

ZÁVĚREČNÁ STUDIJNÍ PRÁCE

dokumentace

FoxTunes – Discord bot na přehrávání hudby

David Kanovský



Obor: 18-20-M/01 INFORMAČNÍ TECHNOLOGIE
se zaměřením na počítačové sítě a programování

Třída: IT4

Školní rok: 2023/2024

Poděkování

Rád bych poděkoval Eriku Grafovi, Aleši Najserovi, Ondrovi Víchovi, Robinu Harazimovi, Jiřímu Rybovi za testování aplikace. Také bych rád poděkoval Lukáši Kanovskému za design loga.

Prohlašuji, že jsem závěrečnou práci vypracoval samostatně a uvedl veškeré použité informační zdroje.

Souhlasím, aby tato studijní práce byla použita k výukovým účelům na Střední průmyslové a umělecké škole v Opavě, Praskova 399/8.

V Opavě 31. 12. 2023

podpis autora práce

ABSTRAKT

Tento projekt se zaměřuje na vývoj Discord bota (automatizovaného uživatele) napsaného v TypeScriptu, který umožňuje uživatelům přehrávat hudbu z různých platforem, jako jsou například YouTube a Spotify pomocí jediného příkazu. Uživatelé také mají možnost vytvořit svůj vlastní playlist, do kterého si mohou ukládat hudbu. Tento playlist je automaticky nahrán do databáze MongoDB. Uživatelé si do této databáze mohou dále ukládat svou oblíbenou hudbu pomocí tlačítka embedovaného do zprávy, která obsahuje informace o momentálně hrající hudbě. Při vývoji byly využity moderní technologie a programovací jazyk TypeScript. Zvláštní důraz byl kladen na integraci Lavalinku pro optimalizované přehrávání hudby.

ABSTRACT

This project focuses on the development of a Discord bot (automated user) written in TypeScript, enabling users to play music from various platforms such as YouTube and Spotify with a single command. Users also have the option to create their own playlist to save music, which is automatically uploaded to a MongoDB database. Users can further save their favorite music to this database using a button embedded in a message containing information about the currently playing music. Modern technologies and the TypeScript programming language were used in the development, with particular emphasis on integrating Lavalink for optimized music playback.

OBSAH

ÚVOD.....	7
1 DISCORD A DALŠÍ POUŽITÉ TECHNOLOGIE	8
1.1 CO JE TO DISCORD.....	8
1.1.1 Discord API.....	8
1.1.2 Discord bot	8
1.2 TYPESCRIPT	8
1.3 DISCORD.JS	9
1.4 NODE.JS	9
1.5 LAVALINK A KNIHOVNA MAGMASTREAM	9
1.5.1 Pluginy	10
1.5.2 Magmastream	10
1.6 MONGODB A MONGOOSE.....	10
1.6.1 NoSQL databáze	10
1.6.2 Mongoose.....	11
1.7 NODEMON	11
2 ZPŮSOBY ŘEŠENÍ A POUŽITÉ POSTUPY.....	12
2.1 PŘIHLAŠOVÁNÍ KLIENTA K DISCORDU, LAVALINKU A MONGODB.....	12
2.1.1 Discord	12
2.1.2 Lavalink.....	12
2.1.3 MongoDB.....	14
2.2 LOKÁLNÍ MONGODB SERVER A DATABÁZOVÁ SCHÉMATA	14
2.2.1 Schéma playlistů	15
2.2.2 Schéma seznamu playlistů	16
2.3 MIXINS DO KNIHOVNY MAGMASTREAM.....	16
2.3.1 Co to je mixin.....	16
2.3.2 Proč mixin používám?.....	16
3 ZPŮSOBY TESTOVÁNÍ FUNKČNOSTI.....	17
3.1 VLASTNÍ FUNKCE LOGMESSAGE().....	17
3.2 VÝVOJOVÁ A PRODUKČNÍ VERZE.....	18
3.2.1 Implementace Nodemonu	18
3.2.2 Rozdělení .env souboru na dva	18
4 UŽIVATELSKÝ MANUÁL	20
4.1 PŘIDÁNÍ BOTA NA SERVER.....	20
4.2 POUŽÍVÁNÍ PŘÍKAZŮ NA SERVERU	21
4.2.1 Prefixové příkazy	21
4.2.2 Slash příkazy	21

4.3	OBECNÉ POUŽITÍ BOTA	21
4.3.1	Přehrávání hudby	22
4.3.2	Používání vlastních playlistů.....	23
4.3.3	Další jednoduché příkazy	24
ZÁVĚR		25
SEZNAM POUŽITÝCH INFORMAČNÍCH ZDROJŮ		26

ÚVOD

Discord je bezplatná komunikační platforma, která umožňuje uživatelům posílat zprávy a uskutečňovat hovory. Jedná se o populární prostředek pro organizaci chatů, sdílení obsahu a vytváření komunit online.

Discord nabízí otevřený ekosystém, který umožňuje uživatelům integrovat různé doplňky, známé jako 'bot'. Boti jsou automatizovaní uživatelé, kteří mohou přidávat různé funkce a možnosti na Discord servery.

Jelikož využívám Discord každý den, rozhodl jsem se vytvořit vlastního bota pro přehrávání hudby. K tomu mě motivovalo několik důvodů. Jedním z nich byl můj předešlý zájem o hudební rozšíření v Discordu. Hlavním důvodem však bylo mé pozorování, že mnoho existujících botů omezuje některé klíčové vlastnosti a funkce a účtuje si za jejich odemknutí. Mým cílem bylo poskytnout uživatelům bezplatnou a snadno použitelnou alternativu.

Dokumentace se skládá z celkem čtyř kapitol.

- V první se podrobněji dozvíme o tom, co to je Discord, také se dozvíme o hlavních použitých technologiích.
- Druhá kapitola obsahuje více technicky zaměřené vysvětlení implementace jednotlivých technologií a dalších důležitých funkcí.
- Ve třetí kapitole se dozvíme, jaké metody byly použity při testování aplikace.
- Čtvrtá kapitola je manuál pro uživatele, který obsahuje instrukce, jak pozvat bota na server a jak používat jeho hlavní funkce. Také zahrnuje teorii a vysvětlení věcí souvisejících s tématem.

1 DISCORD A DALŠÍ POUŽITÉ TECHNOLOGIE

1.1 Co je to Discord

Discord je komunikační platforma, která spojuje uživatele přes textové, hlasové a video komunikace. Původně byl Discord oblíbený hlavně mezi hráčskými komunitami, nyní ovšem nachází uplatnění i v různých vzdělávacích, sportovních a pracovních skupinách. Kromě přímých správ a hovorů také, umožňuje uživatelům vytvářet vlastní servery, což jsou virtuální prostory pro komunikaci. Tyto servery mohou obsahovat různé kanály, které usnadňují organizaci diskuzí na specifická témata. Kromě toho Discord nabízí systém rolí a oprávnění, umožňující přiřazovat uživatelům specifické role s vybranými oprávněními, což zajišťuje řízený přístup k různým funkcím a kanálům na serveru.

1.1.1 Discord API

Discord poskytuje rozhraní API (rozhraní pro programování aplikací), což je sada pravidel a nástrojů, které umožňují externím aplikacím komunikovat s Discord platformou. Funguje prostřednictvím HTTP požadavků. Také umožňuje vytváření Webhooků.

1.1.2 Discord bot

Discord bot je automatizovaný uživatel, vytvořený pomocí Discord API, který může provádět různé úkoly na serverech. Tihle boti mohou sloužit k moderaci, přehrávání hudby, správě rolí a poskytování dalších funkcí. Jsou programovány k reakci na příkazy nebo události na serveru, což zvyšuje interaktivitu na platformě Discord.

1.2 TypeScript

TypeScript je nadstavba nad jazykem JavaScript, která přináší do vývoje s JavaScriptem statickou typovou kontrolu. TypeScript umožňuje definovat typy proměnných, parametrů funkcí a návratových hodnot. Tímto způsobem poskytuje vývojářům možnost odhalit chyby již při psaní kódu, což zvyšuje jeho stabilitu a snižuje pravděpodobnost pádu aplikace. Taktéž se tím zlepšuje čitelnost kódu, protože získáváte lepší přehled o tom, jaké hodnoty očekávat a jak s nimi pracovat.

Má aplikace také využívá standardizované ECMAScript moduly (ES moduly). ES moduly umožňují efektivní načítání a přispívají k čisté struktuře kódu, což společně s TypeScriptem zvyšuje stabilitu a udržitelnost aplikace.

1.3 Discord.js

Discord.js je knihovna, která slouží jako spojovací článek mezi aplikací a Discord API. Tuto knihovnu jsem si zvolil záměrně, protože již mám předchozí zkušenosti s jejím používáním, a navíc se jedná o jednu z nejpoužívanějších knihoven pro komunikaci s Discord API. Také lze Discord.js používat s TypeScriptem, což přináší výhody při vývoji a zabraňuje chybám.

1.4 Node.js

Node.js je open-source multiplatformní runtime prostředí postavené na JavaScriptu. Jeho klíčovou vlastností je asynchronní zpracování událostí, což je nezbytné pro efektivní manipulaci s různými asynchronními událostmi v Discord API, například zprávami nebo událostmi uživatelů. Node.js disponuje aktivní a rozsáhlou komunitou a poskytuje širokou škálu nástrojů a knihoven, což výrazně usnadňuje integraci a rozšíření různých funkcí pro Discord boty.

1.5 Lavalink a knihovna Magmastream

Lavalink představuje hudební serverový software implementovaný v jazyce Java. Jeho úkolem je poskytovat zpracování zvukových dat pro hudební přehrávání v mém případě pro Discord bota.

Lavalink zajišťuje efektivní načítání a přehrávání hudebních skladeb z různých zdrojů, včetně populárních platforem jako např. YouTube.

Lavalink je samostatný server, který operuje nezávisle, tudíž minimalizuje zátěž na hlavní aplikaci Discord bota, čímž zajišťuje optimální výkon pro Lavalink i Node.js aplikaci.

1.5.1 Pluginy

Lavalink podporuje řadu pluginů (rozšíření aplikace), které přidávají funkčnosti aplikace. V mém případě jsem ale použil jen jeden a to LavaSrc. LavaSrc umožňuje podporu více zdrojů přehrávání, zejména Spotify.

1.5.2 Magmastream

Magmastream je knihovna sloužící ke komunikaci s Lavalink serverem. Zároveň usnadňuje psaní programu díky předdefinovaným funkcím, jako je například fronta pro skladby, metoda pro vyhledávání hudby a další.

Volba knihovny Magmastream byla motivována přítomností specifických funkcí, které jsem nenalezl u ostatních dostupných knihoven. Díky těmto funkcím jsem v některých případech urychlil a usnadnil vývoj své aplikace.

1.6 MongoDB a mongoose

MongoDB je NoSQL databáze, která ukládá data v dokumentech pro použití aplikací. MongoDB nabízí mnoho výhod oproti tradičním relačním databázím. Některé z nich jsou:

- **Flexibilní schémata dokumentů:** MongoDB umožňuje modelovat a manipulovat s prakticky jakoukoli strukturou dat snadno a rychle
- **Nativní přístup kódů k datům:** MongoDB ukládá a reprezentuje data ve formátu dokumentů, což umožňuje přístup k datům z jakéhokoli jazyka v nativním datovém typu
- **Snadná změna struktury dat:** MongoDB je navržena tak, aby umožňovala snadnou změnu struktury dat bez nutnosti vypnutí aplikace nebo webu
- **Výkonné dotazování a analýza:** MongoDB umožňuje snadné vyhledávání a analýzu dat

1.6.1 NoSQL databáze

NoSQL databáze jsou databáze, které ukládají data v jiném formátu než relační tabulky. Tyto databáze poskytují flexibilní schémata a snadno se škálují s velkým množstvím dat a vysokým počtem uživatelů. Existuje několik typů NoSQL databází, včetně dokumento-

vých, klíč-hodnota, širokých sloupců a grafů. MongoDB je příkladem NoSQL databáze, která ukládá data v dokumentech. NoSQL databáze umožňují vývojářům ukládat obrovské množství nestrukturovaných dat, což jim dává velkou flexibilitu.

1.6.2 Mongoose

Mongoose je knihovna pro Node.js, která poskytuje vyšší úroveň abstrakce nad MongoDB a umožňuje vám definovat datové modely pomocí přístupu založeného na schématu. Je navržena tak, aby fungovala s flexibilní strukturou dokumentů MongoDB a poskytuje jednoduché a intuitivní API pro dotazování a manipulaci s daty. Mongoose také poskytuje vestavěnou podporu pro ověřování dat.

1.7 Nodemon

Nodemon je nástroj pro sledování změn v kódu v průběhu vývoje. Jeho hlavním účelem je automaticky restartovat Node.js aplikaci, když detekuje změny v souborech. Tímto zajišťuje plynulý vývojový cyklus bez potřeby ručního restartování aplikace při každé úpravě.

V rámci vývoje aplikace používám dvě verze, jednu pro vývoj a druhou pro produkční prostředí. Více se tomu budu věnovat v kapitole „3.2 Vývojová a produkční verze“. Nodemon využívám pouze pro vývojovou verzi, což výrazně zrychluje testování aplikace a umožňuje ihned vidět efekty provedených změn.

2 ZPŮSOBY ŘEŠENÍ A POUŽITÉ POSTUPY

2.1 Přihlašování klienta k Discordu, Lavalinku a MongoDB

2.1.1 Discord

Aby se Node.js aplikace přihlásila k Discordu, je potřeba vytvořit uživatele na stránce Discord Developer Portal. Při vytvoření uživatele je automaticky generován API klíč, který aplikace využije k přihlášení:

```
const client = new Client({
  intents: [
    GatewayIntentBits.Guilds,
    GatewayIntentBits.GuildMessages,
    GatewayIntentBits.GuildMembers,
    GatewayIntentBits.GuildVoiceStates,
    GatewayIntentBits.MessageContent,
  ],
}) as Client & { manager: Manager };

client.login(Keys.clientToken)
```

Kód 1: Přihlášení klienta do Discordu

V kódu je vytvořen nový klient s "intents", což je součást Discord API umožňující botům reagovat na události. Klient je rovněž vytvořen s `as Client & { manager: Manager };`, což umožňuje implementaci Magmastreamu. Následně se aplikace přihlásí jako bot s API klíčem pomocí `client.login(Keys.clientToken)`. V mém případě používám: `Keys.clientToken`, kde je klíč uložen.

2.1.2 Lavalink

Přihlášení k Lavalinku probíhá podobně jako k Discordu, avšak používá se heslo místo API klíče.

První část kódu obsahuje node, který se později inicializuje a přihlásí k Lavalinku.

```
const nodes = [
  {
    host: 'localhost',
    identifier: 'main',
    password: Keys.lavalinkPassword,
    port: 2334,
    retryDelay: 5000,
    retryAmount: 500,
  }
];
```

Kód 2: Definování „nodes“

Na rozdíl od přihlašování k Discordu je zde potřeba nastavit několik dalších parametrů:

- `host: 'localhost'` – Adresa Lavalink serveru.
- `identifier: 'main'` – Identifikační jméno „nodu“.
- `password: Keys.lavalinkPassword` – Heslo na přihlášení.
- `port: 2334` – Port nastavený na Lavalink serveru.
- `retryDelay` a `retryAmount` jsou nepovinné proměnné, které určují, kolikrát a po kolika milisekundách se pokusí node připojit k Lavalink serveru, pokud ztratí spojení.

Dále je přiřazen nový Magmastream manager na Discord klienta:

```
client.manager = new CustomManager({
  nodes,
  send: (id, payload) => {
    const guild = client.guilds.cache.get(id);
    if (guild) guild.shard.send(payload);
  },
});
```

Kód 3: Přiřazení Magmastream managera na klienta a posílání hlasových dat Discordu

Dříve jsem zmínil `as Client & { manager: Manager }`; díky toho nyní můžeme na Discord klienta přiřadit Magmastream managera `client.manager = new Manager({});`. Tento manager obsahuje naše předem definované „nodes“ a funkci `send`, která zajišťuje odesílání hlasových dat Discordu. Důležitou částí je `client.on('raw', (d) => client.manager.updateVoiceState(d));`, protože posílá „raw“ události Magmastreamu. Bez ní by Magmastream nefungoval.

Nakonec inicializujeme managera a připojíme node, což provedeme až po připojení k Discordu, a to do „ready“ události Discord klienta:

```
client.once("ready", () => {
  client.manager.init(client.user!.id);
});
```

Kód 4: „Ready“ událost Discord klienta a inicializování Magmastream managera

2.1.3 MongoDB

```
await mongoose.connect(process.env.MONGODB_URI!, {
  auth: {
    username: Keys.mongodbUsername,
    password: Keys.mongodbPassword
  }
}).then(() => {
  spinnerMongodbLogin.succeed(chalk.green('Database connection success-
ful!'));
  spinnerDiscordLogin.start();
  clientConnectionStatus.isStandby = false;
  clientConnectionStatus.isMongoDBConnected = true;
}).catch((err) => {
  console.error(err);
  spinnerMongodbLogin.fail(chalk.red('Database connection failed!'));
  process.exit(1);
});
```

Kód 5: Připojení k MongoDB databázi

Jelikož připojuji k MongoDB databázi jako první, což je důležité pro rychlý a brzký přístup, v části „catch“ funkce program automaticky ukončím v případě chyby. Podobně jako při přihlašování v předchozích případech, zde používám uživatelské jméno a heslo. V části „then“ si následně nastavuji potřebné proměnné pro další načítání.

2.2 Lokální MongoDB server a databázová schémata

Pro zajištění rychlejšího přístupu jsem využil lokální MongoDB databázi. V mé aplikaci používám dvě schémata. První schéma reprezentuje samotný playlist, zatímco druhé schéma slouží k uchovávání seznamu playlistů každého uživatele.

2.2.1 Schéma playlistů

V rámci schématu playlistů ukládám identifikátor uživatele a přiřazuji pole písní. Každá píseň je reprezentována dalším schématem, který obsahuje parametry z třídy „Track“ z knihovny Mongoose. Tímto způsobem ukládám informace o skladbách do databáze.

```
const trackSchema = new mongoose.Schema({
  track: { type: String, required: true },
  artworkUrl: { type: String, required: true },
  sourceName: { type: String, required: true },
  title: { type: String, required: true },
  identifier: { type: String, required: true },
  author: { type: String, required: true },
  duration: { type: Number, required: true },
  isSeekable: { type: Boolean, required: true },
  isStream: { type: Boolean, required: true },
  uri: { type: String, required: true },
  thumbnail: { type: String, required: false },
  requester: { type: Object, required: false },
});

const customPlaylistSchema = new mongoose.Schema({
  userId: {
    type: String,
    required: true,
    unique: true
  },
  songs: [trackSchema]
});
```

Kód 6: Schéma vlastních playlistů

2.2.2 Schéma seznamu playlistů

Toto schéma je podstatně jednodušší než schéma playlistů. Zde opět ukládám identifikační číslo uživatele. Rozdíl spočívá v tom, že ukládám pouze pole názvů playlistů, které si uživatel vytvořil.

```
const playlistNamesSchema = new mongoose.Schema({
  userId: {
    type: String,
    required: true,
    unique: true
  },
  playlists: {
    type: [String],
    required: true
  }
});
```

Kód 7: Schéma názvů playlistů

2.3 Mixins do knihovny Magmastream

2.3.1 Co to je mixin

Mixin je koncept v programování, kde se jedna třída nebo objekt skládá do jiné třídy tak, aby získala nebo rozšířila její vlastnosti a metody.

2.3.2 Proč mixin používám?

Jelikož ukládám svůj vlastní objekt „Track“ do MongoDB databáze, základní validační funkce při spouštění hudby implementovaná v Magmastreamu vyhazuje chybu. Tento problém vzniká v souvislosti s přidáváním symbolu „TRACK_SYMBOL“ k objektu „Track“ během vyhledávání písničky přes Magmastream, což nelze uložit do mé databáze. Mé řešení spočívá v tom, že zabráním validační funkci ve vyhazování chyby. Díky použití TypeScriptu mám jistotu, že nemohu do funkce předat neplatný objekt, což eliminuje riziko, že by validační funkce propustila neplatný objekt.

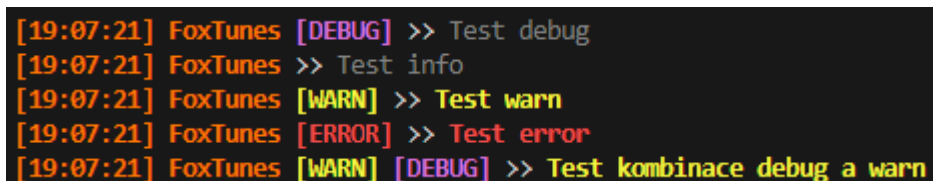
3 ZPŮSOBY TESTOVÁNÍ FUNKČNOSTI

3.1 Vlastní funkce logMessage()

Tato funkce je poměrně jednoduchá. Používám ji k úpravě výpisu do konzole pro zpřehlednění a k vypsání různých proměnných pro testování. Využívám knihovnu „chalk“ pro jednoduché obarvení konzole. Funkce nabízí několik možností pro vypsání zprávy. První z nich je „debug“, který používám k testování ve vývojové verzi aplikace. Dále si můžu vybrat, zdali chci oznámit varování nebo chybu. Níže jsou názorné ukázky výstupu:

```
logMessage('Test debug', true);
logMessage('Test info');
logMessage('Test warn', false, 'warn');
logMessage('Test error', false, 'error');
logMessage('Test kombinace debug a warn', true, 'warn');
// Debug lze také kombinovat s 'error'
```

Kód 8: Ukázka použití funkce logMessage()



```
[19:07:21] FoxTunes [DEBUG] >> Test debug
[19:07:21] FoxTunes >> Test info
[19:07:21] FoxTunes [WARN] >> Test warn
[19:07:21] FoxTunes [ERROR] >> Test error
[19:07:21] FoxTunes [WARN] [DEBUG] >> Test kombinace debug a warn
```

Obrázek 1: Ukázka výstupu funkce logMessage()

3.2 Vývojová a produkční verze

3.2.1 Implementace Nodemonu

Implementace Nodemonu je vcelku jednoduchá. Nodemon se konfiguruje v souboru `nodemon.json`. Ukázka mého souboru `nodemon.json`:

```
{
  "restartable": "rs",
  "ignore": [
    ".git",
    "node_modules/"
  ],
  "watch": ["src/"],
  "execMap": {
    "ts": "ts-node --esm"
  },
  "env": {
    "NODE_ENV": "development"
  },
  "ext": "ts,js,json"
}
```

Kód 9: Ukázka souboru `nodemon.json`

Nodemon používám pouze ve vývojové verzi aplikace. Docílím toho tak, že si v souboru `package.json` definuji tzv. „start skript“ pro obě verze aplikace.

```
"scripts": {
  "start": "node .",
  "dev": "nodemon --config nodemon.json src/index.ts",
  "build": "tsc"
},
```

Kód 10: Start skripty v souboru `package.json`

Vývojovou verzi aplikace spouštím pomocí „dev“. Produkční verzi aplikace spouštím pomocí „start“. Než však mohu spustit produkční verzi, musím použít „build“, což přeloží kód přes TypeScript kompilátor do JavaScriptu.

3.2.2 Rozdělení `.env` souboru na dva

Soubor `.env` obsahuje všechny důležité unikátní klíče a hesla používané pro přihlášení do jiných aplikací a API. Rozdělení funguje na principu, že mám dva různé uživatele (bo-

ty), kteří používají odlišné API klíče. Také mám oddělené databáze, aby v případě testování ve vývojové verzi nedošlo k poškození dat v produkční verzi. Před načtením klíčů do aplikace si určím, podle toho, která verze je zapnutá, který soubor .env použiji:

```
const EnvFile = process.env.NODE_ENV === 'development' ? '.env.dev' : '.env';
```

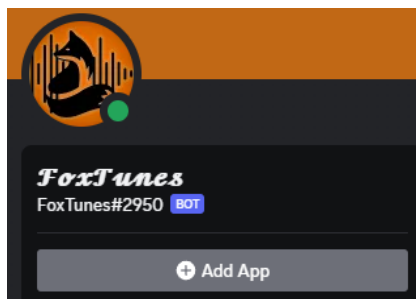
Kód 11: Výběr správného .env souboru pro momentální aplikaci

4 UŽIVATELSKÝ MANUÁL

4.1 Přidání bota na server

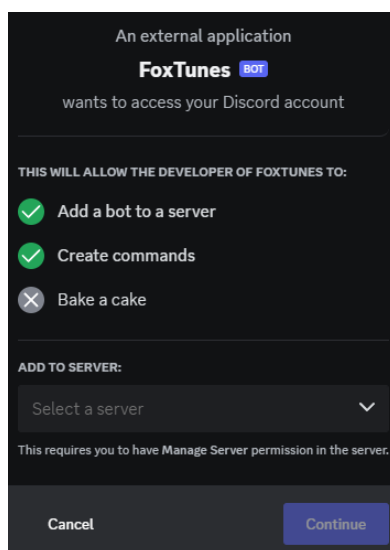
Pro používání bota musí uživatel nejprve přidat bota na svůj Discord server. Tento proces je jednoduchý a vyžaduje pouze existenci serveru a odkazu na pozvání bota. Existují dvě možnosti, jak tento odkaz získat:

1. Majitel bota odkaz zveřejní.
2. Pokud majitel bota povolí pozvání přes tlačítko, u bota se objeví možnost „Add App“, která obsahuje odkaz.



Obrázek 2: Ukázka tlačítka na přidání bota

Po kliknutí na toto tlačítko se zobrazí menu, kde si uživatel může zkontrolovat, jaká oprávnění bot potřebuje, a vybrat, na který server jej chce přidat. Po přidání může uživatel plně využívat všechny funkce bota.



Obrázek 3: Ukázka menu na přidání bota

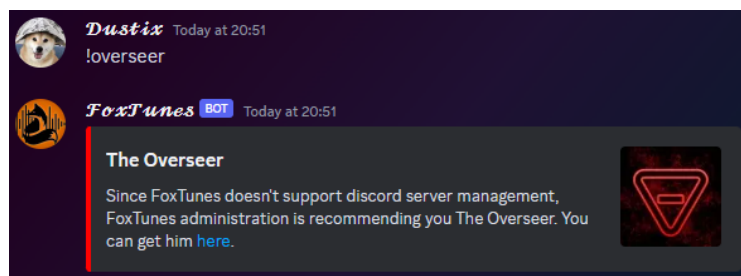
4.2 Používání příkazů na serveru

Momentálně se na Discordu používají dva typy příkazů. Do mého bota jsem implementoval obě možnosti, aby si uživatel mohl vybrat preferovanou variantu. Jediný příkaz, který nepoužívá slash příkazy, je příkaz playlist. Toto je z důvodu jeho komplexnosti.

4.2.1 Prefixové příkazy

Tyto příkazy fungují tak, že bot má v kódu definovaný prefix, v mém případě „!“. Když uživatel napíše zprávu začínající tímto prefixem, bot na ni reaguje.

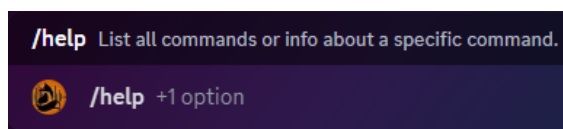
Například „overseer“ příkaz:



Obrázek 4: Ukázka použití prefixového příkazu *!overseer*

4.2.2 Slash příkazy

Na rozdíl od prefixových příkazů jsou slash příkazy přímo funkcí nabízenou Discordem. Nefungují na principu zprávy, ale na principu vytvoření interakce s botem, na kterou bot následně reaguje.



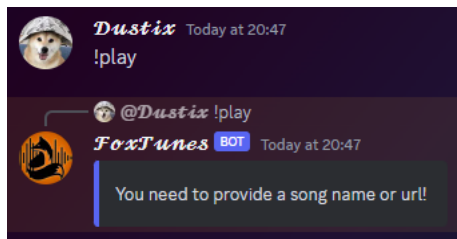
Obrázek 5: Ukázka použití slash příkazu

4.3 Obecné použití bota

Bot nabízí příkaz pro zobrazení všech dostupných příkazů a jejich funkcionalit pomocí příkazu "help".

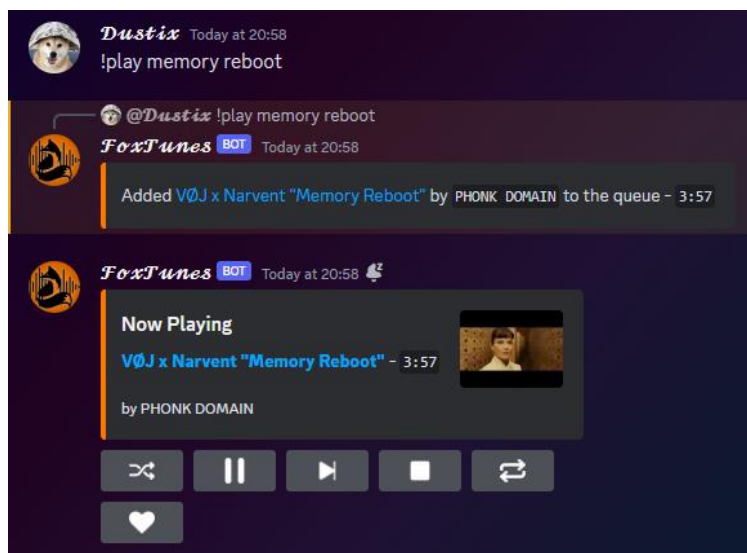
4.3.1 Přehrávání hudby

Uživatel může přehrát hudbu, pokud splňuje určité podmínky. Pokud nesplní některou z nich, bot vypíše důvod, proč uživatelovu žádost nebylo možné splnit. Například:



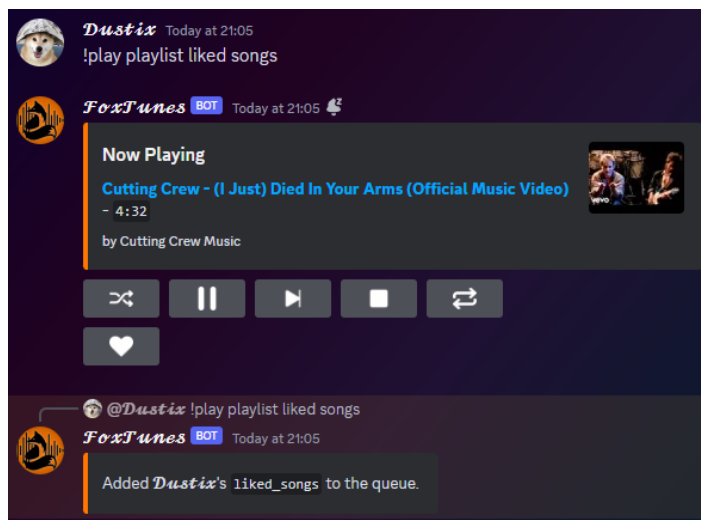
Obrázek 6: Ukázka špatně použitého příkazu „play“

Pokud uživatel splní všechny podmínky, bot se připojí do hlasového kanálu, začne přehrávat hudbu a zobrazí zprávu, díky které lze ovládat přehrávač hudby. Přehrávač lze také ovládat pomocí příkazů.



Obrázek 7: Funkční příkaz !play

Uživatel si může hudbu vyhledat pomocí jména a odkazu. Když uživatel napíše *!play playlist <název playlistu>*, může si pomocí názvu playlistu pustit některý z jeho vlastních playlistů.



Obrázek 8: Ukázka přehrání vlastního playlistu pomocí *!play* příkazu

4.3.2 Používání vlastních playlistů

Každý uživatel má možnost si vytvořit jakýkoli počet vlastních playlistů. Přidávání skladeb do playlistů, vytváření nových playlistů a vlastně vše, co se týká úpravy playlistů, lze provést pomocí příkazu *!playlist*.

Pro vypsání všech playlistů může uživatel použít příkaz *!playlist list*. Vypsání všech písní v playlistu lze provést pomocí *!playlist <název playlistu> list*.

Uživatel si playlist vytvoří jednoduchým příkazem *!playlist create <název playlistu>*. Stejným způsobem může také playlist odstranit pomocí *!playlist delete <název playlistu>*.

Do playlistu vložíme skladby pomocí *!playlist <název playlistu> add <název nebo odkaz na skladbu>*. Odebírání skladeb je trochu odlišné. Kromě odstranění skladby pomocí jejího názvu *!playlist <název playlistu> remove <název skladby>*, je také možné odstraňovat sklady podle jejich pořadí v playlistu.

4.3.3 Další jednoduché příkazy

- **Lyrics** – Příkaz vyhledá text k momentálně hrající skladbě. Text hledá pomocí knihovny genius-lyrics.
- **Shuffle** – Příkaz zamíchá aktuální frontu skladeb.
- **Queue** – Příkaz vypíše aktuální frontu následujících skladeb.
- **Insert a remove** – Příkaz insert vloží skladbu na určité místo v aktuální frontě skladeb. Naopak příkaz remove odebere skladbu z určitého místa v aktuální frontě.
- **Replay** – Příkaz přehraje naposledy přehranou skladbu.

ZÁVĚR

Cílem projektu bylo naprogramovat funkčního Discord bota, který je jednoduchý na použití a umožňuje uživatelům pohodlně přehrávat hudbu, spravovat vlastní playlisty a intuitivně ovládat přehrávač hudby prostřednictvím tlačítek. Tento Discord bot byl napsán v programovacím jazyce TypeScript a využívá technologie Lavalink pro přehrávání hudby a NoSQL databázi MongoDB pro ukládání playlistů a dalších relevantních dat.

Úspěšně jsem dosáhl všech mnou předem stanovených cílů, a navíc jsem implementoval několik funkcí, o kterých jsem z počátku ani nepřemýšlel. Během práce na projektu jsem získal cenné zkušenosti s programovacím jazykem TypeScript a s prací s NoSQL databází MongoDB.

Do budoucna mám několik nápadů na vylepšení, které bych rád implementoval.

- **Vyhledávání rádia:** Bot momentálně umí přehrát stream z rádia, ale uživatelé nemají žádnou možnost vyhledání rádia, takže si musí odkaz na stream najít na internetu sami, což někdy není zrovna jednoduché. Implementace vyhledávacího handleru pro rádia by tento problém vyřešila.
- **Více interakcí mezi uživateli:** Například zavedení možnosti hodnocení a recenzí vlastních playlistů uživatelů.

Užitečné odkazy k projektu:

- GitHub: <https://github.com/Dustix1/FoxTunes>
- Odkaz na pozvání bota:

https://discord.com/api/oauth2/authorize?client_id=888459671466295316&permissions=8&scope=bot%20applications.commands

SEZNAM POUŽITÝCH INFORMAČNÍCH ZDROJŮ

- [1] Discord.js dokumentace. Online. 2023. Dostupné z:
<https://discord.js.org/docs/packages/discord.js/14.14.1>. [cit. 2024-01-11].
- [2] Discord.js guide. Online. 2021, aktualizováno 27.12.2021. Dostupné z:
<https://discordjs.guide/>. [cit. 2024-01-11].
- [3] Npm.js. Online. Dostupné z: <https://www.npmjs.com/>. [cit. 2024-01-11].
- [4] MongoDB. Online. © 2023. Dostupné z: <https://www.mongodb.com/>. [cit. 2024-01-11].
- [5] Magmastream dokumentace. Online. Dostupné z:
<https://docs.blackforhosting.com/>. [cit. 2024-01-11].
- [5] Lavalink GitHub. Online. 2018, aktualizováno 3.12.2023. Dostupné z:
<https://github.com/lavalink-devs/Lavalink>. [cit. 2024-01-11].