umožňuje uskutečnit události, které jsou naplánovány pomocí příkazu a jsou zobrazeny a evidovány přímo v chatu, uživatelé se na tuto událost mohou přihlásit (odmítnout, předběžně) a budou moci být upozorněni před počátkem konání. Aplikace umožňuje také pro zlepšení uživatelské zkušenosti audio-přehrávač ve hlasovém hovoru v kanále komunity.

Conclusions

Goal of this theses was to create an application for Discord platform, which allows its users to enhance their experience on the platform with better management of the guild communities and its members, mediate an events organizer and audio player.

Theoretical part of the theses contains an introduction to the Discord platform, its functionality, describes technologies which the platform offers, how they work and how to use them. It also describes technologies which the application uses in addition to the platform itself.

The practical part describes the design of the application, which should be able to meet the requirements and more detailedly describes its features, then describes the ways in which the design was implemented and used.

The output of the theses is a bot, which allows moderators of the communities on the Discord platform to manage their community in ways such as assigning privileges using their own defined role, checking newly connected users and their records of punishments, which the bot has available and created for itself before, allows to set rules valid for the community and these rules are automatically sent to the newly arriving user. The bot allows to hold events, which are planned using a command and are displayed and recorded directly in the chat, users can register for this event (reject it, mark tentative) and can be notified before the start of the event. The application also allows for an audio player in a voice channel in the guild community to improve the user experience.

A Obsah přiloženého datového média

theses/

Kompletní text této práce v PDF formátu.

src/

Kompletní zdrojové soubory implementované aplikace, jejími konfiguračními soubory a souborem obsahující všechny knihovny třetích stran pro jejich instalaci skrze program `pip`, dostupný po nainstalování interpretu programovacího jazyka popsáném v souboru s instrukcemi `readme.txt`.

readme.txt

Instrukce pro instalaci a spuštění programu *barmaid.py*, včetně všech požadavků pro jeho bezproblémový provoz. / Instrukce a webová adresa, na které je aplikace nasazena pro účel testování při tvorbě posudků práce a pro účel obhajoby práce.

Literatura

- [1] Wikipedia. *Final Fantasy XIV*. [online]. 2013. [cit. 2023-16-03]. Dostupné z: https://en.wikipedia.org/wiki/Final_Fantasy_XIV
- [2] Wikipedia. *Discord*. [online]. 2014. [cit. 2023-16-03]. Dostupné z: https://en.wikipedia.org/wiki/Final_Fantasy_XIV
- [3] Chanty. *Slack vs Discord – Gaming, Working or Both?* [online]. 2022. [cit. 2023-16-03]. Dostupné z: <https://www.chanty.com/blog/discord-vs-slack/>
- [4] Discord. *Rate Limits*. [online]. 2015. [cit. 2023-17-03]. Dostupné z: <https://discord.com/developers/docs/topics/rate-limits>
- [5] Discord. *Discord API*. [online]. 2015. [cit. 2023-17-03]. Dostupné z: <https://discord.com/developers/docs/reference#api-reference>
- [6] Discord. *Gateway*. [online]. 2015. [cit. 2023-17-03]. Dostupné z: <https://discord.com/developers/docs/topics/gateway>
- [7] Toptal. *How to Make a Discord Bot: an Overview and Tutorial*. [online]. 2010-2023. [cit. 2023-17-03]. Dostupné z: <https://www.toptal.com/chatbot/how-to-make-a-discord-bot>
- [8] Caleb Hattingh. *Using Asyncio in Python*. O'Reilly Media, Incorporated, 2020. ISBN 978-149-2075-332
- [9] Raspberry. *Computing for everybody*. [online]. 2012. [cit. 2023-18-03]. Dostupné z: <https://www.raspberrypi.com/>
- [10] Amazon. *What is cloud computing?* [online]. 2006. [cit. 2023-18-03]. Dostupné z: <https://aws.amazon.com/what-is-cloud-computing/>
- [11] Discord. *Community Resources*. [online]. 2015. [cit. 2023-18-03]. Dostupné z: <https://discord.com/developers/docs/topics/community-resources#libraries>
- [12] Interactive Bees Blog. *Asyncio - vylepšené asynchronní paradigma*. [online]. 2022. [cit. 2023-18-03]. Dostupné z: <https://blog.neonkid.xyz/283>
- [13] Wikipedia. *FFmpeg*. [online]. 2007. [cit. 2023-18-03]. Dostupné z: <https://cs.wikipedia.org/wiki/FFmpeg>
- [14] FFmpeg. *Ffmpeg*. [online]. 2023. [cit. 2023-18-03]. Dostupné z: <https://ffmpeg.org/>
- [15] Github. *discord.py*. [online]. 2023. [cit. 2023-18-03]. Dostupné z: <https://github.com/Rapptz/discord.py>