Web aplikacija za obradu i pripremu podataka za prikaz na platformi Discord

Jačmenjak, Karlo

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Karlo Jačmenjak

WEB APLIKACIJA ZA OBRADU I PRIPREMU PODATAKA ZA PRIKAZ NA PLATFORMI DISCORD

ZAVRŠNI RAD

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Karlo Jačmenjak

JMBAG: 0016156498

Studij: Informacijski i poslovni sustavi

WEB APLIKACIJA ZA OBRADU I PRIPREMU PODATAKA ZA PRIKAZ NA PLATFORMI DISCORD

ZAVRŠNI RAD

Mentor:

doc. dr. sc. Matija Novak

Varaždin, rujan 2024.

Karlo Jačmenjak

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Cilj ovog rada je pokazati kako razviti web aplikaciju s programskim jezikom Python s pomoću FastAPI okvira za razvoj web aplikacija koja komunicira s web aplikacijom treće strane. Ukratko će se opisati Python programski jezik, usporediti tri aplikacije za istovremeno slanje poruka te kako na jednostavan način pomoću Python paketa (*engl. package*) kreirati novog autonomnog korisnika na platformi Discord. Pokazat će se kako implementirati funkcionalnost prikupljanja podataka sa servera video igara i Discorda te spremanje podataka u SQLite3 bazu podataka. S pomoću web aplikacije te prikaz istih podataka u klubovima na platformi Discord.

Ključne riječi: web aplikacija, aplikacija za slanje istovremenih poruka, Python, FastAPI, Uvicorn, Discord, Pycord, SQLite3

Sadržaj

1. Uvod

S rastom interesa sve veće integracije raznih usluga s ciljem da se poveća korisnička vrijednost (*engl. user value*) originalnog softverskog alata, u sve više programa, bilo to stolne, web, mobilne ili specijalizirane aplikacije, uvodi se funkcionalnosti koje ovise o uslugama koje pruža neka treća strana da međusobno komuniciraju, da se jednosmjerno ili obostrano koriste da bi povećale vrijednosti koje daju dionicima koji upotrebljavaju njihove mogućnosti. Taj porast interesa za razvoj softvera treće strane (*engl. third-party software development*) može se pripisati razvoju napredne tehnološke infrastrukture koja smanjuje ulaznu prepreku koju korisnici mogu iskusiti. Olakšava stvaranje korisnički generiranog sadržaja (*engl. user-generated content, dalje u tekstu UGC*) koji može dosegnuti korisnike na globalnoj razini na velikom broju online platformi [1].

Za veliki broj poznatih i ustanovljenih aplikacijsko programska sučelja (*engl. application programming interfaces dalje u tekstu API*). Vlasnici tog API-ja ili zajednice koje upotrebljavaju to sučelje stvaraju programske knjižnice ili komplete za razvoj softvera (*engl. Software development kits, dalje u tekstu SDK*) za jedan ili više programskih jezika. Tako Amazon nudi za nekolicinu programskih jezika knjižnice i alate kako bi se pristupilo njihovom API-ju od usluga kao što su Amazon Web Services, (*dalje u tekstu AWS*), Selling Partner API, Advertising API i mnoge druge usluge, ali promovira i knjižnice koje su kreirane od strane zajednica, posebno bi se mogla istaknuti stranica `https://dev.twitch.tv/code/` gdje Amazon promovira resurse kreirane od zajednice za platformu Twitch.

Često takvi alati su zbog jednostavnosti zato što mogu pristupiti dodatnim resursima preko Interneta i praktičnosti jer API-ji za usluge su također izvedene kao web aplikacije. To im omogućava da budu fleksibilne što se tiče razvoja funkcionalnosti, skalabilnosti sustava, dostupnost usluga i drugih kvaliteta koje su poželjne da ovakav alat ima kada bude otvoren za javnost na korištenje. Uz to što web aplikacija nudi mogućnost da manipulira API-jem, također može nuditi druge usluge kao što su web stranice, administrativna sučelja za uređivanje i personaliziranje korisničkog iskustva prilikom korištenja njihove usluge, kao što nudi web aplikacija NightBot `https://nightbot.tv/`

Za izradu web aplikacije prvi korak je odlučiti u kojem programskom jeziku želimo implementirati. U trenutku pisanja najpopularnije jezike prema [2] koji imaju mogućnost izrade web aplikacije uključuje Javascript, Python, C# i druge jezike. Svi navedeni jezici također dolaze s velikim brojem drugih knjižnica koje mogu doprinijeti vrijednosti koju aplikacija može pružiti i istovremeno olakšati i ubrzati razvojnom timu vrijeme do tržišta (*engl. time to market*). Neki primjeri funkcionalnosti koje knjižnica može pružati jest generiranje različitih tipova datoteka, statističke analize nad podacima, pristupi trajnim pohranama podataka i druge funkcionalnosti.

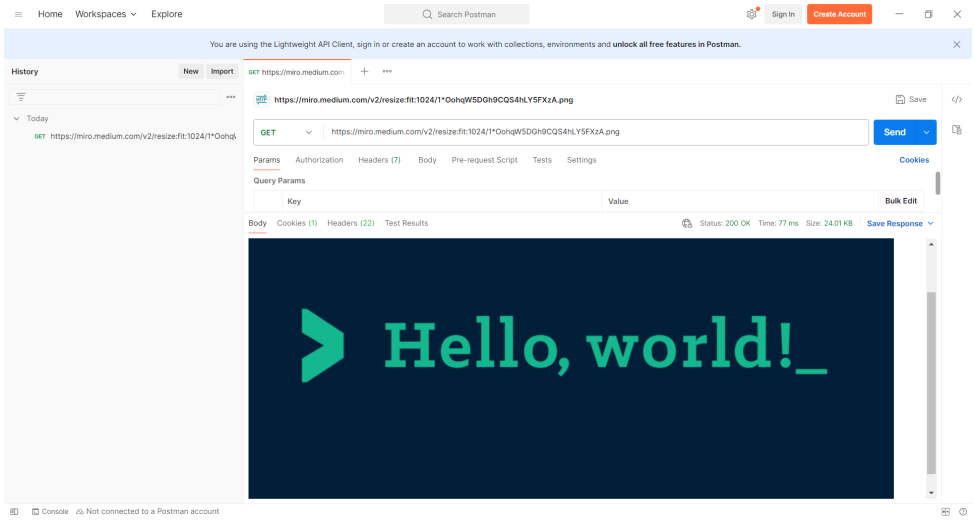
Ostatak ovog rada je organiziran na sljedeći način: poglavlje 2 opisuje razlike između web aplikacije i web stranice te način dohvaćanja web sadržaja, poglavlje 3 uspoređuje tri aplikacije za slanje poruka i korištenje API-ja jedne od njih, poglavlje 4 govori o razvoju web aplikacija i programskom jeziku Python, dok poglavlje 5 prikazuje primjer razvijene web aplikacije u FastAPI-ju.

2. Web aplikacije

Prema Taivalsaari i dr. [3] moderno doba razvoja web aplikacija ušlo je u uzbudljivo doba svojeg postojanja. Taivalsaari i dr. predviđaju da će Internet postati najdominantnija platforma za razvoj softvera za krajnje korisnike. Tehnologije za razvoj web aplikacija se približavaju se po karakteristikama nativnim aplikacijama pisanim za specifičan hardver, odnosno operacijski sustav. Uz to nude prednosti kao što su dijeljenje informacija s uređajima gotovo bilo gdje u svijetu, smanjenje potrebe za visokom specifikacijom krajnjih klijentskih uređaja, a najbitnija je što smanjuju potrebu za visokom razinom poznavanja računala i računalne tehnologije.

Važno je uočiti da postoji velika razlika između web stranica i web aplikacija. Glavna karakteristika web stranica je grupa dokumenata koji su međusobno srodne sadržajno te na neki način i povezane bilo to hipervezama ili drugim mehanizmom. Web stranica se može sastojati od samo jednog dokumenta, no danas je češće da su one izrađene od više unikatnih statičnih dokumenata koji zajedno predstavljaju neki sadržaj. Klijent najčešće pristupa sadržaju tako da pošalje zahtjev poslužitelju, koji onda klijentu postepeno vraća sav potreban sadržaj nakon čega se mrežna veza između oba računala zatvara.

Za razliku od web stranice, web aplikacija je softverska aplikacija koja se nalazi na web poslužitelju. Web aplikacije obično su napisane u skriptnom jeziku kao što su PHP, Perl ili Java i koriste bazu podataka kao što je MySQL ili PostgreSQL za pohranu podataka. Sadržaj koje web aplikacije poslužuju uglavnom je dinamičke prirode, kao što je generiranje dokumenata, upravljanje sadržaja koji klijent vidi na temelju klijentovog zahtjeva. Naravno, web aplikacije mogu pružati i statičan sadržaj po potrebi. Kod web aplikacija, veza između klijenta i poslužitelja se održava sve dok klijent tu vezu prekine, a rijetko vezu prekida i poslužitelj web aplikacije.



Slika 1: Prikaz dohvaćanja slike pomoću Postman programa [autorski rad]

Za većinu ljudi web stranice i web aplikacije otvaraju se pomoću web preglednika kao što su Google Chrome, Firefox, Safari i sl. No postoje alternativne metode pristupa istima, kao što su alati za naredbenu liniju curl, aplikacija Postman kao na slici 1 i drugi alati.

Ovi alati sadržaje koje web poslužitelji pružaju dohvaćaju i najčešće ne tumače njihov sadržaj za poseban prikaz kao što rade web preglednici. To bi se moglo smatrati nepraktičnim za ljudskog korisnika aplikacije, no ovaj pristup dohvaćanja sadržaja odgovara računalnim sustavima. Primjer takvih sustav su klijentske aplikacije koje pretvaraju taj sadržaj u format čitljiv ljudima (aplikacije kao što su Postman jer može tumačiti neke standardne formate podataka i prikazati ih kao na slici 1), druge web aplikacije koje ovise o tom sadržaju kao što su ugrađivanje njihovih sadržaja u svoje stranice, web puzaći (*engl. web crawler*) koji mapiraju pitanje koje nudi naša stranica za svoje potrebe, a više o njima može se pročitati na ovoj poveznici: `https://www.cloudflare.com/learning/bots/what-is-a-web-crawler/`.

Web aplikacije mogu postojati i bez ikakvog grafičkog korisničkog sučelja npr. mreža za isporuku slikovnog sadržaja (*engl. image content delivery network*) kao što je TransformImgs `https://github.com/Pixboost/transformimgs` te korištenjem druge web aplikacije kao zamjena za svoje grafičko korisničko sučelje kao što su Discord automatizirani računi, a često mogu koristiti više oblika istovremeno radi olakšavanja korištenja njihovih usluga.

U današnjem ekosustavu interneta komunikacija između više web aplikacija postaje norma, bilo to komunikacija između web aplikacija trećih strana ili s internim web aplikacijama poduzeća, jer dolazi do sve većeg pritiska korisnika na velike platforme kako bi mogle dostaviti svoje usluge velikom broju korisnika na malim područjima, kap što su veliki gradovi, te istu uslugu nuditi na globalnoj razini uz što manje kašnjenja odaziva na zahtjeve klijentskih uređaja. Načini na koje ove probleme rješavaju su flote poslužitelja (*engl. fleet of servers*) koji pokreću replike iste web aplikacije kako bi ravnomjerno distribuirali promet na aplikaciju.

Bilo to replika web aplikacija u floti poslužitelja ili komunikacija dvaju zasebnih web aplikacija na internetu,komunikacija između njihovim se odvija primarno s pomoću API-ja. API je skup pravila ili protokola koji softverskim aplikacijama omogućuje međusobnu komunikaciju radi razmjene podataka, značajki i funkcionalnosti [4]. API dolazi u više oblika, kao što su lokalni API (unutar istog uređaja), web API preko internetske veze, udaljeni API-iji preko alternativnih načina komunikacije kao što su radiovalovi. Za web aplikacije najčešća opcija jest komunikacija s pomoću web API-ija, no ako uvjeti dopuštaju web aplikacije mogu komunicirati lokalnim API-ijem.

Uz to API-iju web aplikacije može također pristupiti običan korisnik web aplikacije kako bi mogao koristiti aplikaciju za njenu opću namjenu kao što je dohvat podataka. No naspram toga API može biti dizajniran s ciljem da korisnik može urediti svoje korisničko iskustvo ili iskustvo veće zajednice vlastitim aplikacijama. Prema Cai i dr. [1] takve platforme svoje mogućnosti da ispune raznovrsne potrebe svojih podzajednica korisnika, ove platforme uz standardne usluge pružaju API-je otvorenog pristupa da bi razvojni programeri treće strane imali mogućnost da razne alate (npr. automatizirane račune, ekstenzije, pluginove) dodaju na platformu da pomognu sebi ili drugim korisnicima u upravljanju standardne usluge ili zajednice sa specifičnim potrebama ili normama, pri tome stvarajući raznovrsno i rastuće tržište alata treće strane.

2.1. Tipovi web aplikacija

Postoje dva osnovna tipa web aplikacija nad kojim se grade svi drugi tipovi web aplikacija, a to su statične i dinamičke web aplikacije. Glavna razlika jest u tome kakav sadržaj daju klijentu i kako se taj sadržaj ponaša prema web aplikaciji nakon što ga korisnik u potpunosti preuzme.

Statične web aplikacije

Da bi se web aplikacija smatrala statičkom, sadržaj koji nudi mora biti uvijek isti bez obzira kada i koliko puta se taj sadržaj zatraži. To znači da iako sadržaj web stranice koji nam vraća web aplikacija animiran, dohvaća podatke s API-ja treće strane i generalno bi se smatrao dinamičnim, je poslano s statičke web aplikacija jer nakon uspješnog prijenosa stranice zatvara se veza i tek nakon što korisnik traži osvježavanje sadržaja ponovno šalje istu stranicu.

Statične web aplikacije često nude sadržaje koji su vrlo malog volumena pa tako su generalno brži za dohvatiti s web aplikacije, često nisu dinamičke prirode pa tako ih većina klijentskih aplikacija može tumačiti bez problema s kompatibilnošću funkcionalnosti. Rade izvrsno kada klijent izgubio internetsku vezu ili je prešao u način rada izvan mreže, jer ako se web stranica u potpunosti dohvatila, sadržaj same web aplikacije ne zahtijeva nikakve dorade od same web aplikacije. Ovakve stranice su izvrsne u situacijama gdje želimo prikazati informacije koje su uvijek pristupačne web aplikaciji kao što je sadržaj datoteke.

Primjer situacija gdje se ovakav tip web aplikacija bio idealan za korištenje uključuje blogove, web stranice poduzeća koje predstavljaju područje rada poduzeća, online enciklopedije, priručnici i web stranice s vijestima.

Dinamične web aplikacije

Nasuprot statičkim web aplikacijama stoje dinamičke, koje kontinuirano komuniciraju sa klijent aplikacijom, bilo to preglednik ili drugi program. To često uključuje ažuriranje sadržaja web stranice s novim sadržajem kojeg klijentska aplikacija nije imala prethodno, zamjenom tekućeg sadržaja stranice s drugim sadržajem bez potrebe da se web aplikacija osvježi u pregledniku i slične aktivnosti.

Iako nisu fleksibilne kao i statične web aplikacije kada rade bez internetske veze, nadoknađuju tu manu s velikom količinom sadržaja koji često doprinose drugi korisnici, koji su s pomoću posebnih algoritama odabrani da bi zadovoljili klijentove interese i potrebe.

Velika većina web aplikacija danas koje korisnici danas koriste jesu dinamičke web aplikacije jer omogućavaju rad nad podacima koji se trajno moraju spremiti da bi korisnik mogao kasnije ih upotrijebiti. Primjer takvih aplikacija su web trgovine, socijalne stranice, sustavi za podržavanje učenja na daljinu, razne baze znanja i arhive i mnoge druge.

3. Aplikacije za slanje istovremenih poruka

Prema Bridgewater i dr. [5] aplikacije za slanje istovremenih poruka (*engl. Instant messaging applications*) je vrsta internetske usluge koja dopušta korisnicima da međusobno komuniciraju u stvarnom vremenu. Aplikacije za slanje istovremenih poruka generalno se baziraju na tekstualnoj razmjeni poruka, no dosta usluga nudi dijeljenje datoteka kao što su digitalne slike, audio-zapisi i videozapisi, dijeljenje web sadržaja s pomoću poveznica te prijenos sadržaja (*engl. streaming content*).

Komunikacija od osobe do osobe s pomoću računalnog sustava postoje od kada su se ti sustavi prvi puta počeli umrežavati, navode Bridgewater i dr. Bilo s pomoću ravnopravne mreže (*engl. peer-to-peer network*) ili protokola kao što je IRC (*engl. Internet Relay Chat*).

Prvi oblik modernih aplikacija za slanje istovremenih poruka razvila je Izraelska kompanija Mirablis pod nazivom ICQ. Da bi korisnik mogao koristiti ICQ usluge, oni prvi puta dobivaju klijentsku aplikaciju koja se brine o komunikaciji s ICQ serverom da ostvari komunikaciju s drugim korisnicima, te je nudila mogućnost da klijent pokaže da je aktivan drugim korisnicima u svojem popisu kontakata. Mirablisov model aplikacije za slanje istovremenih poruka ostao je i danas popularan model na kojem se baziraju sve moderne popularne aplikacije koje spadaju u tu kategoriju aplikacija za istovremeno slanje poruka.

S razvojem tehnologije, mnoge moderne usluge za istovremeno slanje poruka usvajaju klijentske aplikacije bazirane na web tehnologiji kao što su web preglednici i web aplikacije kao jedna metoda pristupa njihovim uslugama. To omogućava korisnicima platforme da pristupaju njihovim uslugama s gotovo bilo kojeg računalnog sustava, bilo to stolna računala ili mobiteli, s jedinim preduvjetom da podržava pristup internetu i pokretanje web preglednika. Moderne aplikacije za slanje istovremenih poruka generalno nasljeđuju sljedeće komponenta korisničkog sučelja na klijentskoj aplikaciji koje je ICQ ustanovio ICQ 1996 [5]:

- Popis kontakata
  - Sadrži korisnike s kojima je korisnik komunicirao, te često omogućuje dodavanje i uklanjanje korisnika s te liste. Moderne aplikacije u ovu listu s ciljem poboljšavanja korisničkog iskustva uključuju i grupne razgovore više korisnika i slične funkcionalnosti.
- Informacije o računu i korisničke postavke
  - Dopušta korisniku usluge da promjeni podatke svog korisničkog računa, kao što su e-mail adresa i lozinka, te često postavke same aplikacije kao što su izgled aplikacije, upravljanje vidljivosti korisnikovih aktivnosti drugim korisnicima i druge opcije.
- Prozor za razgovor s drugom osobom
  - Omogućava korisniku da šalje i prima podatke od sugovornika te da gleda kroz povijest razgovora s tim sugovornikom. Danas nerijetko se ovaj prozor može prilagoditi od strane korisnika, kao što su promjena dizajna prozora, dodavanje nadimaka svim govornicima u razgovoru i slično.

3.1. Usporedba aplikacija za slanje istovremenih poruka

Osim navedenih funkcionalnosti iz prethodnog poglavlja, moderni razvojni timovi unaprjeđuju standardne komponente aplikacije te uvode nove funkcionalnosti koje unaprjeđuju korisničko iskustvo i daju korisniku dodatnu vrijednost. Zbog toga je jako teško kvalitetno usporediti usluge aplikacija za slanje istovremenih poruka, jer često svoje slabije korisničko iskustvo kroz određeno vrijeme unaprijede ili drastično promjene.

Radi toga vrlo je važno odabrati platformu koja nudi potrebne kvalitete za lakšu izradu alata kojega planiramo izraditi. Najvažnija karakteristika potrebna za praktični rad jest mogućnost izrade automatiziranih računa. Automatizirani računi, često dokumentacije API-ja nazivaju ih bot od riječi robot. To su računi koji imaju dio ili sve mogućnosti kao i obični korisnički račun, kao što je slanje i primanje poruka, ali uz prednost da dopušta da se koristi bez klijentske aplikacije nego direktno kroz API platforme. Time se postiže programirljivost tih računa bez da se pokuša tumačiti klijentska aplikacija, što je često neželjeno ponašanje korisničkog računa u pogledu organizacije iza platforme, stoga omogućavaju automatizirane račune na svojim platformama. Uz to za potrebe praktičnog rada aplikacije koja će komunicirati s jednom od platformi za slanje istovremenih poruka, usporedit će se na temelju popularnosti aplikacije i jednostavnosti izrade bot računa, kroz sljedeće kriterije:

- Broj mjesečno aktivnih korisnika (*engl. Monthly active users*)
- Broj registriranih korisnika koji koriste uslugu
- Javni API za interakciju s platformom
- Mogućnost stvaranja automatiziranih računa
- Postojanje knjižnice za razvoj automatiziranih računa (*engl. software development kit*, dalje u tekstu SDK)

3.1.1. Platforma Matrix.org

Početna web stranica platforme: `https://matrix.org/`

Platforma Matrix.org je decentralizirana, otvorena mreža aplikacija za slanje istovremenih poruka. Omogućava sudionicima mreže da implementiraju svoje vlastite web aplikacije koje podržavaju Matrix otvoreni standard za sigurnu i interoperabilnu komunikaciju u stvarnom vremenu. Ti serveri se u njihovom ekosustavu kolokvijalno nazivaju *engl. homeserver* ili matični poslužitelj. S obzirom na to da Matrix protokol decentraliziran, prilikom registracije korisnik mora odabrati koji homeserver koji će biti glavni za korisnički račun, slično kao što prilikom izrade e-mail korisničkog računa korisnik mora odabrati poslužitelja koji upravlja komunikacijom do drugih web aplikacija koje podržavaju Matrix protokol. Također kao i poslužiteljske aplikacije sudionici mogu implementirati vlastite klijentske aplikacije za korisnike platforme kako bi mogli sudjelovati u ekosustavu Matrix matičnih poslužitelja. Neke od popularnih klijentskih aplikacija su Element, Nheko i Cinny.

Također Matrix.org nudi bogatu dokumentaciju za razvoj i klijentskih i web aplikacija, uz normalnu dokumentaciju postoji i dokumentacija za korištenje automatiziranih računa. koje koriste Matrix standard, uz to postoje SDK biblioteke razvijene i održavane od strane zajednice na poveznici `https://matrix.org/ecosystem/sdks/`.

Za većinu web aplikacije i klijentskih aplikacija dostupan je izvorni programski kod pod licencama otvorenog programskog koda kao što su MIT, Apache 2.0 i GPLv3, što omogućava gotovo potpunu mogućnost uređivanja korisničkog iskustva po vlastitim potrebama i proširenje mogućnosti web aplikacija.

Uz to Matrix.org ima veliku zajednicu korisnika koji aktivno rade na razvoju platforme, kako financijskom podrškom tako i doprinosom programskog koda. Pa tako prema Simmons, Eylenburg, Manning, Calixte i dr. za Matrix.org u 2023. godini bilo je:

- 7.5 milijuna mjesečno aktivnih korisnika [6]
- Oko 115 milijuna registriranih korisnika [6]
- Više od milijun mobilnih korisnika [7]

3.1.2. Platforma Telegram

Početna web stranica platforme: `https://telegram.org/`

Telegram je platforma za slanje istovremenih aplikacija s fokusom na sigurnost i brzinu. Dopušta jednostavnu sinkronizaciju poruka i datoteka između svih uređaja korisnika kao što su mobiteli, stolna računala i slični uređaji. Uz osnovne usluge web aplikacije za slanje istovremenih poruka nudi i mogućnost pretraživanja interneta s pomoću ugrađenog web preglednika, preuzimanje ugrađenih aplikacija unutar Telegrama, objava sadržaja koji mogu pogledati kontakti korisnika na jedan dan i slične funkcionalnosti.

Jedan od glavnih fokusa Telegram organizacije jest omogućiti programerima da stvaraju aplikacije koje upotrebljavaju njihov API, što se postiže korištenjem njihovom bibliotekom za razvoj aplikacija na više platformi Telegram Database Library i omogućavanje korištenja automatiziranih računa na platformi. Uz to su službene klijentske aplikacije licencirane pod licencama GPL v2 i v3, AGPL v3 što može pomoći u razvoju vlastitih inačica. Za mobilne uređaje razvili su protokol pod nazivom MTProto Mobile kako bi ubrzali prijenos podataka na slabim mobilnim internetskim vezama.

Prema Dean, Eylenburg, Manning, Calixte i dr. za Matrix.org u 2023. godini bilo je:

- 500 milijuna mobilnih korisnika [7]
- 900 milijuna mjesečno aktivnih korisnika [8]

3.1.3. Platforma Discord

Početna web stranica platforme: `https://discord.com/`

Discord je poznata platforma za komunikaciju i stvaranje zajednica oko zajedničkih interesa ljudi koji koriste ovu platformu. Discord ulazi u tržište aplikacija za slanje istovremenih poruka 2015. godine, da bi se suprotstavio dvjema duboko ukorijenjenim aplikacijama za sličnu namjenu: Skype i TeamSpeak [9]. Naime fokus Discorda je bio u začetcima da pojednostavi komunikaciju grupe ljudi koja zajedno uživa u igranju videoigri, no danas ugošćuje raznolik niz zajednica vezanih uz gotovo svaku sferu ljudskih interesa.

Uz to daje mogućnost integracije s drugim web uslugama kao što je usluga audio streaming Spotify, prikaz sumiranja događaja unutar aplikacija koje koristimo kao što su događaji u videoigrama ili koji se dokumenti uređuju u tekstualnim uređivačima uz funkcionalnost Discord Rich Presence, stvaranje kluba (*engl. guild*), ekvivalent grupnih razgovora u drugim platformama s dodatnim funkcionalnostima kao što su organiziranje prava korisnika pomoću uloga, stvaranje i grupiranje tekstualnih i audiovizualnih kanala za komunikaciju, stvaranje vlastitih emoji ikona za slanje u tekstualnim kanalima, i mnoge druge mogućnosti.

Uz to Discord nudi pristup svojim bogatim API-u uz prethodnu registraciju Discord računa na njihovoj platformi. API su nazvali Discord Developer Portal, koji se može istražiti na poveznici `https://discord.com/developers/docs/reference`.

Uz to prema Curry, platforma Discord imala je u 2023. godini:

- 560 milijuna registriranih korisnika
- 200 milijuna mjesečno aktivnih korisnika

3.1.4. Odabir platforme

Uz prethodnu analizu platformi potrebno je odabrati platformu na kojoj će se aplikacija razviti. U tablici 1 nalazi se sumirana usporedba ovih platformi za slanje istovremenih poruka u tabličnom obliku. Iz tablice se vidi da sve aplikacije za slanje istovremenih poruka nude zadovoljavajuća svojstva koje smo tražili od njih, pa se odluka o tome koju platformu želimo odabrati svodi na preference svojstava koje nude platforme. Na Matrix.org nudi veliku slobodu o tome kako želimo sudjelovati u njihovoj zajednici te mogućnost da stvorimo vlastite implementacije klijentske i web aplikacije ako nam je to potrebno. Telegram nudi ogromno zajednicu korisnika te fokus na privatnost i sigurnost korisnika. Discord nudi pregršt alata i funkcionalnosti za personaliziranje zajednica uz dobro sučelje za interakciju naših web aplikacija s platformom.

Zbog toga odabir platforme će biti Discord, jer skup klijenata kojima bi funkcionalnosti praktičnog rada bile najkorisnije i interesente primarno koriste platformu Discord, a koje su to funkcionalnosti će se objasniti u poglavlju 5 ovog rada. Bez obzira na to što se odabrala platforma Discord, može uz ovu implementaciju dodati i implementacija za preostale platforme ili za neku treću, ako se iskaže potreba ili interest korisnika.

Traženo svojstvo / Naziv platforme	**Matrix.org**	**Telegram**	**Discord**
Broj mjesečno aktivnih korisnika (mil.)	7,5	900	200
Broj registriranih korisnika koji koriste uslugu (mil.)	115	Nepoznato	560
Javni API i automatizirani računi	✓	✓	✓
Knjižnice za razvoj automatiziranih računa	✓	✓	✓

Tablica 1: Usporedba platformi za slanje istovremenih poruka

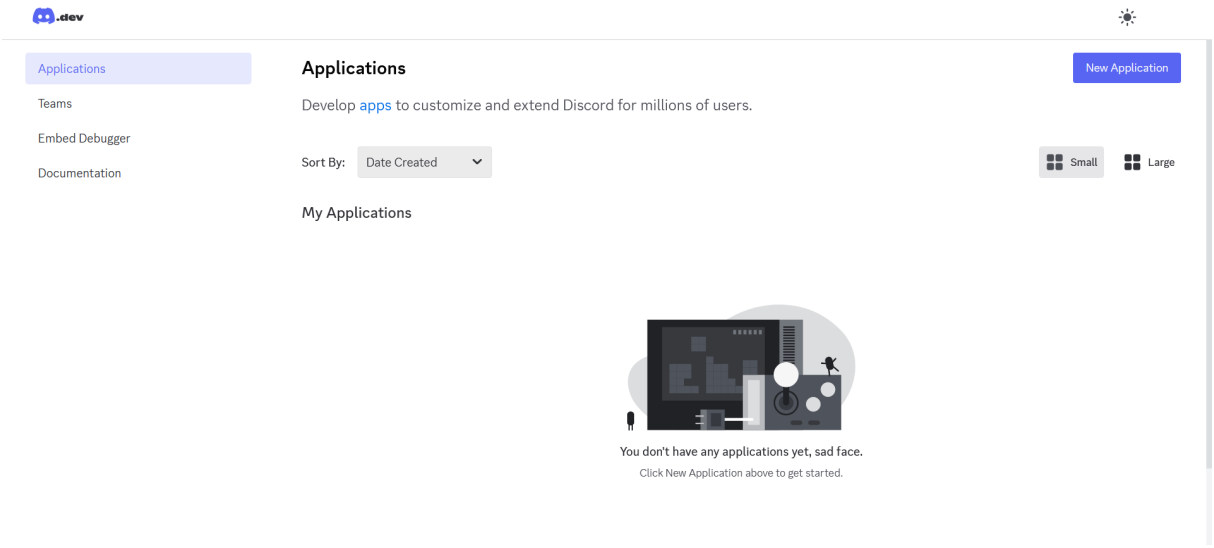
3.2. Kreiranje Discord bota i OAuth2 prijave

Discord API temelji se na dva odvojena API-ja, a to su WebSocket API i Rest API. Websocket API koristi se za primanje događaja s Discorda u stvarnom vremenu. Bot koristi ovaj API za traženje veze, upravljanje glasovnim vezama i obavljanje temeljnih zadataka, dok je Rest API odgovoran za izvođenje radnji u Discordu i upravljanje OAuth2 prijava te drugim funkcionalnostima platforme Discord [10] [11].

Da bi se započela komunikacija vanjske web aplikacije s Discordovom web aplikacijom, Discord zahtjeva od vlasnika vanjske web aplikacije da registrira aplikaciju na Discordovoj stranici. Registracija se odvija na web stranici `https://discord.com/developers`. Na toj stranici Discord nudi dokumentaciju svojeg otvorenog API-ja preko kojeg vanjska web aplikacija može upravljati dodijeljene resurse, te uz to nude mogućnost kreiranje timova s kojima više korisničkih računa mogu imati pristup i uvid o informacijama registriranih aplikacija.

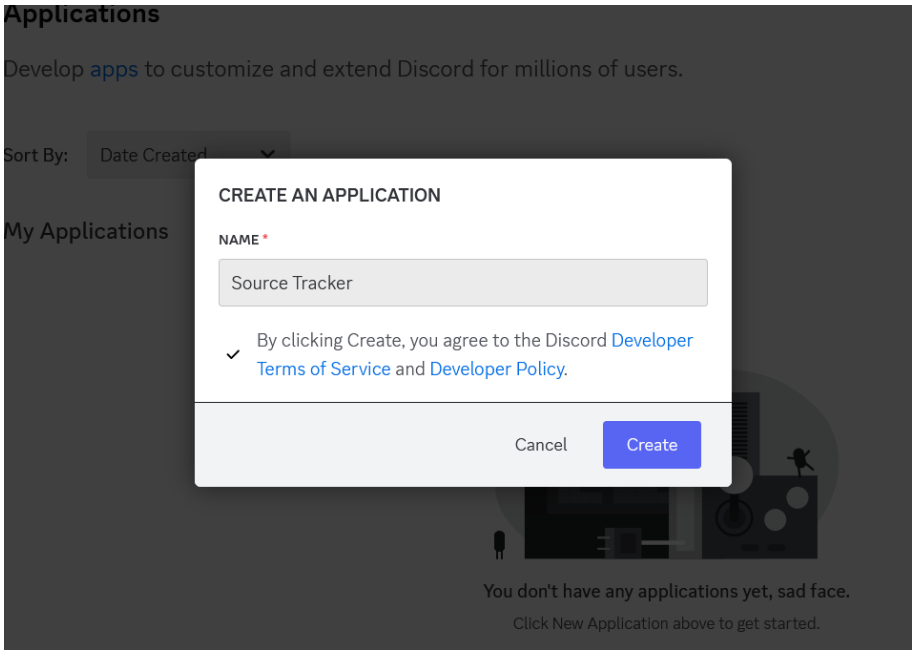
3.2.1. Kreiranje aplikacije u Discordu

Kako bi mogli započeti rad s Discord botom i drugim uslugama koje nudi Discord, potrebno je otvoriti Discord Developer Portal na poveznicu `https://discord.com/developers/applications` i kreirati aplikaciju, tako da se klikne na `New Application` i gore desnom kutu na web stranici.



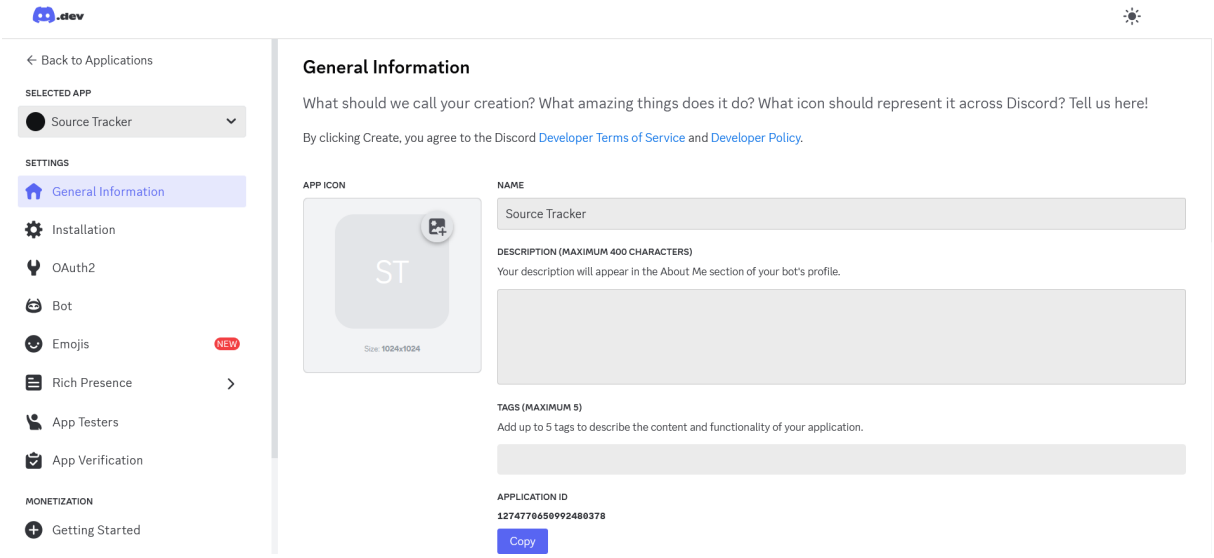
Slika 2: Prikaz sučelja za prikaz i kreiranje web aplikacija [autorski rad]

Nakon toga otvara se skočni prozor koji traži ime aplikacije i prihvaćanje Uvijete korištenja za razvojne programere. Nakon čega je potrebno kliknuti gumb `Create`. Web aplikacija bude se zvala Source Tracker, pa ćemo ju registrirati pod tim nazivom.



Slika 3: Kreiranje aplikacije s imenom Source Tracker [autorski rad]

Nakon toga otvara se stranica `General Information` novo kreirane aplikacije, gdje se mogu mijenjati svojstva aplikacije, kao što su ikona aplikacije, opis aplikacije, URL poveznice za pravila korištenja aplikacije, statistiku o tome u koliko Discord klubova se nalazi bot i koliko korisnika direktno u privatnim porukama koriste bota.



Slika 4: Prikaz generalnih informacija o novoj aplikaciji [autorski rad]

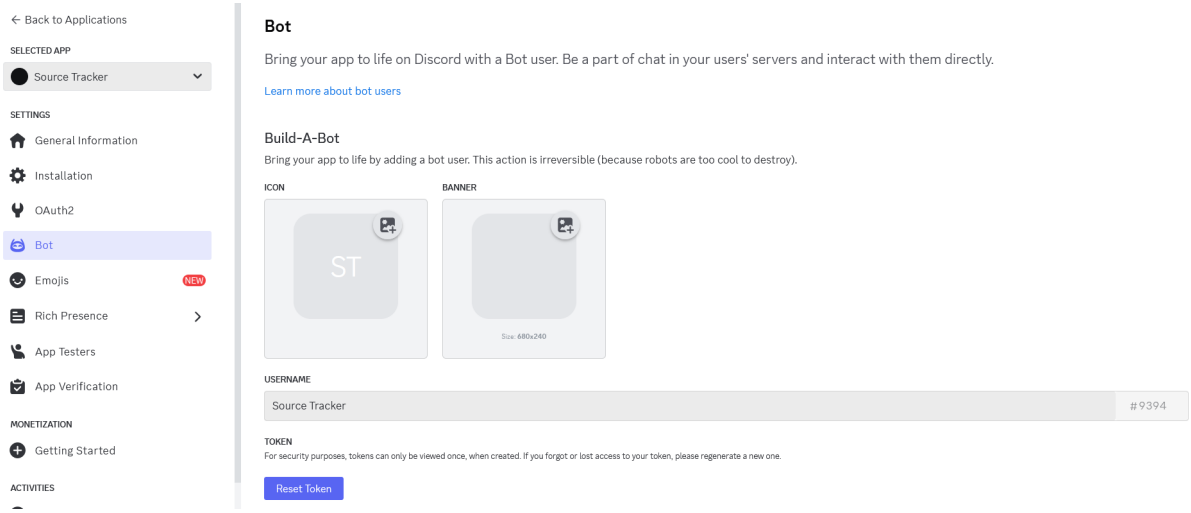
3.2.2. Dohvaćanje Discord tokena za rad s Discord botom

Na lijevoj strani korisničkog sučelja je potrebno kliknuti na `Bot` navigaciju kako bi mogli postaviti Discord bota. Ovo sučelje sada služi za rad nad samim Discord botom, slično se može urediti ime bota, profilna slika koju će imati unutar Discord kluba i slično. Trenutno je potrebno kliknuti `Reset Token` gumb i kopirati ga s pomoću gumba `Copy`.

Token je dugački niz znakova koji predstavlja lozinku (*engl. secret*) s kojom ostvarujemo pristup automatiziranom računu i dopušta izvršavanje svih prava koje Discord bot ima u svim klubovima, zato je važno **ne dijeliti token i pravilno ga zaštiti**, jer ako dođe u ruke malicioznih agenata može prouzročiti golemu štetu, kao što je preuzimanje sadržaja iz klubova koji ne bi trebali biti dostupni svim korisnicima. Najbolji način zaštite varijabli okruženja prema `https://gist.github.com/apple502j/d1330461e7e8ad6532cb62a670d06a5a#dont-store-tokens-in-a-python-file` jest korištenje varijable okruženja (*engl. environment variables*).

U istom dokumentu predstavljeni su neki od čestih sigurnosnih propusta koje mogu dovesti do toga da izgubimo kontrolu nad Discord botom, pa se s time preporučuje da se taj dokument pročita. Primjer Discord tokena se može vidjeti ispod, koji je preuzeti iz Discord službene dokumentacije na poveznici `https://discord.com/developers/docs/reference#authentication`:

```
MTk4NjIyNDgzNDcxOTI1MjQ4.Cl2FMQ.ZnCjm1XVW7vRze4b7Cq4se7kKWs
```



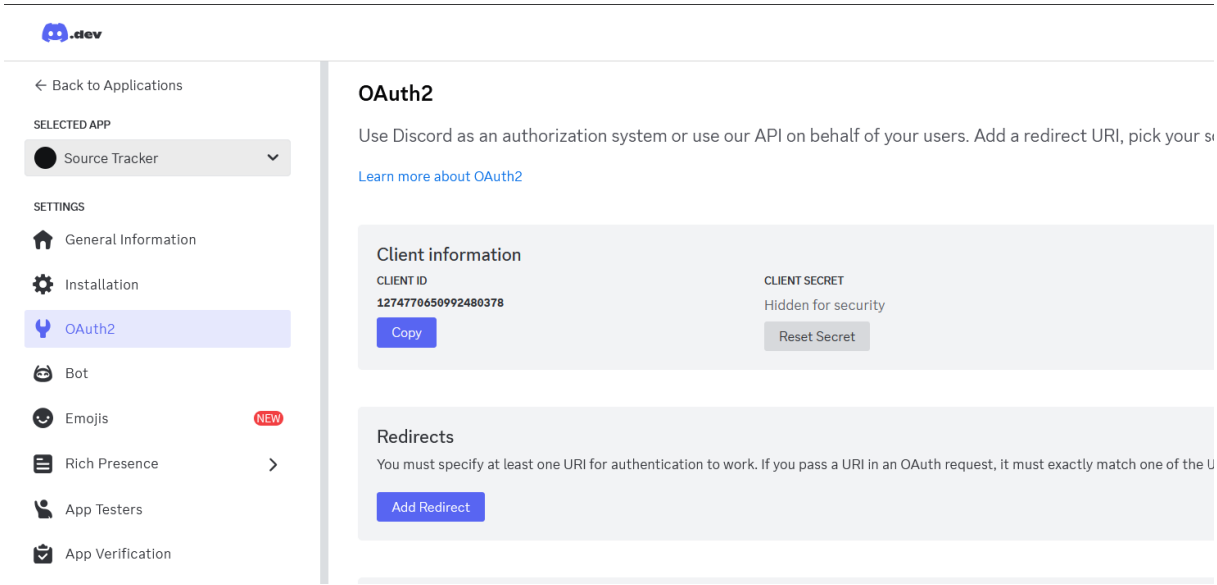
Slika 5: Prikaz dohvata Discord bot tokena [autorski rad]

3.2.3. Dohvaćanje OAuth2

Uz to što smo preuzeli Discord bot token s Discord Developer Portala, za prijavu na web aplikaciju koristit ćemo njihovu uslugu OAuth2 autorizacije kako bi korisnici Discord koji su Discord bota pozvali u svoj klub na olakšani način mogli prijaviti u web aplikaciju.

> OAuth 2.0, što je skraćenica za "otvorenu autorizaciju", standard je osmišljen kako bi web stranici ili aplikaciji omogućio pristup resursima koje pružaju druge web aplikacije u ime korisnika. Zamijenio je OAuth 1.0 2012. i sada je de facto industrijski standard za online autorizaciju. OAuth 2.0 pruža pristup s pristankom i ograničava radnje koje klijentska aplikacija može izvesti na resursima u ime korisnika, bez dijeljenja korisničkih vjerodajnica. [12]

Da bi mogli preuzeti podatke potrebne za osposobljavanja OAuth2 komunikacije iz web aplikacije na Discord, potrebno je navigirati do `OAuth2` sekcije Developer Portala, kao na slici dolje. Tu je potrebno kopirati i na isti način kao i Discord bot token osigurati `CLIENT ID` (*hrv. klijentska identifikacija*) i `CLIENT SECRET` (*klijentska tajna*).



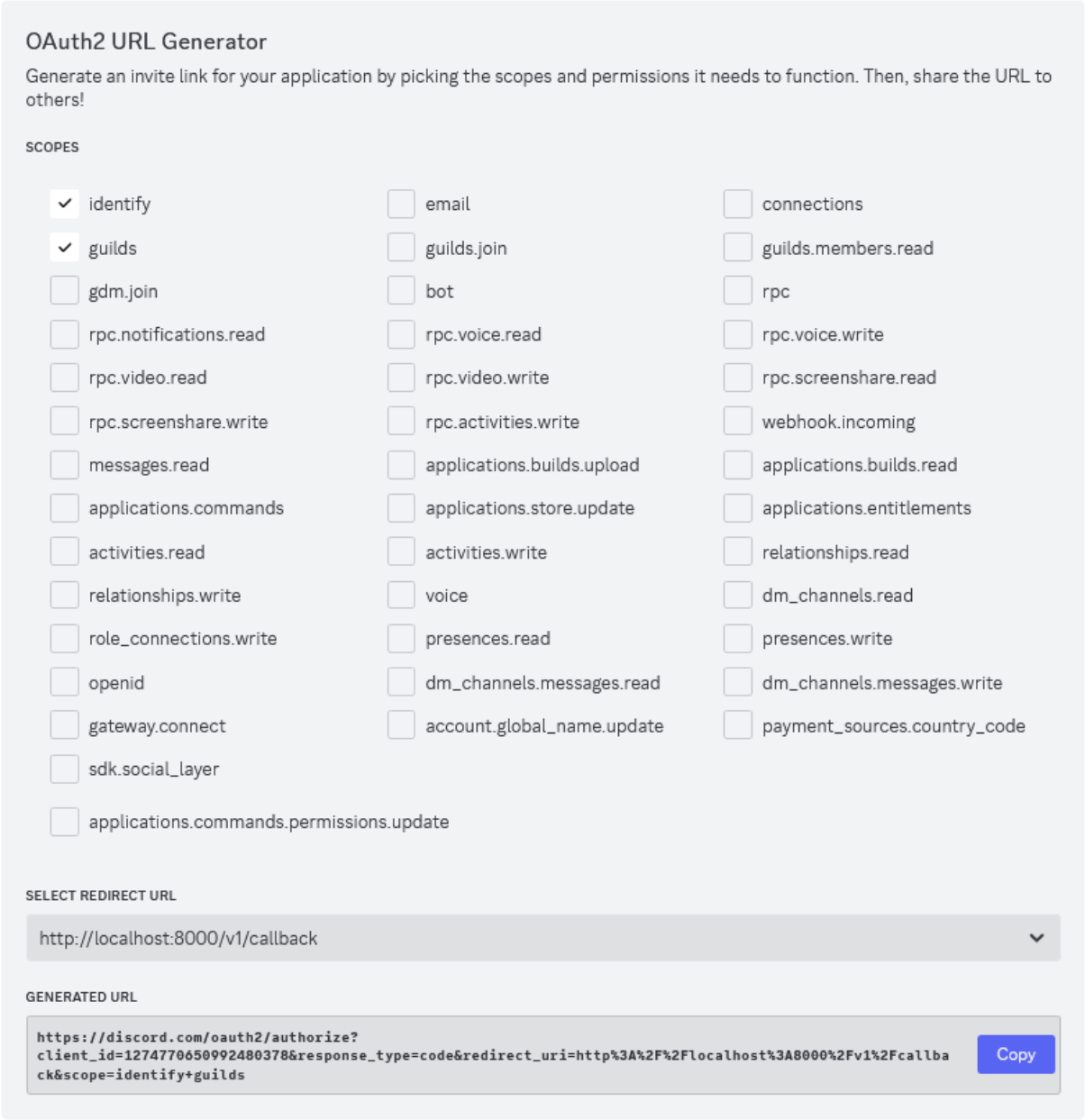
Slika 6: Dohvaćanje klijentske identifikacije i tajne [autorski rad]

Uz to potrebno je napraviti dvije aktivnosti. Prva aktivnost je dati Discordu Redirect poveznicu na koju dobiva web aplikacija verifikacijske podatke da je korisnik koji se je prijavio zaista taj korisnik. Klikom na `Add Redirect` dodajemo barem jedan takav URL, s obzirom na to da se web aplikacija pokreće na lokalno i samo imamo v1 verziju API-ja, dodajemo putanju `http://localhost:8000/v1/callback`.

Druga aktivnost jest generiranje OAuth2 URL-a preko kojeg korisnik se može prijaviti na web aplikaciju. Na istoj stranici `OAuth2` ispod `Add Redirect` nalazi se `OAuth2 URL Generator`. Ovdje je potrebno odabrati djelokruge (*engl. scopes*) koji od korisnika kada otvori poveznicu traže određene dozvole od korisničkog računa. Za web aplikaciju je potrebno

odabrati `identify` (*hrv. identifikacija*) i `guilds` (*hrv. klubovi*). Dozvola `identify` dopušta vanjskoj web aplikaciji da dobi informacije o korisničkome računu korisnika, dok `guilds` dopušta vanjskoj web aplikaciji da vidi sve klubove u kojima se nalazi Discord korisnik. Nakon toga, u `SELECT REDIRECT URL` odabrati putanju koju smo prethodno dodali u `Redirect`. S time je generirani URL i može se kopirati poveznica i tretirati je kao i prethodne tajne.

Generirani URL je putanja preko koje vanjska web aplikacija provjerava ako su informacije poslane na `REDIRECT URL`, ako bi se izgubio ovaj URL od strane nekog napadača, napadač može lažno predstaviti kao vanjska web aplikacija žrtve i naštetiti žrtvi.



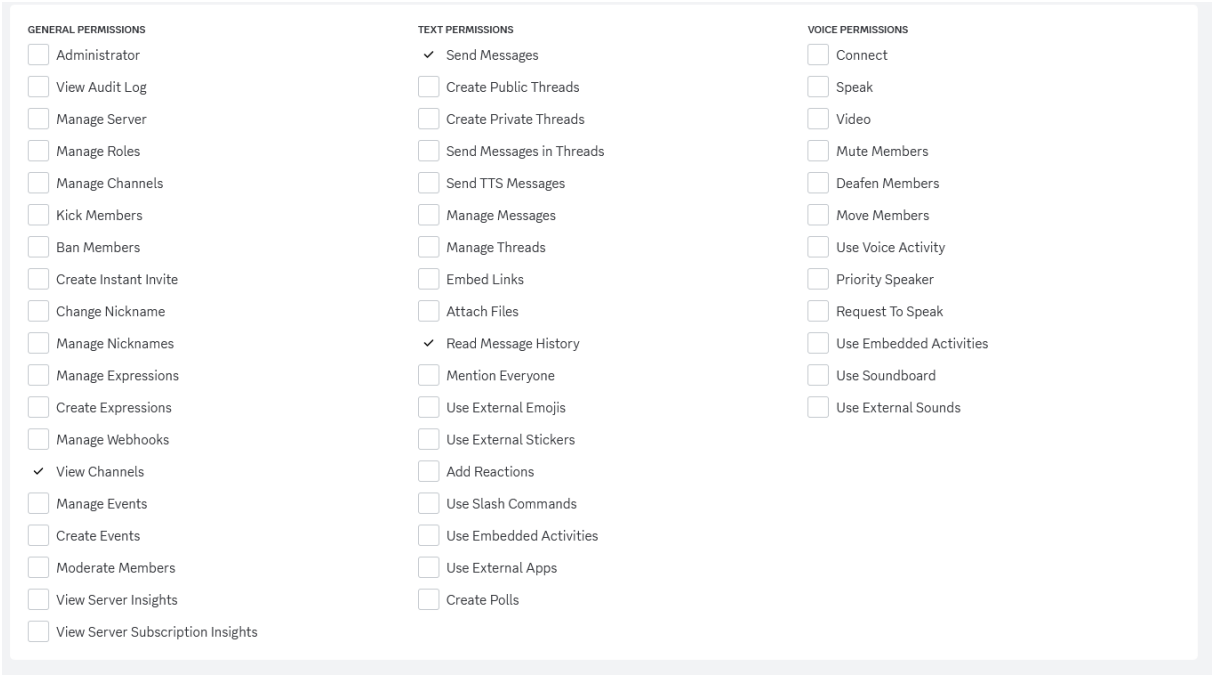
Slika 7: Generiranje OAuth2

3.2.4. Generiranje poveznice za pozivanje Discord bota u klub

Uz prethodne tajne koje su generirane i sigurno spremljene, potrebno generirati javnu poveznicu preko koje Discord korisnik može pozvati Discord bota u klubove gdje su oni vlasnici. Slično kao i za OAuth2 poveznicu, u istom sučelju potrebno je odabrati `bot` i dozvolu za instalaciju naredbi u Discord klubovima, `applications.commands`. Nakon toga potrebno je spustiti prozor web stranice do dna i odabrati sljedeće dozvole.

- *View Channels*
- *Send Messages*
- *Read Message History*

Nakon što se odaberu dozvole koje će Discord bot zatražiti od vlasnika kluba prilikom poziva bota, potrebno je kopirati poveznicu. Iako je poveznica namijenjena da se dijeli s osobama koje će koristiti Discord bot, bit će stavljena u varijablu okruženja zbog lakšeg rada s poveznicom.



Slika 8: Prikaz odabira dopuštenja za Discord bot-a [autorski rad]

4. Tehnologije za razvoj sustava web aplikacija

U današnjem ekosustavu gotovo svi programski jezici za generalnu namjenu podržavaju neki oblik komunikacije s pomoću interneta. Da bi se olakšalo interakcija s internetom, mnogi od njih imaju ili ugrađene ili su proširene s bibliotekama implementacije sustava za interakciju s internetom. U ovom radu pokazat će se kako u programskom jeziku Python s ugrađenim implementacijom za komunikaciju preko interneta se može lagano stvoriti primitivni oblik web aplikacije, kako taj pristup možda nije povoljan te ćemo prikazati ekstenziju pod nazivom FastAPI koja će nam olakšati cijeli proces na moderan način.

Razvoj web aplikacija možemo analogno usporediti s koncertnom predstavom. Uz prethodno rigorozno planiranje i donošenje odluka treba uključiti skup tehnologije koja će omogućiti svim sudionicima koncerta gotovo isto iskustvo. Na primjer, sigurno postoji velika razlika što je potrebno da bi se koncert održao na otvorenome ili u dobro zvučno izoliranoj koncertnoj dvorani da bi se moglo jasno čuti što se govori na pozornici na cijelog površini prostora publike. Pa tako je potrebno se odlučiti se za stog tehnologija (*engl. tech stack*) s kojim će se ovaj projekt realizirati. Ovo je još jedna kritična točka u procesu razvoja softvera gdje loša odluka može dovesti to troškova kasnije, bilo to monetarno ili vremenski trošak za popravak ako se dobro ne promisli.

Često je teško se odlučiti za pravi set tehnologija jer predviđanje potreba projekta unaprijed je iznimno težak zadatak jer je teško predvidjeti buduće potrebe softvera u nekom dužem periodu, zato što zahtjeva ili dovoljno prethodnog iskustva kreiranja sličnih sustava ili ulaganje dodatnog vremena prikupljanjem informacija da bi se moglo preciznije predvidjeti. S obzirom na to da rješavanje ovog zadataka nije jednokratno i postoji mogućnost da će biti potrebno donijeti nove odluke o uvođenju ili uklanjanju tehnologije u stog kako se napreduje s projektom, bitno je samo odabrati dobar bazični stog tehnologija za koje smo pouzdani da imaju najmanju vjerojatnost zamjene tijekom razvoja projekta.

To primarno uključuje programski jezik u kojem se razvija web aplikacija, baza podataka nad kojom web aplikacija izvršava podatkovne transakcije ili tehnologije razvoja korisničkog sučelja, kao što su React, Angular i slične JavaScript knjižnice [13]. U teoriji stog tehnologija može se samo sastojati od jezika u kojem je web server napisan, ali takvi scenariji su rijetki jer jedina opcija za trajnu pohranu podataka je korištenje datotečnog sustava uređaja na kojem se nalaze, što nije skalabilno rješenje kako sustav raste. Stog tehnologija može se odabrati po specifikaciji projekta i procjeni razvojnog tima, a inspiracije za stog se može pronaći od poduzeća koja ih aktivno koriste na stranicama kao što su `https://stackshare.io/stacks/trending` i `https://himalayas.app/companies`.

Za ovaj rad pogledane su obje stranice za inspiraciju tehnološkog stoga, uz njih pogledano je *TIOBE Programming Community Index* koji su trenutno najpopularniji programski jezici u trenutku pisanja ovog rada. Zato je za stog tehnologija od kojeg će biti sastavljen od baze podataka SQLite3, programskog jezika Python i biblioteka asociranih s ovim jezikom.

Prema govoru Dan Callahana na konferenciji PyCon 2018 - "Python is the second best language for anything", odnosno poznavanjem Python programskog jezika može se koristiti kao polaznu točku u veliki spektar programskih domena, ne samo kao web aplikacija nego i alat za analizu podataka, stvaranje jednokratnih skripti za posebnu namjenu, a danas kao alat za rad s umjetnom inteligencijom. Sumu njegovog bloka na konferenciji i video se može naći na poveznici `https://github.com/ikding/pycon_2018`.

4.1. Programski jezik Python

Python je interpretirani viši programski jezik opće namjene. Pythonova filozofija dizajna naglašava čitljivost koda uz značajnu upotrebu uvlačenja kao dio semantike jezika. Njegove jezične konstrukcije kao i objektno orijentirani pristup imaju za cilj pomoći programerima da napišu jasan, logičan kod za male i velike projekte [14].

Guido van Rossum počeo je raditi na Pythonu kasnih 1980-ih, kao nasljedniku programskog jezika ABC, a prvi put ga je objavio 1991. kao Python 0.9.0. Python 2.0 objavljen je 2000. godine i uveo je nove značajke, kao što su komprehenzija lista i sustav skupljanja smeća koji koristi brojanje referenci. Python 3.0 objavljen je 2008. i bila je velika revizija jezika koja nije u potpunosti kompatibilna s prethodnim verzijama i mnogo Python 2 koda ne radi neizmijenjeno na Pythonu 3. Python 2 je ukinut s verzijom 2.7.18 i potpora za jezik prestaje 2020. godine. [14]. Službena dokumentacija Python v3 verzije može se pronaći na poveznici `https://docs.python.org/3/`.

Python programski jezik cilja da programski kod bude što bliži ljudskome tekstu da bi se ubrzao proces razvoja softvera. Koristi ključne riječi koje su pretežito kratke riječi iz engleskog jezika, izbjegava pretjerano korištenje znakova ili posebnih znakova koji bi smanjili čitljivost koda. Temeljna filozofija jezika sažeta je u dokumentu pod nazivom *The Zen of Python* (PEP 20) [14], koji uključuje aforizme kao što su:

Lijepo je bolje nego ružno.
Eksplicitno je bolje od implicitnog.
Jednostavno je bolje od složenog.
Složeno je bolje nego komplicirano.
Čitljivost se računa.

Implementacija Python programskog jezika je platformski nezavisna, što znači da programski kod napisan na platformi Windows, ako se ne koriste knjižnice specifično za rad s Windows API-jem, mogu se pokrenuti na Linux/Unix i MacOS platformama te ponašanje programa bi trebalo ostati isto, što olakšava razvoj na jednoj platformi, a ciljana platforma jest neka druga, kao što su web poslužitelji koji koriste Linux/Unix kao operacijski sustav.

4.2. Karakteristike Python jezika

Python spada u skupinu interpretiranih programskih jezika, to znači da programski kod napisan za programski jezik Python se tumači tek kada se pokrene interpreter koji onda tumači sadržaj unutar programskog koda s kojim je pokrenuti interpreter. To omogućava da se programski kod može dinamički tijekom svojeg rada može evaluirati i generirati.

Python podržava nekoliko programskih paradigmi, od kojih dominantne jest objektno orijentirano programiranje i strukturirano programiranje, a mnoge njegove značajke podržavaju funkcijsko programiranje i aspektno orijentirano programiranje (uključujući metaprogramiranje i metaobjekte). Mnoge druge paradigme podržane su putem proširenja, uključujući projektiranje prema ugovoru i logičko programiranje [14].

Umjesto da ima svu svoju funkcionalnost ugrađenu u jezgru, Python je dizajniran da bude izuzetno proširiv (s modulima). Ova kompaktna modularnost učinila ga je posebno popularnim kao sredstvo za dodavanje programabilnih sučelja postojećim aplikacijama [14], kao što su alati kao Blender [15], Unreal Engine [16] i mnogi druge aplikacije koje podržavaju sličan oblik ekstenzije.

4.2.1. Semantika i sintaksa programskog jezika Python

Kao što je prije navedeno Python programski jezik koristi uvlačenja umjesto vitičastih zagrada ili ključnih riječi kako bi se stvorila granica između dvaju blokova. Povećanje uvlake koje dolazi nakon određenih izjava označava gniježđenje bloka koda unutar postojećeg bloka, dok smanjenje uvlake označava kraj ugniježđenog bloka. Tako prilikom definiranje funkcije s riječju `def`, korištenjem petlji kao što su `for` i `while` ili posebne ključne riječi `with` stvaramo nove programske blokove.

```
def print_numbers(n: int):
    for i in range(0, n):
        print("Trenutni broj:")
        print(i)
    print("Kraj ispisa")

print_numbers(3) — Poziv funkcije
```

Programski kod gore je funkcija s imenom `print_numbers` koja prima jedan argument `n` tipa `int`, odnosno prima cjelobrojnu vrijednost. Vrijednost `n` daje se generatorskoj funkciji `range` koja generira niz vrijednosti od `0` do `n-1` cjelobrojnih vrijednosti te za svaku od njih sprema vrijednost u `i` varijablu. S pomoću ugrađene funkcije `print` ona se ispisuje, a prethodi ju tekst `Trenutni broj:`. Kada se prođe kroz sve vrijednosti ispisuje se `Kraj ispisa`. Tako kada se funkcija pozove s `n` koji ima vrijednost 3, dobivamo sljedeći ispis:

```
Trenutni broj:
0
Trenutni broj:
1
Trenutni broj:
2
Kraj ispisa
```

Varijable u Python programskome jeziku su imenovani dijelovi memorije koji spremaju neku vrijednost s određenim značenjem. Za deklaraciju varijabli u programskom jeziku Python potrebno je samo dati unikatno ime varijabli i s operatorom dodjele (=) dodijeliti neku vrijednost. O mogućim vrijednostima će se diskutirati u poglavlju 4.2.2.

```
def square_sum(n: int):
    sum = 0
    for i in range(1, n):
        sum = sum + (i * i)
    return sum

n = 3
print("Suma prvih", n, "cjelobrojnih brojeva je", square_sum(n))
```

Rezultat ovog isječka koda je:

```
Suma prvih 3 cjelobrojnih brojeva je 14
```

Uz gore navedene primjere ključnih riječi, Python prepoznaje skup od 35 ključnih riječi s trenutnom verzijom 3.12 [17]. Preporučuje se da se upozna s uporabom ovih ključnih riječi jer se koriste gotovo sve ključne riječi u složenijim programima od prijašnjih 2 primjera. Jedan od dobrih resursa za učenje Python programskog jezika je web stranica W3Schools na poveznici `https://www.w3schools.com/python/default.asp`.

Ključne riječi u Python verziji 3.12			
False	class	from	or
None	continue	global	pass
True	def	if	raise
and	del	import	return
as	elif	in	try
assert	else	is	while
async	except	lambda	with
await	finally	nonlocal	yield
break	for	not	

Jedne od važnih ključnih riječi potrebne za izradu završnog rada su ključne riječi `async` i `await`. Ove ključne riječi služe rad s asinkronim programiranjem na kojem će biti bazirani završni rad. Asinkrono programiranje je tehnika programu omogućuje pokretanje potencijalno dugotrajnog zadatka i uz to može reagirati na druge događaje dok se taj zadatak izvodi, umjesto da mora čekati dok taj zadatak ne završi. Nakon što je zadatak završen, program prikazuje rezultat dugotrajne operacije [18].

U programskom jeziku Python, za asinkrone zadatke koristimo funkcije definirane s ključnim riječima `async def`, a za obradu asinkrone funkcije stavlja se ključna riječ `await` ispred poziva funkcije. Asinkroni zadaci često ne ponašaju se kako su napisani u programskome kodu i ne odvijaju se slijedno. Najčešća uporaba asinkronih funkcija je za dohvaćanje podataka iz baze podataka, komunikacija s API-jem i zahtjevni izračuni funkcija. Primjer dolje demonstrira kako pomoću biblioteke `asyncio` se pokreću asinkroni programi.

```
import asyncio

async def main():
    await asyncio.gather(
        dohvati_podatke(),
        racunaj(),
        posalji_na_api(),
    )
    print("Zadaci gotovi")

asyncio.run(main())
```

4.2.2. Tipovi podataka

Python koristi način tipiziranja varijabli pod nazivom "duck typing". Duck typing u računalnom programiranju primjena je testa patke: "Ako hoda kao patka i kvoca kao patka, onda mora biti patka", kako bi se utvrdilo može li se objekt koristiti za određenu svrhu. Kod normalnog tipiziranja varijabli, prikladnost određenih operacijama nad varijablom je određena tipom objekta koji je inicijalno dodijeljen varijabli. Kod duck typing-a, prikladnost objekta određena je prisutnošću određenih metoda i svojstava, a ne vrstom samog objekta [19].

Python objekti su uvijek tipizirani, ali dopušteno je da se varijabla deklarira bez dodijeljenog tipa, jer Python interpreter može zaključiti tip podatka prema vrijednosti koja se dodjeljuje toj varijabli. Ograničenja nad tipom varijabli se ne provjeravaju za vrijeme prevođenja programskog koda. Unatoč tome što je dinamički tipiziran (*engl. dynamically-typed*), Python je strogo tipiziran (*engl. strongly-typed*), što znači da zabranjuje operacije koje nisu dobro definirane (na primjer, zbrajanje numeričke vrijednosti s nizom znakova).

Python podržava sve standardne tipove podataka, kao što su brojevi (`bool, int, float i complex`) i nizove. Uz to ima implementacije važnih podatkovnih tipova kao što su skupovi (*engl. set*) i rječnici (*engl. dictionary*). Uz to, omogućava da se varijabli dodjele funkcije kao vrijednosti, pod nazivom `Callable`, što je jedan od preduvjeta za podržavanje funkcijskog programiranja. O standardnim tipovima podataka može se više pročitati na poveznici: `https://docs.python.org/3/library/stdtypes.html`.

Uz standardne tipove podataka Python omogućuje programerima da definiraju vlastite tipove s pomoću klasa. Primjer ispod je način kako se definira tip i tipizira varijable u programskom jeziku Python:

```
class MojTipPodataka:
    cjelobrojna_varijabla: int
    znakovni_niz: str

moja_varijabla: MojTipPodataka = MojTipPodataka(
    cjelobrojna_varijabla = 3,
    znakovni_niz="Pozdrav svijete!",
)

print(moja_varijabla.znakovni_niz)
```

4.3. Instalacija Python jezika i potrebnih ovisnosti

Programski jezik Python se može instalirati za platforme Windows, Linux/Unix i MacOS putem poveznice `https://www.python.org/downloads/`. Instalacija za Windows i MacOS se odvija u instalacijskom programu te je potrebno pratiti upute koje nudi aplikacija, dok za Linux/Unix može se koristiti ili arhive na službenoj stranici ili s pomoću naredbene linije (u primjeru je za Ubuntu i srodne inačice):

```
$ sudo apt-get update
$ sudo apt-get install python3
```

Nakon instalacije, za provjeru dali se Python pravilno instalirao na svim platformama se može izvršiti sljedeća naredba da se prikaže trenutno instalirana verzija Python interpretera:

```
$ python3 --version
```

Nakon što smo uspješno instalirali Python, preporučljivo je kreirati lokalno razvojno okruženje za Python u direktoriju po želji pod nazivom virtualno okruženje (*engl. virtual environment*). Virtualno okruženje je analogno `node_modules` direktoriju koji se koristi u Node.js projektima, izolira instalaciju paketa Python knjižnica od drugih postojećih i budućih okruženja te može se instalirati za više verzija Pythona. Da bi se kreiralo virtualno okruženje, potrebno ga je kreirati izvršavanjem naredbe [20]:

```
$ python -m venv /putanja/do/virtualnog/okruženja/naziv_okruženja
```

Gdje `/putanja/do/novog/virtualnog/okruženja/` je direktorij u kojem želimo stvoriti okruženje, a `naziv_okruženja` je naziv direktorija koji će sadržati samo okruženje. Standardna praksa je dodijeliti ime okruženju "venv" [20]. U daljnjem tekst za direktorij u kojem se nalazi virtualno okruženje ćemo koristiti će se naziv .

Sada je potrebno aktivirati virtualno okruženje kako bi promjene o instalaciji paketa bile spremljene u okruženju umjesto da se spremaju globalno, potencijalno s konfliktima s drugima Python skriptama. U tablici dolje navedene su naredbe koje pokrivaju standardne programe naredbene linije za POSIX i Windows sustave. Nakon što se izvede naredba može se koristiti naredbe poput `pip` i `timeit`, koje će samo raditi unutar virtualnog okruženja.

Razvojna okruženja kao što su Visual Studio Code [21] i PyCharm [22] mogu kreirati virtualno okruženje za korisnika, te pokrenuti ga kada korisnik se nalazi u projektu s kreiranim virtualnim okruženjem.

Platforma	Shell	Naredba za aktivaciju virtualnog okruženja
POSIX	bash/zsh	`$ source <venv>/bin/activate`
	fish	`$ source <venv>/bin/activate.fish`
	csh/tcsh	`$ source <venv>/bin/activate.csh`
	PowerShell	`$ <venv>/bin/Activate.ps1`
Windows	cmd.exe	`C: <venv>\Scripts \activate.bat`
	PowerShell	`PS C: <venv>\Scripts\Activate.ps1`

Tablica 2: Naredbe za pokretanje virtualnog okruženja (Python.org, bez dat.)

Python tretira ovisnosti o paketima kao module. Moduli su Python projekti koje možemo uvoziti u program tako da koristimo ključnu riječ `import` i ime direktorija. Također korisnik može kreirati module tako što stvori direktorij koji sadrže datoteku s imenom `__init__.py`. Iako to više nije nužno jer Python može implicitno zaključiti da se radi o modulu [23]. Naspram tome `__init__.py` dopušta dodatne funkcionalnosti kao što je olakšavanje korištenja `import` ključne riječi, skraćujući putanje do ugniježđenih datoteka, definiranje konstantnih vrijednosti i sličnih funkcionalnosti.

```
projekt
├── venv/
├── moj_modul/
│   ├── __init__.py
│   └── moja_skripta.py
└── main.py
```

Sada u `main.py` možemo napisati `import moj_modul.moja_skripta` da bi imali pristup funkcijama, varijablama i klasama unutar te skripte. Kada dodamo liniju koda u datoteku `__init__.py`: `from .moja_skripta import *`, u `main.py` možemo pristupiti `moja_skripta` s linijom koda `import moj_modul` i `main.py` će imati pristup svom sadržaju iz `moja_skripta`.

Na sličan princip rade i paketi. Da bi se dohvatio paket, on se može preuzeti s pred instaliranim alatom `pip` da bi se dohvatio sa službenog repozitorija paketa. Za primjer, instalacija popularnog paketa numpy se može ostvariti sljedećom naredbom:

```
$ python3 -m pip install "numpy"
```

ili

```
$ pip install numpy
```

Ova naredba dohvaća ciljani paket te pakete o kojima ovisi ciljani paket. Kako bi se olakšala prenosivost Python projekta, može se kreirati `requirements.txt` datoteka u kojoj se navode svi potrebni paketi, koji se mogu prosjediti i automatski preuzeti s naredbom s pomoću zastavice `-r`:

```
python3 -m pip install -r requirements.txt
```

Sadržaj `requirements.txt` formatiran je tako da se željeni paket stavi u zaseban red. Ako je potrebna specifična verzija paketa, potrebno je formatirati red u oblik `==`. Primjer sadržaja datoteke `requirements.txt` prikazan je ispod:

```
# requirements.txt
------------------
ezcord==0.6.5
fastapi==0.112.0
httpx==0.27.0
jinja2==3.1.4
py-cord==2.6.0
pytest==8.3.2
SQLAlchemy==2.0.32
testcord==0.0.1
uvicorn==0.30.6
```

4.4. Opis glavnih ovisnosti

Da bi se u Pythonu kreirao web aplikacija iz nule, potrebno je koristiti paket `socket`. Dolje primjer pokazuje kako bi izgledala web aplikacija koja klijentu vraća znakovni niz "Pozdrav svijete!" na zahtjev. Može se provjeriti ako se veza s web aplikacijom uspostavila tako da se pročita dekodirana `request` varijabla.

```
import socket

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server_socket.bind(('0.0.0.0', 8000))
server_socket.listen(1)
print(f'Server pokrenut na portu: 8000')

while True:
    client, _ = server_socket.accept()
    request = client.recv(1024).decode()
    print(request)

    response = 'HTTP/1.0 200 OK\n\nPozdrav svijete!'
    client.sendall(response.encode())
    client.close()

server_socket.close()
```

U pravilu ovo je dovoljno za napraviti web aplikaciju koja poslužuje statični sadržaj kao što je tekst, Hypertext Markup Language (dalje u tekstu HTML) ili prikaz slika. Za složeniji sadržaj ovaj pristup brzo postaje nepraktičan za korištenje jer zahtjeva puno vremena za izradu samog pozadinskog programskog koda koji održava web aplikaciju. Uz to svaki zahtjev se obrađuje slijedno, što znači da web aplikacija može posluživati samo jednog klijenta, dok drugi klijent potencijalnog nikada neće doći do sadržaja ako je promet prema stranici veći od brzine obrade zahtjeva.

Kao riješene tome, postoje paketi koji implementiraju specifikaciju *Web Server Gateway Interface* (dalje u tekstu WSGI). WSGI paketi smanjuju potrebu za prethodnim programskim kodom, kao što je paket WSGIServer. S njima web aplikacija se može smanjiti na minimalnu implementaciju kojoj je potrebna samo poslovna logika. Uz to postoji varijanta *Asynchronous Server Gateway Interface* (dalje u tekstu ASGI) specifično za asinkrone web aplikacije, što omogućava obradu više klijentskih istovremeno, ali ne nužno paralelno. Primjeri takvih paketa su Hypercorn, Uvicorn i Gunicorn. Primjer ispod prikazuje kako prethodni programski kod izgleda kada se koristi WSGIserver:

```
    import wsgiserver

    def moja_aplikacija(environ, start_response):
        start_response('200 OK', [('Content-Type', 'text/plain')])
        return [b'Pozdrav svijete!']

    server = wsgiserver.WSGIServer(moja_aplikacija)
    server.start()
```

Programski kod implementiran s WSGIServer paketom puno pregledniji od primjera s `socket` standardnim paketom, ali još uvijek postoji dužnost razvojnog programera da obrađuje informacije kao što su putanje iz *environ* varijable, kao u primjeru ispod:

```
def moja_aplikacija(environ, start_response):
    path = environ.get('PATH_INFO', '/')

    if path == '/world':
        status = '200 OK'
        response_body = 'Pozdrav svijete'
    elif path == '/you':
        status = '200 OK'
        response_body = 'Pozdrav čitatelju'
    else:
        status = '404 Not Found'
        response_body = 'Page not found'

    response_body = response_body.encode('utf-8')

    headers = [('Content-Type', 'text/plain; charset=utf-8'),
            ('Content-Length', str(len(response_body)))]

    start_response(status, headers)

    return [response_body]
```

Zbog toga postoje paketi kao što su Django, Flask i FastAPI koji pružaju okvir(*engl. framework*) koji olakšava izradu web aplikacija tako što apstrahira programski kod web aplikacije i dozvoljava razvojnom programeru veći fokus na poslovnu logiku web aplikacije i drastično ubrzavaju razvoj web aplikacije. Uz to mnogi okviri su proširivi s paketima koji dopunjavaju i unaprjeđuju njihove kvalitete.

4.4.1. FastAPI, Jinja2 i Uvicorn

Kada se koristi FastAPI i Uvicorn da se implementira web aplikacija kao u prethodnom primjeru, programski kod izgleda kao sljedeći:

```
from fastapi import FastAPI
import uvicorn

app = FastAPI()

@app.get("/world")
async def get_world():
    return "Pozdrav svijete"

@app.get("/you")
async def get_you():
    return "Pozdrav čitatelju"

if __name__ == "__main__":
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

Uz to što FastAPI čini programskim lakšim za navigirati, kao što je u imenu paketa, je jedan od najbržih okvira za razvoj web aplikacija [24], brži od popularnih Django i Flask paketa. Također nudi funkcionalnosti kao modularnost API ruta pomoću klase APIRouter, automatsko generiranje otvorenih standarda za API-je OpenAPI i JSON Schema [25]. Uz to podržava Jinja2, paket za generiranja sadržaja iz predložaka, koji se primarno koristi za umetanje Python varijabli u HTML sadržaj. Ispod je primjer APIRoutera i Jinja2 programskog koda:

```
# index.html
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Moja stranica</title>
</head>
<body>
    <h1>Web aplikacija kaže: {{ vrijednost }}!</h1>
</body>
</html>
```

```
# main.py
from random import choice
...
from fastapi.templating import Jinja2Templates

app = FastAPI()
router = APIRouter()
templates = Jinja2Templates(directory="templates")
varijable = ["Pozdrav svijete!", "Pozdrav tebi!", "Pozdrav Jinja2"]


@router.get("/", response_class=HTMLResponse)
async def read_homepage(request: Request):
    return templates.TemplateResponse(
        "index.html",
        {
            "request": request,
            "vrijednost": choice(varijable),
        },
    )

app.include_router(router)

if __name__ == "__main__":
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

4.4.2. Pycord i ezcord

Pycord paket baziran na prethodniku discord.py koji služi kao API omotač (*engl. API wrapper*) za Discord API. Olakšava izradu programske logike Discord bota, uz to podržava asinkrono programiranje. Paket ezcord je paket koji proširuje mogućnosti Pycord paketa.

Pycord podržava sve funkcionalnosti koje pruža Discord, kao što su naredbe kose crte (*engl. slash commands*), rad s modelima podataka koje koristi Discord, funkcionalnosti Discorda slanje poruka, kreiranje događaja u klubovima, snimanje audio poziva i slično.

Omogućava modularnost preko ekstenzija pod imenom zupčanika *engl. cogs* kako bi se poslovna logika Discord bota podijelila na više dokumenta.

```
import discord

bot = commands.Bot()

@bot.event
async def on_ready():
    print(f'Prijavljen kao {bot.user.name}')

@bot.slash_command()
async def hello(ctx):
    await ctx.respond("Pozdrav!")

bot.run('UNESI_BOT_TOKEN')
```

4.4.3. python-a2s

Paket python-a2s skup alata za slanje upita Source i GoldSource poslužiteljima video igara. Implementira A2S protokol dizajniran od poduzeća Valve. Podržava slanje sinkronih i asinkronih upita poslužiteljima. A2S protokol bazira se na UDP/IP protokolima [26] i detaljna specifikacija protokola može se vidjeti na poveznici `https://developer.valvesoftware.com/wiki/Server_queries`.

Paket nudi 3 funkcije u sinkronim: `info` vraća informacije o kakvom se Source ili GoldSource poslužitelju radi i detaljne informacije o poslužitelju, `players` vraća popis igrača s njihovim podacima i `rules` vraća pravila poslužitelja specifična za videoigru koja je pokrenuta na poslužitelju. Asinkrona verzije ovih funkcija imaju prefix `a`, pa tako su asinkrone varijante su `ainfo`, `aplayers` i `arules`.

Svaka od ovih funkcija prima 3 parametra: `address: Tuple[str, int]` par vrijednosti koji predstavljaju IP adresu ili DNS adresu poslužitelja i broj porta, `timeout: float` broj sekundi koliko upit čeka odgovor od poslužitelja i `encoding: str | None` vrsta kodiranja niza znakova.

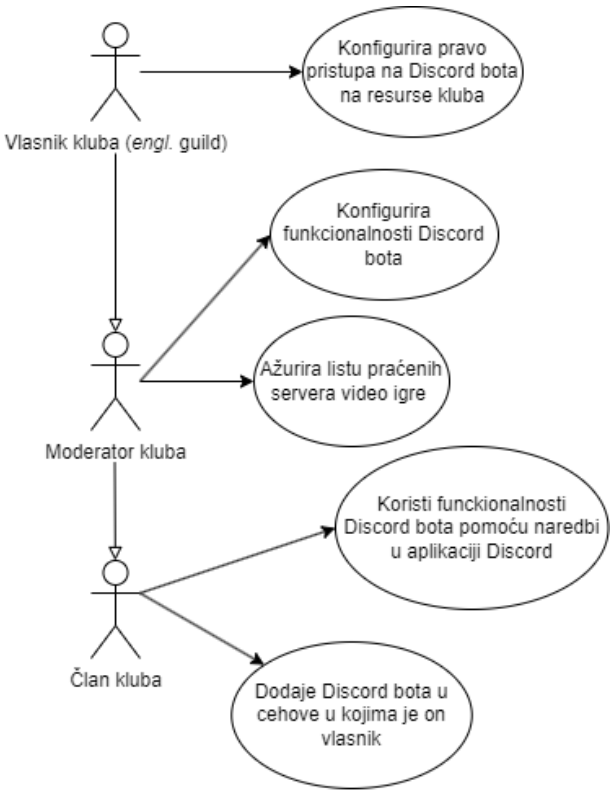
5. Izrada web aplikacije

5.1. Kratki opis web aplikacije

Web aplikacija će periodično prikupljati podatke o broju igrača u pojedinom Source engine serveru videoigre i prikazivati ih u klubu korisnika koji je dodao Discord bota. Web aplikacija će upravljati automatiziranim računom platforme Discord koji će slati obavijesti o tome koliko je trenutno igrača na Source serveru videoigre, prikaz igrača koji se trenutno nalaze u pojedinim Source Engine serverima i druge funkcionalnosti. Aplikacija će također pružati upravljačku ploču za upravljanje postavkama ponašanja Discord bota nad svakim klubom koji prijavljeni korisnik nadgleda, te ako neki klub nema dodanog Discord bota pružiti metodu da ga pozove u taj klub.

5.2. Funkcionalni zahtjevi web aplikacije

Web aplikacija će biti primarno dizajnirana za tri skupine korisnika platforme: vlasnik kluba, moderator kluba te običan član Discord kluba. Vlasnik kluba i moderator kluba mogu pomoću web stranice web aplikacije konfigurirati kako se funkcionalnosti Discord bota ponašaju za pojedine čehove u kojima se nalaze. Običan korisnik će moći samo koristiti te funkcionalnosti i u slučaju da želi, pozvati Discord bota u svoje klubove u kojima je on vlasnik.

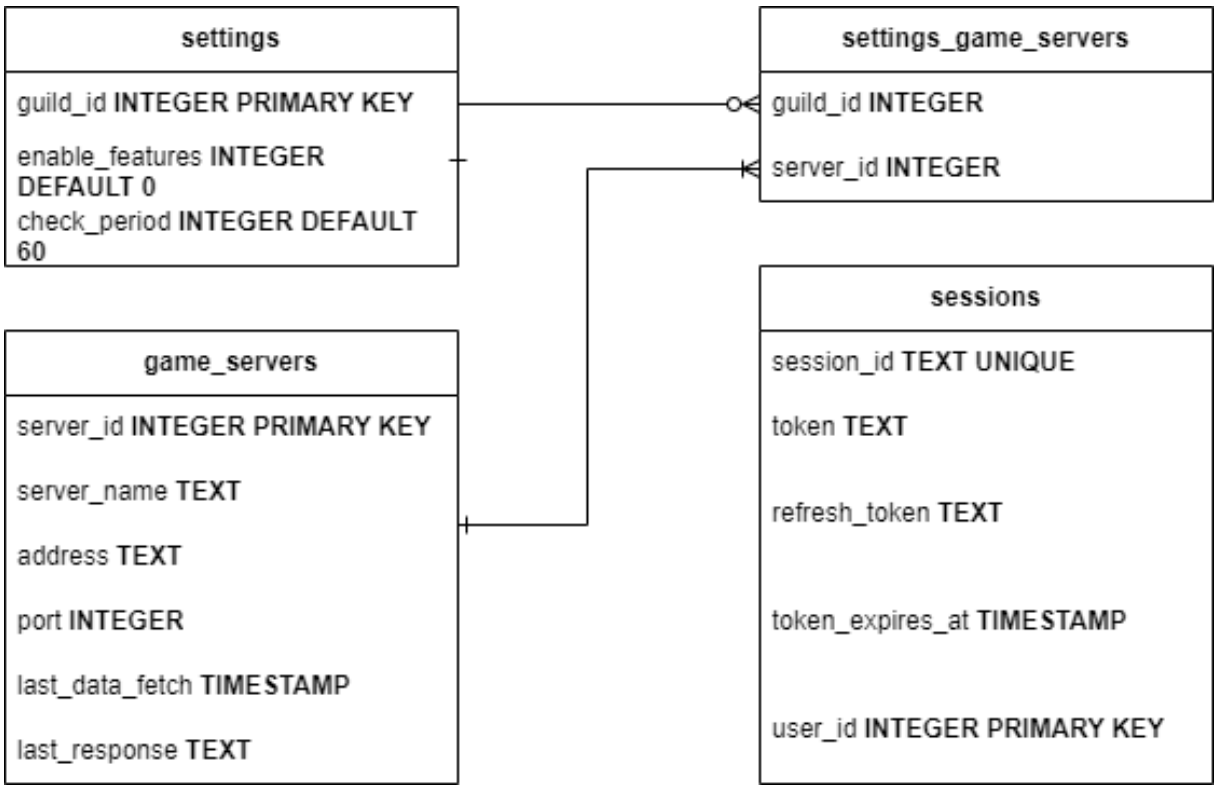


Slika 9: Dijagram slučaja upotrebe [autorski rad]

- **Član Discord kluba**
  - Može vidjeti sve Source Engine servere koje je vlasnik kluba postavio da Discord bot prati pomoću naredbe u Discordu koja prikazuje ažurnu listu servera.
  - Može vidjeti poruke koje prikazuju ažurnu listu broj igrača u svim Source Engine serverima koje je vlasnik kluba podesio da prati.
  - Može pomoću naredbe u Discord sučelju vidjeti ažurnu listu igrača koji se nalaze u Source Engine serveru iz popisa servera.
  - Može predložiti Source Engine servere koje vlasnik kluba može prihvatiti ili odbiti.
  - Može pozvati Discord bota direktno iz sučelja Discord aplikacije u vlastite klubove.
  - Može se prijaviti na web aplikaciju da doda Discord bota u svoj klub i podesiti njegove postavke za vlastiti klub.
- **Moderator Discord kluba**
  - Može koristiti sučelje web aplikacije da vidi sve klubove nad kojima može koristiti funkcionalnosti Discord bota.
  - Može vidjeti ime, broj trenutno aktivnih korisnika i ukupno aktivnih korisnika.
  - Može konfigurirati ako Discord bot šalje poruke s trenutnim brojem korisnika u svim Source Enigine serverima koji su dodani u konfiguraciju kluba.
  - Može podesiti vremenski razmak između svake poruke s trenutnim brojem korisnika i svim Source Enigine serverima koji su dodani u konfiguraciju kluba.
  - Može podesiti koji su Source Engine serveri prisutni u konfiguraciji Discord bota za server nad kojim moderira.
  - Može uklanjati Source Engine servere s popisa praćenih servera.
- **Vlasnik Discord kluba**
  - Može dodati i ukloniti Discord bota iz Discord klubova u kojima je on vlasnik.
  - Može odrediti koja prava Discord bot ima unutar Discord kluba, odnosno dali može slati i vidjeti poruke u klubu, instalirati naredbe u Discord klubu i dali može vidjeti druge članove Discord kluba.
  - Može pomoću Discorda ograničiti gdje funkcionalnosti Discord bota se mogu koristiti, na primjer samo unutar određenog tekstualnog kanala.

5.3. ERA dijagram baze podataka web aplikacije

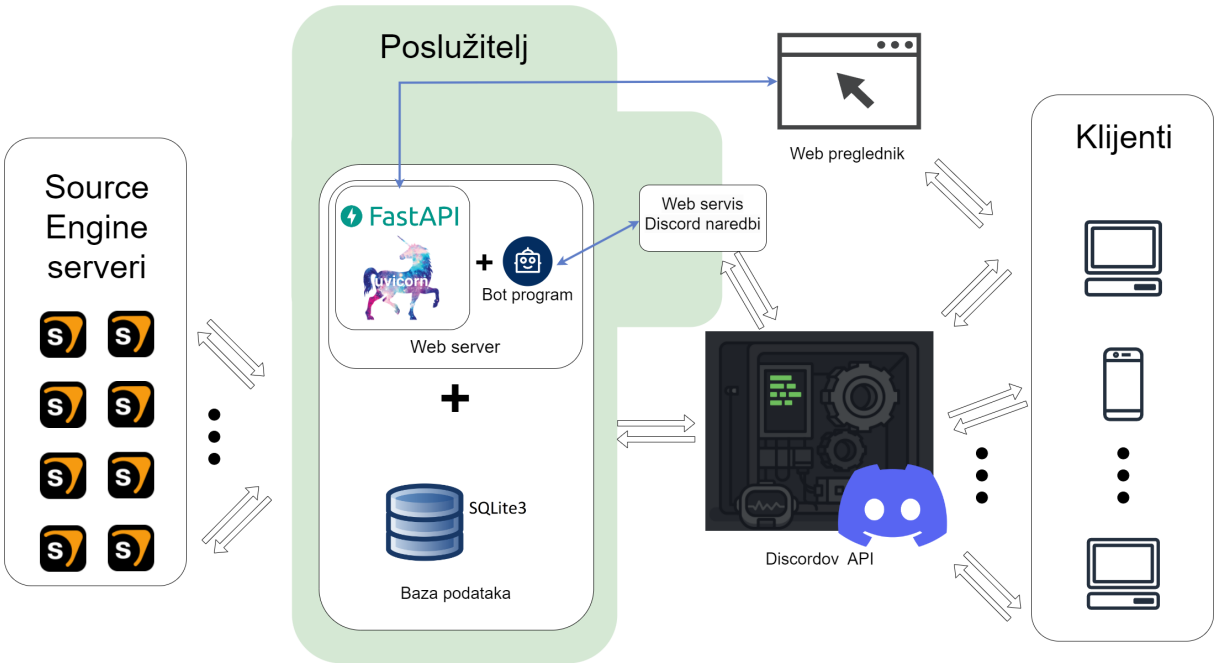
Relacijska baza podataka SQLite3 web aplikacije sastoji se od 4 tablica: `settings`, `settings_game_servers`, `game_servers` i `sessions`. `sessions` tablica služi za spremanje sesija koje se uspostavljaju nakon što Discord API potvrdi indentitet korisnika pomoću OAuth2 mehanizma. Tablica `settings` sprema vrijednosti postavki koje vlasnik kluba personalizira, uz to ona se povezuje s tablicom `game_servers` preko međutablice `settings`, `settings_game_servers` kako bi na jednostavan način mogli pratiti sve Source Engine servere koje vlasnik kluba želi da njegovi korisnici mogu pratiti.



Slika 10: ERA dijagram arhitekture baze podataka [autorski rad]

5.4. Dijagram arhitekture sustava

Glavna arhitektura web aplikacije prikazana je u zeleno obojanom dijelu prikaza ispod. Arhitektura aplikacije sastojati će se od 4 sloja. Prvi sloj aplikacije jest dohvaćanje podataka s raznih Source Engine poslužitelja video igara koji se nalaze na internetu. Njihovim podacima pristupa se pomoću paketa `python-a2s` kako bi prikupili potrebne podatke koje korisnici web aplikacije žele vidjeti. Drugi sloj je sloj poslužitelja web aplikacije koji se brine o obradi podataka, gdje web aplikacija obrađuje HTTP zahtjeve, dohvaća i trajno pohranjuje podatke na SQLite3 bazu podataka. Tu se nalazi i komunikacija web servisa kojeg Pycord uspostavlja da bi komunicirao s Discordovim API-jem za uspostavljanje i obradu Discord naredbi kose crte. Treći sloj aplikacije su posrednici u komunikaciji s web aplikacijom. Ovo uključuje Discordov API koji dio sadržaja web aplikacije prikazuje pomoću klijentskih programa i web verzije klijentskog programa na uređajima klijenata. Uz to tu se nalaze web preglednici klijenata koji dohvaćaju sadržaj s web aplikacije. Zadnji sloj uključuje sama klijentska računala koja tumače sadržaje i šalju zahtjeve za sadržajem od Discorda i web aplikacije.



Slika 11: Dijagram arhitekture [autorski rad]

5.5. Specifikacija API-ja web aplikacije

Web aplikacija će imati svoj API za rad s klijentima i API-jem Discorda. API će biti verzioniran i inicijalna verzija API-ja će se nalaziti. To omogućuje da ako u budućnosti dođe do primjene API-ja, a postoji dio korisnika koji je na poseban način (uglavnom alternativne metode od web preglednika koje su navedene u poglavlju 2) komunicira s web aplikacijom da može nastaviti koristiti koristiti i konzumirati web aplikaciju na taj isti način.

S obzirom da se koristi FastAPI kao okvir razvoja web aplikacije, automatski se generira dokumentacija API-ja web aplikacije na putanji `/docs`, gdje korisnik može pročitati upute kako koristiti putanje aplikacije, uz dodatnu opciju da ih testira ručnim unosom podataka. Uz to shema API-ja se može dobiti u JSON obliku s putanje `/openapi.json`.

Shema API-ja web aplikacije izgleda na sljedeći način:

- `GET /v1/` — Bazična ruta verzije 1 API-ja, na ovoj putanji korisnik se može prijaviti i uspostaviti sesiju
- `GET /v1/dashboard` — Korisnik na ovoj stranici vidi klubove u kojima je on vlasnik
- `GET /v1/dashboard/{guild_id}` — Stranica na kojoj korisnik vidi i mijenja postavke kluba
- `POST /v1/dashboard/{guild_id}/settings` — Ruta preko koje korisnik šalje promjene postavki kluba
- `GET /v1/404` — Ruta na kojoj završava korisnik ako upiše krivu rutu
- `GET /v1/callback` — API ruta na koju Discord šalje podatke o verificiranom korisniku
- `GET /v1/logout` — Ruta pomoću koje klijent zatvara postojeću sesiju
- `GET /` — Preusmjerava klijenta na najnoviju verziju API-ja

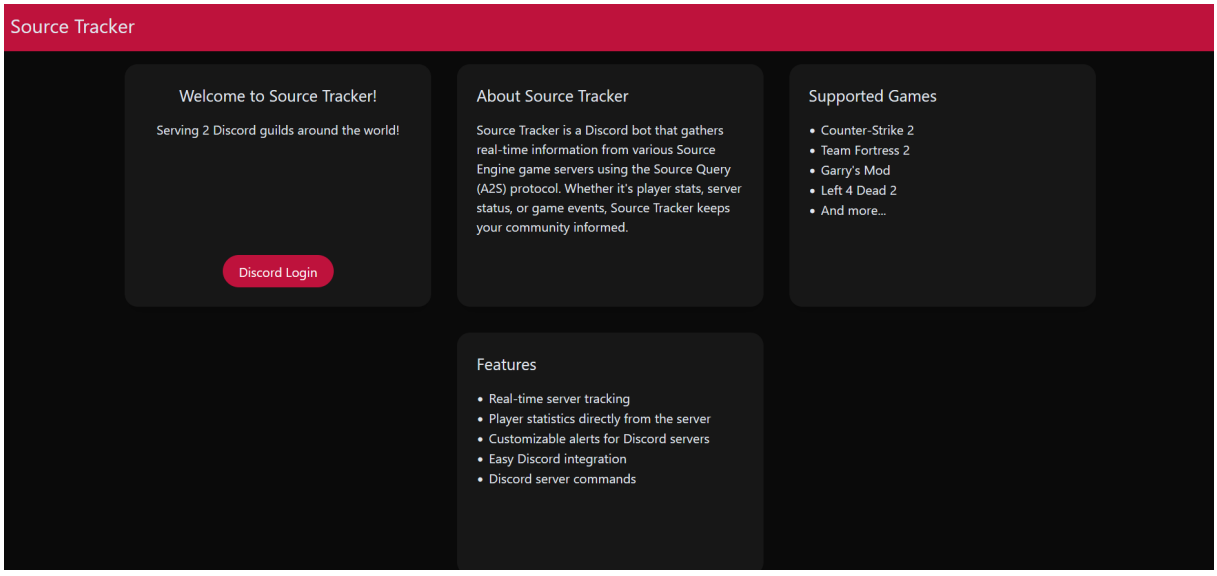
5.6. Glavne funkcionalnosti web aplikacije

U nastavku ovog odjeljka prikazane su najvažnije funkcionalnosti web aplikacije zajedno s prikazom ekrana. Prvi dio prikazuje dio web aplikacije kojoj korisnik pristupa pomoću web preglednika, a drugi dio aplikacije prikazuje funkcionalnosti vezane uz platformu Discord. Napominje se da u slučaju da se želi ugasiti web aplikacija može doći do toga da terminal naredbenog reda se blokira zbog trenutno prisutnog problema s paketom Pycord u trenutku pisanja ovog rada. Ovaj problem može se zaobići tako da se gasi terminal i ponovno pokrene dok još postoji greška. Greška je dokumentirana na poveznici: `https://github.com/Pycord-Development/pycord/issues/872`

5.6.1. Prijava na web aplikaciju S Discord OAuth2

Za prijavu na web aplikaciju preduvjet je imati kreirani i aktivirani korisnički račun na platformi Discord. Kada su ispunjen preduvjet korisnik može klikom na `Discord Login` prijaviti se u web aplikaciju. Tada ga web aplikacija preusmjerava na Discordovo sučelje za prijavu putem OAuth2 sustava.

Nakon što korisnik se prijavi u web aplikaciju kreira se sesija i web aplikacija preusmjerava klijenta na glavnu nadzornu ploču web aplikacije. U nastavku su prikazane slike ekrana postupka prijave na web aplikaciju.



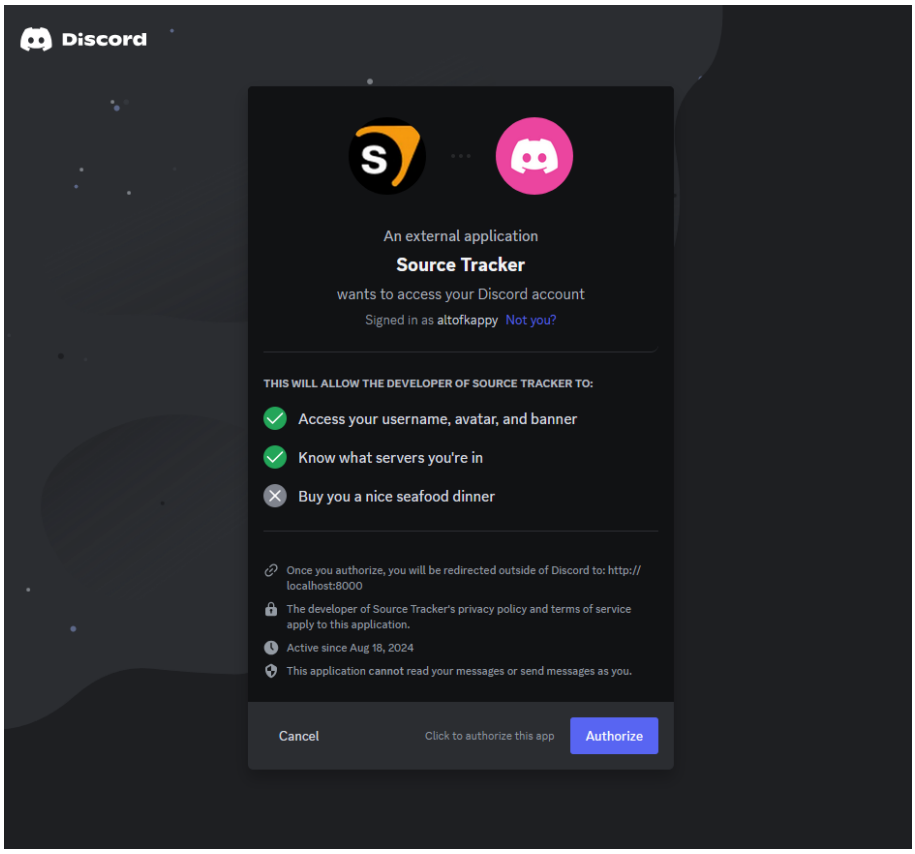
Slika 12: Prijava u web aplikaciju [autorski rad]

U nastavku je prikazana FastAPI glavna funkcija koja se brine o tome da korisnik završi na pravoj putanji ovisno o tome ako već postoji sesija u kolačićima (*engl. cookies*) preglednika i u bazi podataka web aplikacije. Ako su zadovoljeni prethodni uvjeti korisnik će biti preusmjeren na glavnu upravljačku ploču, a u suprotnom će se prikazati web stranica iz `main` funkcije.

```
pages_router = APIRouter()
```

```
@pages_router.get("/")
async def main(
    request: Request,
    page_controller: PageController = Depends(
        ControllerFactory.get_page_controller,
    ),
):
    data = MainDataModel(
        guild_count=bot.guild_count(),
        login_url=os.environ[DiscordAPI.login_url],
    )
    session_id = request.cookies.get("session_id")
    if session_id and await local_db.get_session(session_id):
        return RedirectResponse(url="/v1/dashboard")
    return page_controller.main(request=request, data=data)
```

Kada korisnik pritisne na `Discord Login` i obavi Discord autorizaciju, Discordov API vraća odgovor na putanju web aplikacije `/callback`. Kada dođe do "/callback" URL-a, prima se kod autorizacije koji se koristi za dobivanje pristupnog tokena s Discord API-a. Ako je token uspješno dobiven, dohvaća se korisnički podatak te se kreira sesija u lokalnoj bazi podataka. Sesija se identificira pomoću `session_id` kolačića koji se postavlja u preglednik korisnika. Na kraju, korisnik se preusmjerava na `/v1/dashboard` stranicu.



Slika 13: Prijava pomoću Discordovog OAuth2 sustava [autorski rad]

U nastavku je prikazano dio metode `callback`. Zaprimljeni znakovni kod `code` uz druge podatke šalje se na Discordov API asinkronom metodom `get_token_response`. Kada ako se dobije odgovor, web aplikacija kreira sesiju u bazi podataka i kreira RedirectResponse s kolačićem sesije koji ima varijable `httponly=True` i `secure=True` za povećanje sigurnosti kolačića.

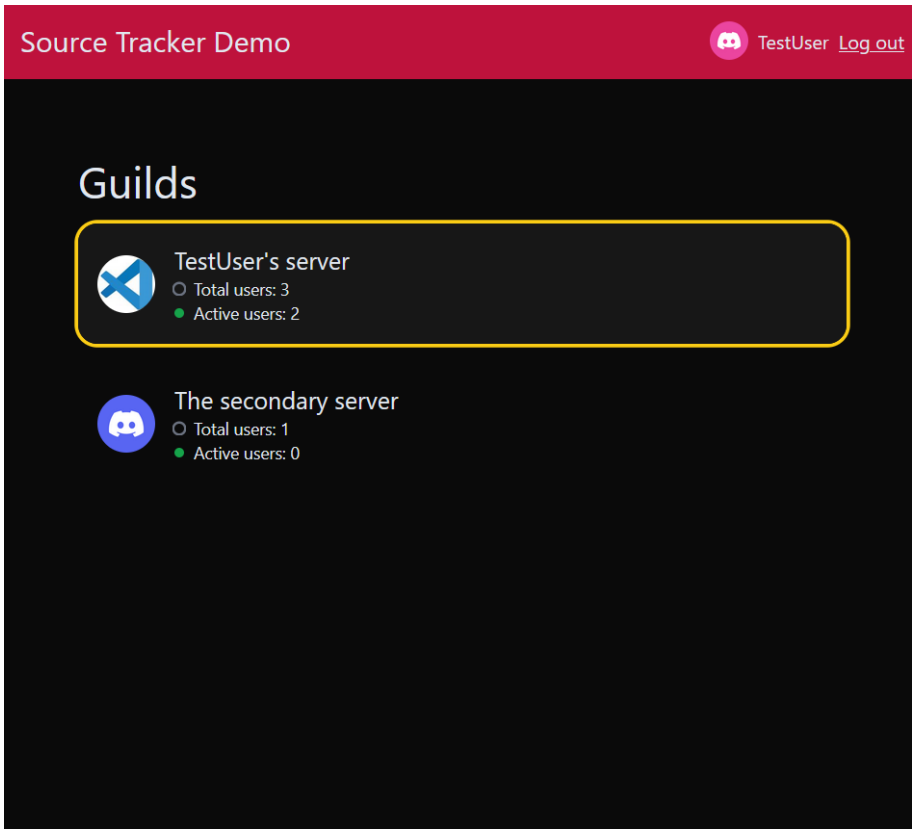
```
oauth2_router = APIRouter()

@oauth2_router.get("/callback")
async def callback(
    code: str,
    ...
    ) -> RedirectResponse:
    await discord_api.setup()
    data = OAuth2BodyData(
        ...
        code=code,
    )
    result = await discord_api.get_token_response(
        data.model_dump()
    )
```

```
    if result is None:
        raise HTTPException(
            status_code=401,
            detail="Invalid Auth Code",
        )
    token, refresh_token, expires_in = result
    user = await discord_data.get_user(token)
    user_id = user.id
    session_id = await local_db.add_session(
        token,
        refresh_token,
        expires_in,
        user_id,
    )
    response = RedirectResponse(url="/v1/dashboard")
    response.set_cookie(
        key="session_id",
        value=session_id,
        httponly=True,
        secure=AppConstants.https_secure_mode,
    )

    await discord_api.close()
    return response
```

5.6.2. Glavna upravljačka ploča web aplikacije

Na glavnoj upravljačkoj ploči korisnik može vidjeti sve klubove u kojima on ima administrativna prava nad klubom, ukupan broj članova i trenutno aktivnih članova po klubu. Također korisnik može vidjeti svoje korisničko ime i profilnu sliku Discord računa s kojim se on prijavio u web aplikaciju.

Klikom na jedan klub iz popisa klubova korisnika se preusmjerava na putanju `/v1/dashboard/{guild_id}` gdje `guild_id` jest Discordov unikatni indentifikator tog specifičnog Discord kluba.



Slika 14: Prikaz klubova gdje je korisnik vlasnik [autorski rad]

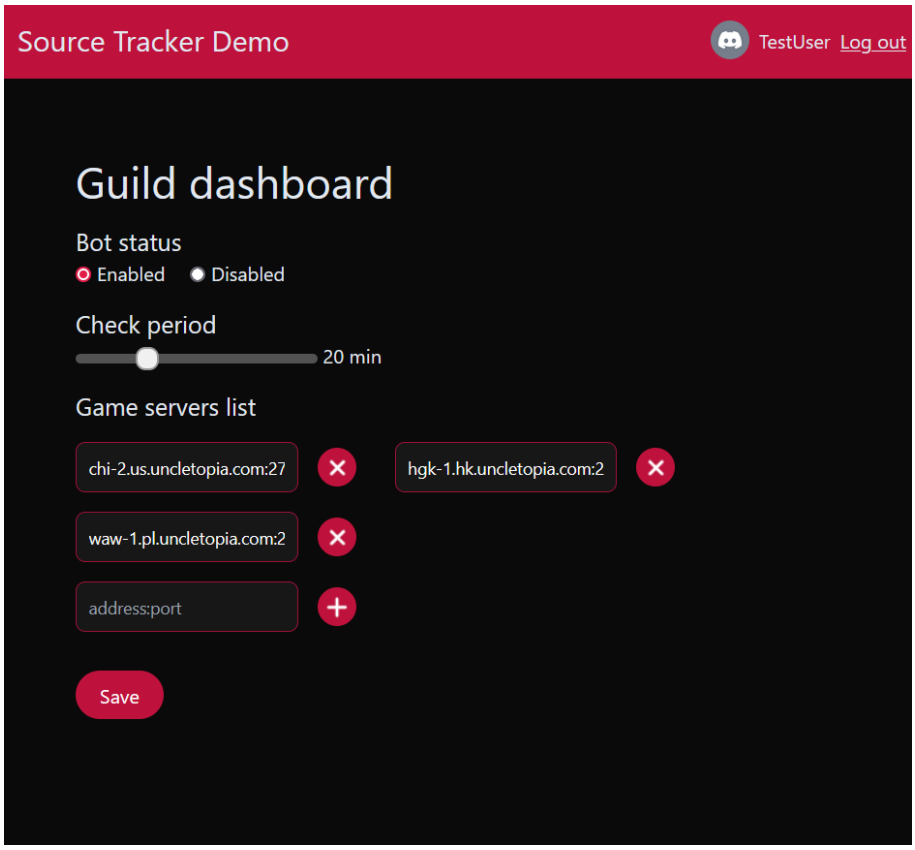
U nastavku je prikazana funkcija glavne upravljačke ploče. Ovdje se provjerava ako klijent ima validnu sesiju i ako sesija nije istekla. Nakon toga dohvaćaju se podaci iz baze podataka potrebni za glavnu upravljačku ploču i prosljeđuju se `page_contolleru` koji upravlja Jinja2 odgovorima.

```
@pages_router.get("/dashboard", response_class=HTMLResponse)
async def global_dashboard(
    ...
) -> _TemplateResponse:
    session_id = request.cookies.get("session_id")
    session = await local_db.get_session(session_id)
    if not session_id or not session:
        return RedirectResponse("/")
    token, refresh_token, token_expires_at = session
    user = await discord_data.get_user(token=token)
    if datetime.now() > token_expires_at:
        await check_session(session_id, refresh_token)
    user_guilds = await discord_data.get_guilds(token=token)
```

```
dashborad_guilds = filter_guilds(user_guilds)
user_avatar = discord_data.get_user_avatar(user)
return page_controller.global_dashboard(
    request=request,
    data=DashboardDataModel(
        user_avatar=user_avatar,
        username=user.global_name,
        guilds=dashborad_guilds,
    ),
)
```

5.6.3. Upravljačka ploča Discord kluba

Nakon što korisnik odabere jedan klub za koji želi promjeniti postavke, web aplikacija ga preusmjerava na upravljačku ploču tog kluba. Ovdje korisnik može dodati Source Engine servere koje želi da drugi članovi kluba imaju pristup iz kluba, koliko često da se provjerava koliko jest igrača na Source Engine serverima i dali su funkcionalnosti Discord bota uključene ili isključene.



Slika 15: Prikaz sučelja za promjenu postavki korisnikovog kluba [autorski rad]

Programski kod upravljačke ploče kluba sličan je globalnoj upravljačkoj ploči, uz razliku da se šalje više potrebnih podataka da Jinja2 može uvjetovano dodavati i uklanjati sadržaj s web stranice ovisno o tome ako je prisutan Discord bot u klubu koji je korisnik otvorio.

```
@pages_router.get(
    "/dashboard/{guild_id}/",
    response_class=HTMLResponse,
    )
async def guild_dashboard(
    guild_id: int,
    ...
```

```
) -> _TemplateResponse:
    session_id = request.cookies.get("session_id")
    session = await local_db.get_session(session_id)
    if not session_id or not session:
        return RedirectResponse("/")
    token, _, _ = session
    user = await discord_data.get_user(token=token)
    user_avatar = discord_data.get_user_avatar(user)
    settings = await local_db.get_bot_settings(guild_id)
    game_servers = await local_db.get_game_servers(guild_id)
    enable_features = False
    check_period = DashboardConstants.check_period_min
    if settings:
        _, enable_features, check_period = settings

    bot_invited = bot.is_in_guild(guild_id=guild_id)
    invite_url = HttpUrl(
        url=os.environ[DiscordAPI.bot_invite_link]
    )
    data = GuildDashboardDataModel(
        guild_id=guild_id,
        ...
    )
    return page_controller.guild_dashboard(
        request=request, data=data
    )
```

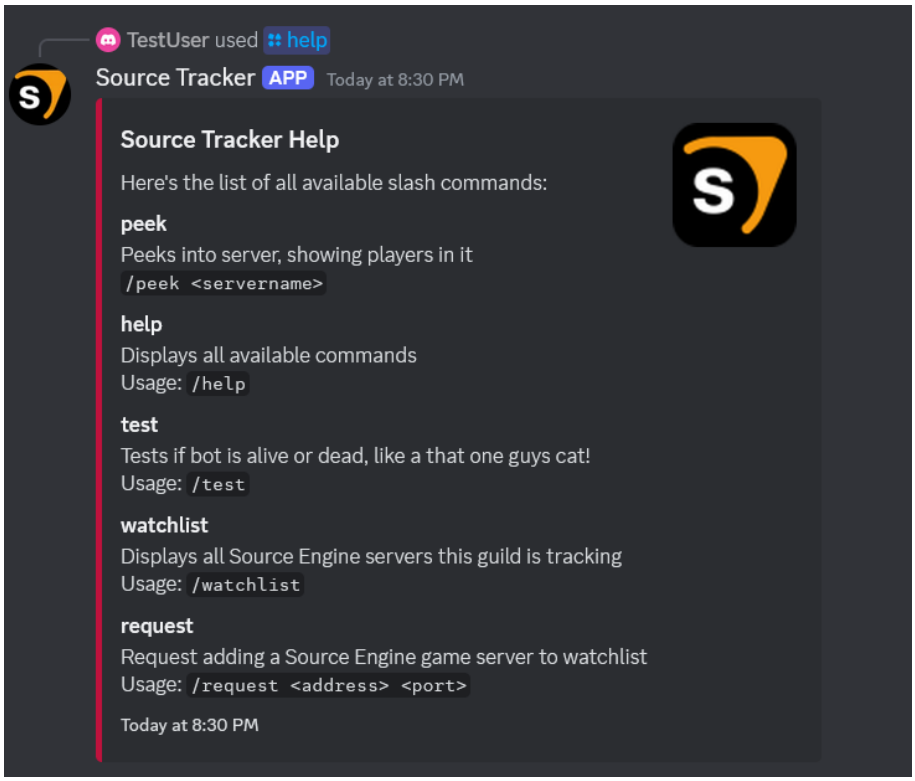
Iako su slične putanje, upravljačka ploča kluba na istoj putanji prima ažurirane podatke od klijenta preko `POST` metode. Od klijenta zatražuje `guild_id`, `new_settings` i `session_id` da bi se promjene uspješno trajno spremile u bazu podataka web aplikacije. `new_settings` je podatkovna klasa koja prima podatke iz sučelja. Bazira se na klasi `BaseModel` iz Pydantic paketa o kojem ovisi FastAPI, koji apstrahira validaciju podataka unutar aplikacije i od klijenta, pa time olakšava razvojnom programeru pisanje programske logike.

```
@pages_router.post("/dashboard/{guild_id}/settings")
async def change_settings(
    guild_id: int,
    new_settings: DashboardSettings,
    session_id: str = Cookie(None),
):
    user_id = await local_db.get_user_id(session_id)
```

```
if not session_id or not user_id:
    raise HTTPException(
        status_code=401,
        detail="No authorization",
    )
perms = bot.check_perms(guild_id=guild_id, user_id=user_id)
if not perms:
    return JSONResponse(
        status_code=401,
        content="You do not have access to this guild",
    )

settings = await local_db.exists_settings(guild_id=guild_id)

if settings:
    new_settings.guild_id = guild_id
    await local_db.update_dashboard_settings(new_settings)
else:
    new_settings.guild_id = guild_id
    await local_db.add_settings(new_settings)
```

5.6.4. Discord naredbe kose crte

Naredbe kose crte (*engl. slash commands*) funkcionalnost je platforme Discord koja razvojnim programerima omogućava kreiranje svojih funkcionalnosti koje korisnici platforme mogu koristiti uz uvjet da imaju u klubu Discord bota koji pruža naredbe. Da bi korisnik mogao izvršiti naredbu, mora započeti pisati `/` i Discord će otvoriti sučelje sa svim mogućim naredbama kose crte. Naredbe kose crte mogu zaprimiti parametre koje Discord provjerava ako su ispravnog tipa podataka za razvojnog programera treće strane. Najjednostavniji primjer takvih naredbi su naredbe pomoći (*engl. help commands*) koje vraćaju uputstva kako koristiti druge funkcionalnosti Discord bota.



Slika 16: Prikaz naredbe za pomoć pri radu s Discord botom [autorski rad]

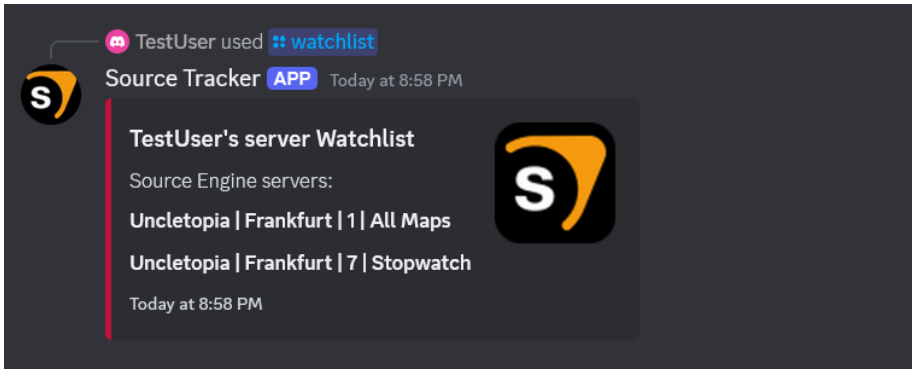
U paketu Pycord ta interakcija s Discordu se pojednostavljuje s takozvanim anotacijskom funkcijom (*decorator function*) `discord.ext.commands.slash_commands` koja se stavi iznad funkcije koju želimo koristiti kao poslovnu logiku određene naredbe kose crte. Anotacijska funkcija je skraćeno pisanje funkcije koja vraća drugu funkciju kao svoj rezultat. Funkcija poslovne logike obavezno mora zaprimiti parametar podatkovnog konteksta `ctx` koji je tipa `discord.ApplicationContext` koji sadrži podatke o tome na koji način se je izvršila naredba i okolnosti kao što je u kojem klubu, koja osoba i slični podaci. Svaka interakcija s nekom naredbom mora završiti s time da web aplikacija odgovori Discord API-ju s naredbama `ctx.respond` ili `ctx.send_followup`, s time da prethodna naredbu mora prethoditi poziv naredbe `ctx.defer`. Primjer ispod je programska logika `/help` naredbe kose crte.

```
@commands.slash_command(
    usage="/help",
    description=
    "Displays all available commands\n Usage: `/help`",
)
async def help(self, ctx: discord.ApplicationContext) -> None:
    await ctx.respond(embed=HelpEmbed(self.bot))
```

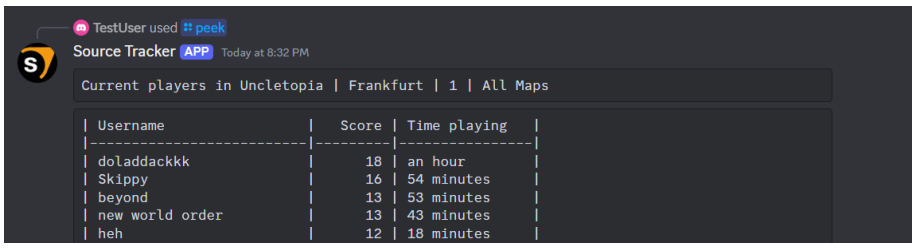
Programska logika je smještena u klasu `HelpEmbed` koja nasljeđuje Pycordovu klasu `discord.Embed`. Ova klasa je apstrakcija Pycorda za Discord element sučelja s istim imenom, koji Pycord šalje na API da bi se prikazao na klijentovom sučelju. Pozivom konstruktora nadklase s funkcijom `super().__init__()` postavljamo sve naslijeđene atribute nadklase koju onda Pycord može prenijeti na Discordov API.

```
class HelpEmbed(discord.Embed):
def \__init__(self, bot: Bot) -> None:
    title = f"{BotConstants.name} Help"
    description = """
    Here's the list of all available slash commands:
    """

    embed_fields: list[discord.EmbedField] = []
    for c in bot.walk_application_commands():
        embed_fields.append(
            discord.EmbedField(name=c.name, value=c.description)
        )

    super().__init__(
        title=title,
        description=description,
        color=BotConstants.color,
        timestamp=datetime.now(),
        thumbnail=bot.user.avatar.url,
        fields=embed_fields,
    )
```

Na sličan način su izrađene naredba `/watchlist` koja prikazuje sve Source Engine servere koje klub prati. Naredba `/peek` izrađena je bez `discord.Embed` klase, nego koristi čistu tekstualnu funkcionalnost Discord-a gdje ako niz znakova je okružen s nizom znakova ‟ ` s lijeve i desne strane, tekst će se tretirati kao isječak programskog koda (*engl. code block*). Uz to `/peek` naredba koristi funkcionalnost da Discord šalje parametre koje korisnik unese web aplikaciji s Pycord anotacijskom funkcijom `discord.option`. Slike ispod prikazuju primjere izvršavanja naredbi unutar Discorda:



Slika 17: Prikaz rezultata naredbe `/watchlist` [autorski rad]



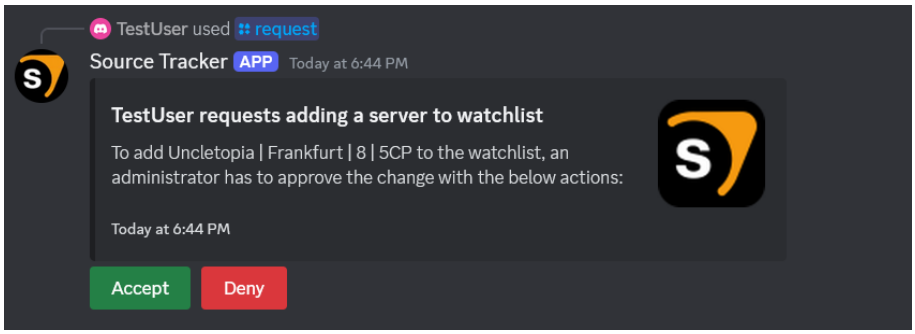
Slika 18: Prikaz rezultata naredbe `/peek` [autorski rad]

Programski kod ispod prikazuje kako se koristi `discord.option` funkcija za parametre naredbe kose crte.

```
...
@discord.option(
    name="server",
    description="Pick a server",
    autocomplete=autocomplete_server,
)
async def peek(
        self,
        ctx: discord.ApplicationContext,
        server: str,
    ) -> None:
    server: GameServer = GameServer.model_validate_json(server)
    game_servers = await local_db.get_game_servers_by_guild(
        guild_id=ctx.interaction.guild_id
```

```
    )
    ...
```

Zadnja funkcionalnost koju web aplikacija obrađuje preko Discordovog API-ja jest naredba kose crte `/request`. Naredba kreira `discord.Embed` s dodatnim Pycord elementom `discord.ui.View`. `discord.ui.View` omogućuje dodavanja elemenata za unos na poruku kao što su gumbovi, padajući izbornici i slični. U ovom slučaju koriste se dva gumba za potvrdu da se zatraženi Source Engine server doda na popis praćenih servera kao na slici ispod:



Slika 19: Prikaz zahtjeva generiranog s `/request` naredbom [autorski rad]

Na sličan način kao i `discord.Embed`, element `discord.ui.View` može se nadodati u poruku s `view` argumentom u nekoj od metoda za slanje poruka kao u isječku ispod:

```
...
view = RequestView(server=valid_server)

if valid_server:
    await ctx.send_followup(
                view=view,
                embed=RequestEmbed(ctx, valid_server),
                ephemeral=False,
            )
...
```

U nastavku je prikazana podklasa `RequestView` koja nasljeđuje `discord.ui.View`. U podklasi `RequestView` prikazano je kako se dodaju interaktivni elementi kao podklasa `ActionButton` koji će se vidjeti na poruci u trenutku kada se ona pošalje.

```
class RequestView(discord.ui.View):
    def __init__(self, server: ValidGameServer) -> None:
        ...
        self.add_item(
            ActionButton(
                label="Accept",
```

```
                style=discord.ButtonStyle.green,
                custom_callback=self.accept,
            )
        )
...

    async def accept(self) -> None:
        await self.save_to_database()
        await self.message.edit(
            f"{self.server.server_name} was successfully accepted",
            embed=None,
            view=None,
        )
...
    async def deny(self):
        await self.message.delete()
```

U programskome kodu ispod prikazana je klasa `ActionButton` koja nasljeđuje klasu `discord.ui.Button`. Metoda nadklase pod nazivom `callback` je nadjačana tako da izvrši `custom_callback` funkciju koja je proslijeđena prilikom stvaranja objekta:

```
class ActionButton(discord.ui.Button):
    def __init__(
        self,
        label: str,
        style: discord.ButtonStyle,
        custom_callback: Callable[..., Coroutine],
    ) -> None:
        self.custom_callback = custom_callback
        super().__init__(
            label=label,
            style=style,
        )

    async def callback(self, interaction: discord.Interaction):
        if interaction.permissions.administrator:
            await self.custom_callback()
            await interaction.respond(
                "`Action was confirmed!`",
                ephemeral=True,
            )
        ...
```

6. Zaključak

Razvoj web aplikacija postaje sve popularniji i dobiva na svojoj važnosti s obzirom na potrebe suvremenih korisnika i poslovnih subjekata za naprednim funkcionalnostima i pouzdanošću. U ovom radu obrađena je implementacija web aplikacije koristeći različite tehnologije kao što su Python, FastAPI, i Discord API, kako bi se izgradila skalabilna aplikacija za podršku zajednica korisnika koji koriste Discord platformu.

FastAPI je moderan okvir za razvoj web aplikacija s fokusom na izradu API-ja. U ovom radu prikazana je primjena FastAPI-ja u izradi robusnih i skalabilnih API-ja koji podržavaju sve potrebne funkcionalnosti web aplikacije. FastAPI omogućuje jednostavnu integraciju s drugim tehnologijama te uvelike smanjuje kompleksnost razvoja zahvaljujući svojoj intuitivnoj sintaksi i mogućnostima koje nudi.

Pycord je asinkrona biblioteka za Python dizajnirana za interakciju s Discord API-jem. Koristeći Pycord, programeri mogu jednostavno integrirati Discord funkcionalnosti unutar svojih web aplikacija, omogućujući kreiranje botova, upravljanje događajima i komunikaciju s korisnicima unutar Discord zajednice i pojedinih klubova. U ovom radu prikazana je implementacija Discord bota korištenjem Pycord-a, što omogućuje učinkovito povezivanje i proširenje funkcionalnosti web aplikacije unutar Discord okruženja. Pycord se ističe po svojoj lakoći korištenja i širokom spektru mogućnosti koje nudi za interakciju s Discord API-jem.

Zaključak ovog rada je da se primjenom odabranih tehnologija može izraditi kvalitetna, prilagodljiva i skalabilna web aplikacija koja zadovoljava moderne korisničke potrebe, s posebnim naglaskom na integraciju s vanjskim platformama poput Discorda, čime se proširuje opseg i funkcionalnost aplikacije.

Popis literature

[1] J. Cai, Y. F. Lin, H. Zhang i J. M. Carroll, „Third-Party Developers and Tool Development For Community Management on Live Streaming Platform Twitch," Association for Computing Machinery, svibanj 2024., ISBN: 9798400703300. DOI: 10.1145/3613904.3642787.

[2] T. S. BV, *TIOBE Programming Community Index*, https://www.tiobe.com/tiobe-index/ (Pristupljeno 13.6.2024.), lipanj 2024. adresa: https://www.tiobe.com/tiobe-index/.

[3] A. Taivalsaari i T. Mikkonen, „The web as an application platform: The Saga continues," 2011., str. 170–174, ISBN: 9780769544885. DOI: 10.1109/SEAA.2011.35.

[4] IBM, *What is an API?* https://www.ibm.com/topics/api (Pristupljeno 4.7.2024.)

[5] R. Bridgewater i M. Cole, *Instant Messaging Reference: A Practical Guide* (Chandos Information Professional Series). Elsevier Science, 2008., ISBN: 9781780631233. adresa: https://books.google.hr/books?id=F4ikAgAAQBAJ.

[6] J. Simmons, *A roadmap and appeal for help from The Matrix.org Foundation*, https://matrix.org/blog/2024/01/2024-roadmap-and-fundraiser/ (Pristupljeno 20.6.2024.), siječanj 2024.

[7] A. Eylenburg, J. Manning, R. Calixte i D. Danelski, *Comparison of Instant Messengers*, svibanj 2021. adresa: https://eylenburg.github.io/im_comparison.htm.

[8] B. Dean, *How Many People Use Telegram?* https://backlinko.com/telegram-users (Pristupljeno 20.6.2024.), srpanj 2024.

[9] D. Curry, *Discord Revenue and Usage Statistics (2024)*, https://www.businessofapps.com/data/discord-statistics/ (Pristupljeno 24.6.2024.), svibanj 2024.

[10] H. Singh i A. Agnihotri, „DISCORD BOT," str. 9, adresa: http://www.ir.juit.ac.in:8080/jspui/bitstream/123456789/3628/1/Discord%20Bot.pdf.

[11] A. Verma, S. Tyagi i G. Mathur, „A Comprehensive Review on Bot - Discord Bot," *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, str. 532–536, travanj 2021. DOI: 10.32628/cseit2172100.

[12] Okta Inc., „What is OAuth 2.0?," https://auth0.com/intro-to-iam/what-is-oauth-2 (Pristupljeno 18.8.2024.)

[13] M. Fric, *Razvoj web aplikacije u programskom jeziku TypeScript primjenom NextJS okvira*, Urn.nsk.hr, srpanj 2023. adresa: https://urn.nsk.hr/urn:nbn:hr:211:833260 (pogledano 14. 8. 2024.).

[14] CodeDocs.org, *Python (programming language)*, https://codedocs.org/what-is/python-programming-language (Pristupljeno 10.8.2024.), 2007.

[15] *Blender 4.2 Python API Documentation*, https://docs.blender.org/api/current/info_quickstart.html (Pristupljeno 13.8.2024.)

[16] *Unreal Python API Introduction*, https://dev.epicgames.com/documentation/en-us/unreal-engine/python-api/introduction?application_version=5.4 (Pristupljeno 13.8.2024.)

[17] P. Kumar, *Python Keywords*, https://www.askpython.com/python/python-keywords (Pristupljeno 13.8.2024.), ožujak 2019.

[18] mozilla.org, *Introducing asynchronous JavaScript*, https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous/Introducing (Pristupljeno 19.8.2024.)

[19] CodeDocs.org, *Duck typing*, https://codedocs.org/what-is/duck-typing (Pristupljeno 14.8.2024.)

[20] Python.org, *venv — Creation of virtual environments*, https://docs.python.org/3/library/venv.html (Pristupljeno 16.8.2024.)

[21] V. S. Code, *Python environments in VS Code*, https://code.visualstudio.com/docs/python/environments (Pristupljeno 16.8.2024.)

[22] PyCharm, *Configure a virtual environment*, https://www.jetbrains.com/help/pycharm/creating-virtual-environment.html#python_create_virtual_env (Pristupljeno 16.8.2024.), 2024.

[23] N. Wamaitha, *What is __init__.py for in Python?* https://sentry.io/answers/what-is-init-py-for-in-python/ (Pristupljeno 16.8.2024.), prosinac 2023.

[24] TechEmpower.com, *Web Framework Benchmarks*, https://www.techempower.com/benchmarks/#section=test&runid=7464e520-0dc2-473d-bd34-dbdfd7e85911&hw=ph&test=query&l=zijzen-7 (Pristupljeno 10.7.2024.)

[25] FastAPI, *FastAPI homepage*, https://fastapi.tiangolo.com/ (Pristupljeno 13.6.2024.)

[26] Valve Developer Community, *Server queries*, https://developer.valvesoftware.com/wiki/Server_queries (Pristupljeno 20.8.2024.), svibanj 2024.

Popis slika

Popis tablica

1. Prilog 1

Ovom radu priložen je programski kod web aplikacije u zip arhivi koji se može preuzeti na sustavu FOI radovi.