|  |
| --- |
| Elektronski fakultet u Nišu | Niš |
| Elektronski fakultet u Nišu  Univerzitet u Nišu |
| Rad |
| **SecureTalk aplikacija** |
| **Student**  **Milan Jovanović 1919/24** |
| Avgust 2025. |

Sadržaj

[Uvod 3](#_Toc207289131)

[Značaj komunikacionih tehnologija u savremenom društvu 3](#_Toc207289132)

[Istorijski kontekst razvoja komunikacionih tehnologija 3](#_Toc207289133)

[Društveni i tehnički uticaj savremenih tehnologija 3](#_Toc207289134)

[Pregled sigurnosnih izazova 4](#_Toc207289135)

[Kratak pregled razvoja instant poruka i VoIP tehnologija 4](#_Toc207289136)

[Ciljevi seminarskog rada 4](#_Toc207289137)

[Šta su instant poruke i kako funkcionišu 5](#_Toc207289138)

[Koncept i protokoli 5](#_Toc207289139)

[HTTP/HTTPS 5](#_Toc207289140)

[WebSocket 5](#_Toc207289141)

[WhatsApp 5](#_Toc207289142)

[Telegram 6](#_Toc207289143)

[Upoređivanje WhatsApp-a i Telegram-a: 6](#_Toc207289144)

[Kako funkcionišu mobilni razgovori 6](#_Toc207289145)

[Razlika između tradicionalnih poziva i VoIP poziva 6](#_Toc207289146)

[Circuit Switching (Klasična telefonska mreža) 6](#_Toc207289147)

[Packet Switching (VoIP tehnologija) 7](#_Toc207289148)

[Osnovni protokoli za VoIP 7](#_Toc207289149)

[Session Initiation Protocol (SIP) 7](#_Toc207289150)

[Real-time Transport Protocol (RTP) 7](#_Toc207289151)

[Sigurnosni izazovi u VoIP komunikaciji 8](#_Toc207289152)

[Server.py 8](#_Toc207289153)

[Biblioteke I konfiguracija 9](#_Toc207289154)

[Inicijalizacija aplikacije 9](#_Toc207289155)

[Struktura podataka 10](#_Toc207289156)

[Pomoćne funkcije 10](#_Toc207289157)

[Upravljanje sobama 10](#_Toc207289158)

[Osnovna ruta 12](#_Toc207289159)

[WebSocket endpoint 12](#_Toc207289160)

[Slanje prvog “pozdrava” 13](#_Toc207289161)

[Primanje i slanje poruka 13](#_Toc207289162)

[Registracija korisnika 14](#_Toc207289163)

[Client.py 16](#_Toc207289164)

[Globalne promenljive 17](#_Toc207289165)

[EncryptedAudioCall 17](#_Toc207289166)

[SecureTalkClient 20](#_Toc207289167)

[Handle Server 27](#_Toc207289168)

[room\_joined 29](#_Toc207289169)

[Presence 29](#_Toc207289170)

[Message 29](#_Toc207289171)

[File 30](#_Toc207289172)

[Ostale funkcije 30](#_Toc207289173)

[Helper 30](#_Toc207289174)

[GUI akcije 31](#_Toc207289175)

[Main 36](#_Toc207289176)

[Literatura 37](#_Toc207289177)

# Uvod

## Značaj komunikacionih tehnologija u savremenom društvu

Komunikacija je osnovna potreba svakog čoveka i temelj razvoja savremenog društva. Sa razvojem tehnologije, način na koji komuniciramo je pretrpeo revoluciju, prelazeći sa tradicionalnih metoda poput pisama i telefonskih poziva na digitalne forme komunikacije koje obuhvataju instant poruke, video pozive i platforme za društvene mreže. Ove tehnologije omogućavaju ne samo brzu i efikasnu razmenu informacija, već i globalnu povezanost ljudi, organizacija i sistema.

Digitalna komunikacija danas ima neprocenjivu ulogu u različitim oblastima života, uključujući poslovanje, obrazovanje, medicinu i društvene odnose. Na primer, aplikacije za razmenu poruka poput WhatsApp-a i Telegram-a svakodnevno koriste milijarde ljudi širom sveta za razmenu informacija, dok VoIP (Voice over IP) tehnologija omogućava ekonomičnije i fleksibilnije telefonske razgovore u poređenju sa tradicionalnim metodama. Pored toga, pandemija COVID-19 dodatno je ubrzala upotrebu ovih tehnologija, jer su postale osnovno sredstvo za održavanje poslovne i lične povezanosti tokom globalnih ograničenja.

## Istorijski kontekst razvoja komunikacionih tehnologija

Razvoj komunikacionih tehnologija može se pratiti od ranih oblika, poput telegrafa i telefona, do savremenih digitalnih mreža. Telegraf, uveden u 19. veku, bio je prva tehnologija koja je omogućila brzu razmenu informacija na velikim udaljenostima, koristeći Morzeovu azbuku za prenos poruka. Ovo je revolucionisalo način na koji su ljudi komunicirali, otvarajući vrata globalnoj povezanosti.

Telefon, patentiran krajem 19. veka od strane Aleksandra Grahama Bela, predstavljao je sledeći veliki korak u razvoju komunikacija. Omogućio je direktan glasovni kontakt, čime je komunikacija postala prirodnija i pristupačnija. Razvoj telefonskih mreža tokom 20. veka dalje je unapredio ovu tehnologiju, uključujući uvođenje automatskih centrala i žica za podmorske kablove, koje su omogućile globalnu telefoniju.

Sa dolaskom interneta krajem 20. veka, komunikacije su ušle u novu eru. E-mail je postao prvi popularni oblik digitalne razmene poruka, dok su rani servisi za razmenu trenutnih poruka, kao što su ICQ i AOL Instant Messenger, omogućili komunikaciju u realnom vremenu. Početkom 21. veka, razvojem pametnih telefona i bežičnih mreža, komunikacione tehnologije su doživele eksponencijalni rast, sa aplikacijama poput WhatsApp-a, Telegram-a i Signal-a koje su postale standard u svakodnevnom životu.

## Društveni i tehnički uticaj savremenih tehnologija

Savremene komunikacione tehnologije ne samo da su promenile način na koji ljudi komuniciraju, već su i uticale na društvene norme i svakodnevni život. Digitalna povezanost omogućila je ljudima da ostanu u kontaktu bez obzira na fizičku udaljenost, dok su poslovni procesi postali brži i efikasniji. Na primer, upotreba platformi za video konferencije poput Zoom-a i Microsoft Teams-a omogućila je kompanijama da održavaju poslovanje čak i u uslovima rada na daljinu.

Tehnički gledano, razvoj komunikacionih tehnologija zahtevao je inovacije u raznim oblastima, uključujući mrežne protokole, šifriranje i bežičnu komunikaciju. Protokoli kao što su HTTP/HTTPS i WebSocket revolucionisali su razmenu podataka na internetu, dok su kriptografski algoritmi kao što su RSA i AES omogućili sigurno čuvanje i prenos informacija.

## Pregled sigurnosnih izazova

Pored svojih prednosti, digitalne komunikacije donose i ozbiljne sigurnosne izazove. Privatnost korisnika je često meta napada, bilo kroz presretanje podataka, bilo kroz sofisticirane sajber napade poput phishing-a i ransomware-a. Ovo čini implementaciju sigurnosnih mera, kao što su end-to-end šifriranje i autentifikacija korisnika, neophodnim za održavanje poverenja i sigurnosti u digitalnom okruženju.

## Kratak pregled razvoja instant poruka i VoIP tehnologija

Instant poruke su se pojavile kao prirodni nastavak razvoja internetskih komunikacija, omogućavajući korisnicima da razmenjuju tekstualne poruke u realnom vremenu preko mreža. Prvi servisi, poput ICQ-a i AIM-a, bili su pioniri u ovoj oblasti, dok su savremene platforme poput WhatsApp-a, Telegram-a i Signal-a postavile nove standarde u brzini, sigurnosti i funkcionalnosti. Ove aplikacije danas nude ne samo tekstualnu komunikaciju, već i glasovne i video pozive, deljenje fajlova i grupne diskusije.

Slično tome, VoIP tehnologija, koja koristi internet protokole za prenose glasovnih komunikacija, predstavlja ključnu inovaciju u oblasti mobilne telefonije. Za razliku od tradicionalnih telefonskih poziva koji koriste preklapanje kanala (circuit switching), VoIP koristi paketnu komutaciju (packet switching), omogućavajući efikasniji prenos glasovnih podataka. Servisi kao što su Skype, Zoom i Microsoft Teams oslanjaju se na VoIP tehnologiju kako bi pružili kvalitetne usluge za ličnu i poslovnu upotrebu.

## Ciljevi seminarskog rada

Glavni cilj ovog seminarskog rada je istraživanje teorijskih i praktičnih aspekata savremenih komunikacionih tehnologija, sa posebnim fokusom na instant poruke i VoIP komunikaciju. Rad će biti podeljen u dva dela:

1. **Teorijska analiza**:
   * Objašnjenje koncepta i funkcionisanja instant poruka i VoIP tehnologija.
   * Razmatranje sigurnosnih izazova, uključujući problem privatnosti i neophodnost šifriranja.
2. **Praktična implementacija**:
   * Razvoj prototipa aplikacije za razmenu šifrovanih poruka.
   * Provera funkcionalnosti aplikacije kroz testiranje različitih scenarija.

Ovaj rad ima za cilj da pruži sveobuhvatno razumevanje moderne digitalne komunikacije, ukazujući na izazove i mogućnosti za dalji razvoj u ovoj oblasti.

# Šta su instant poruke i kako funkcionišu

## Koncept i protokoli

Instant poruke predstavljaju oblik digitalne komunikacije koji omogućava korisnicima razmenu tekstualnih poruka, multimedijalnih datoteka, glasovnih zapisa i drugih informacija u realnom vremenu. Ove aplikacije koriste internet kao osnovnu mrežu za prenos podataka, a njihov rad se zasniva na kombinaciji nekoliko ključnih protokola koji omogućavaju siguran i efikasan prenos informacija.

## HTTP/HTTPS

HTTP (HyperText Transfer Protocol) je osnovni protokol za razmenu podataka na internetu. Koristi se za uspostavljanje veze između klijenta (npr. aplikacije za razmenu poruka) i servera na kojem se nalaze podaci. HTTPS (HTTP Secure) predstavlja sigurniju verziju ovog protokola koja koristi SSL/TLS za šifrovanje komunikacije, čime se obezbeđuje privatnost i integritet podataka koji se razmenjuju. U kontekstu instant poruka, HTTP/HTTPS se često koristi za inicijalno slanje poruka na servere, autentifikaciju korisnika i preuzimanje potrebnih resursa.

## WebSocket

WebSocket je protokol dizajniran za komunikaciju u realnom vremenu. Za razliku od HTTP-a, koji zahteva stalno ponovno uspostavljanje veze za svaki zahtev, WebSocket omogućava uspostavljanje trajne dvosmerne veze između klijenta i servera. Ovaj protokol je idealan za aplikacije za razmenu poruka jer smanjuje kašnjenje i povećava efikasnost prenosa podataka. WebSocket se koristi za slanje i prijem poruka u realnom vremenu, bez potrebe za dodatnim zahtevima ka serveru.

**Ključne prednosti WebSocket-a:**

* Trajna veza koja minimizira mrežne resurse.
* Niža latencija u poređenju sa HTTP/HTTPS.
* Podrška za dvosmernu komunikaciju, omogućavajući brzu razmenu informacija između klijenta i servera.

## WhatsApp

WhatsApp je jedna od najpopularnijih aplikacija za razmenu poruka, sa fokusom na privatnost i sigurnost. Njegov rad zasnovan je na sledećim tehnologijama:

* **End-to-end enkripcija:** Poruke su šifrovane pomoću protokola Signal, što znači da samo pošiljalac i primalac mogu da ih čitaju.
* **Distribuirani serveri:** WhatsApp koristi servere za usklađivanje poruka, ali poruke se ne čuvaju nakon isporuke, osim ako korisnik ne omogući opciju pravljenja rezervnih kopija.
* **Real-time komunikacija:** Kombinacija WebSocket-a i prilagođenih protokola omogućava brzo slanje i primanje poruka i medijskih fajlova.

## Telegram

Telegram je još jedna popularna platforma, poznata po fleksibilnosti i opcijama za napredne korisnike. Ključne karakteristike Telegram-a uključuju:

* **MTProto protokol:** Telegram koristi sopstveni protokol za sigurnu komunikaciju, dizajniran da bude brz i bezbedan.
* **Cloud-based poruke:** Poruke se čuvaju na Telegram serverima i mogu se pristupiti sa više uređaja, dok "tajni četovi" koriste end-to-end enkripciju.
* **Brzina i pouzdanost:** Telegram koristi distribuiranu mrežu servera širom sveta kako bi minimizirao kašnjenje i povećao dostupnost.

## Upoređivanje WhatsApp-a i Telegram-a:

* WhatsApp nudi potpunu end-to-end enkripciju za sve razgovore, dok Telegram koristi enkripciju samo za "tajne četove".
* Telegram je poznat po naprednim opcijama kao što su botovi, kanali i veći kapacitet za deljenje fajlova.
* WhatsApp je zasnovan na telefonskim brojevima za identifikaciju korisnika, dok Telegram dozvoljava korišćenje korisničkih imena.

Ove dve platforme ilustruju različite pristupe dizajnu aplikacija za razmenu poruka, pri čemu svaka ima svoje prednosti i mane u pogledu sigurnosti, brzine i korisničkog iskustva.

# Kako funkcionišu mobilni razgovori

Mobilni razgovori su osnovna vrsta komunikacije u današnjem digitalnom društvu. S obzirom na napredak tehnologije, tradicionalni mobilni pozivi koji koriste fiksne mreže i savremeni VoIP (Voice over Internet Protocol) pozivi koriste različite pristupe za prenos glasovnih podataka. Ove tehnologije se značajno razlikuju po načinu prenosa informacija, protokolima koji se koriste, kao i po izazovima koje donose, posebno kada je reč o sigurnosti.

## Razlika između tradicionalnih poziva i VoIP poziva

Mobilni telefoni tradicionalno koriste **circuit switching** tehnologiju za uspostavljanje i održavanje poziva. Ovo je klasičan metod prenosa glasovnih signala preko mreže. Na drugoj strani, VoIP koristi **packet switching** tehnologiju koja omogućava prenos podataka putem interneta. Razlikovanje između ovih tehnologija ima značajnu ulogu u razumevanju načina na koji mobilni razgovori funkcionišu, kao i njihovih prednosti i slabosti.

## Circuit Switching (Klasična telefonska mreža)

Tehnologija circuit switching je osnova tradicionalnih mobilnih i fiksnih telefonskih mreža. Kada korisnik inicira poziv, telefon se povezuje sa centralnim čvorištem mreže, koje zatim usmerava poziv kroz odgovarajuće čvorove dok se ne uspostavi fizička linija između pozivaoca i primaoca. Ova fizička linija ostaje rezervisana sve dok je poziv aktivan.

* **Princip rada**: Tokom trajanja poziva, linija ostaje rezervisana, čak i kada se korisnik ne govori, što znači da se resursi telefonske mreže ne koriste efikasno.
* **Prednosti**: Ovo je izuzetno pouzdan sistem, jer linija ostaje stabilna i poziv nije podložan gubicima ili smetnjama.
* **Nedostaci**: Ovaj metod je resursno neefikasan, jer se mreža opterećuje i tokom perioda kada se ne koristi. Takođe, skalabilnost ove tehnologije je ograničena.

## Packet Switching (VoIP tehnologija)

VoIP koristi **packet switching** tehnologiju, koja omogućava prenos glasovnih podataka u malim paketima. Kada korisnik započne poziv, njegov glas se konvertuje u digitalne podatke koji se zatim pakuju u male podatke (pakete) i šalju putem internetske mreže. Ovi paketi putuju različitim putem kroz mrežu i ponovo se sastavljaju na odredištu. Time se eliminiše potreba za rezervacijom fiksnih linija, čime se omogućava efikasnija upotreba mrežnih resursa.

* **Princip rada**: Glas se digitalizuje i deli u male pakete koji putuju kroz mrežu. Na odredištu, ovi paketi se ponovo sastavljaju u originalnu glasovnu signalizaciju.
* **Prednosti**: Efikasno korišćenje mrežnih resursa, manji troškovi i mogućnost integracije drugih medija poput videa. Takođe, skalabilnost sistema je znatno veća nego kod tradicionalnih sistema.
* **Nedostaci**: Kvalitet poziva može zavisiti od kvaliteta internetske veze. Ako mreža ima lošu brzinu ili je nestabilna, može doći do prekida, kašnjenja ili lošeg kvaliteta zvuka.

## Osnovni protokoli za VoIP

VoIP komunikacija nije moguća bez upotrebe specifičnih protokola koji omogućavaju usmeravanje poziva i prenos podataka između uređaja. Dva osnovna protokola koji se koriste u VoIP komunikaciji su **Session Initiation Protocol (SIP)** i **Real-time Transport Protocol (RTP)**.

### Session Initiation Protocol (SIP)

SIP je protokol koji se koristi za uspostavljanje, upravljanje i završavanje sesija u komunikaciji. Na primer, SIP omogućava pozivaocu da inicira poziv, a zatim usmerava poziv prema odredištu. Kada se poziv uspostavi, SIP omogućava dodatne funkcionalnosti kao što su promena broja učesnika u pozivu, uključivanje videa ili slanje poruka.

* **Funkcija**: SIP upravlja signalizacijom između učesnika poziva, što znači da je odgovoran za pozivanje, usmeravanje i završavanje sesije.
* **Upotreba**: Osim za glasovne pozive, SIP se koristi i za video pozive i razmenu poruka.
* **Prednosti**: SIP je veoma fleksibilan protokol koji omogućava integraciju sa mnogim vrstama komunikacijskih servisa. Takođe, omogućava veoma jednostavno proširenje sistema.

### Real-time Transport Protocol (RTP)

Dok SIP upravlja signalizacijom, **RTP** je odgovoran za prenos glasovnih i video podataka u realnom vremenu. RTP obezbeđuje sinhronizaciju između paketa podataka i kontroliše njihov redosled, što je ključno za očuvanje kvaliteta komunikacije.

* **Funkcija**: RTP prenosi glasovne pakete, obezbeđujući minimalnu latenciju i visoki kvalitet zvuka.
* **Upotreba**: RTP se koristi za prenos glasovnih podataka i video snimaka tokom VoIP poziva, kao i za druge oblike real-time komunikacije.
* **Prednosti**: RTP omogućava visok kvalitet prenosa glasovnih i video podataka u realnom vremenu, sa minimalnim kašnjenjem.

## Sigurnosni izazovi u VoIP komunikaciji

Kao i u svim oblicima internetske komunikacije, VoIP pozivi suočavaju se sa nizom sigurnosnih izazova. Zbog svoje prirode, VoIP je podložan napadima poput presretanja podataka, **man-in-the-middle** (MITM) napada i zloupotrebe nešifrovanih veza. Ovi problemi predstavljaju ozbiljan rizik za privatnost i sigurnost korisnika.

**Problemi privatnosti**

Jedan od najvećih izazova kod VoIP komunikacije je sigurnost podataka. Pošto VoIP pozivi putuju preko interneta, postoji mogućnost da ih presretne treća strana. Na primer, napadači mogu presresti pozive ili poruke koje sadrže osetljive informacije, što može dovesti do ugrožavanja privatnosti korisnika.

* **Presretanje podataka**: Ako se pozivi ne šifruju, podaci mogu biti lako presretnuti i dešifrovani od strane napadača. Ovaj problem postaje još ozbiljniji kada se koriste neobezbeđene mreže kao što su javni Wi-Fi hot-spotovi.
* **Man-In-The-Middle napadi**: U ovom scenariju, napadač postaje posrednik između dva učesnika u komunikaciji, presrećući i menjajući poruke koje se šalju. MITM napadi mogu biti veoma opasni, jer omogućavaju napadačima da manipulišu porukama i podacima, čineći ih potpuno netačnim.

**Potreba za šifrovanjem**

Šifrovanje je od suštinske važnosti za zaštitu komunikacija od presretanja i napada. Većina modernih VoIP sistema koristi šifrovanje kako bi zaštitila pozive, a jedan od najčešće korišćenih protokola za šifrovanje je **TLS** (Transport Layer Security). Ovaj protokol omogućava siguran prenos podataka između uređaja i servera, čime se sprečava njihovo presretanje od strane neovlašćenih lica.

* **Šifrovanje poziva**: Pored TLS-a, VoIP sistemi koriste i **SRTP** (Secure Real-time Transport Protocol) za šifrovanje glasovnih podataka tokom prenosa. Ovaj protokol osigurava da glasovni podaci ostanu privatni i da ne mogu biti presretnuti ili izmenjeni tokom prenosa.

# Server.py

Ovaj fajl predstavlja **signalizacioni server** za aplikaciju *SecureTalk* – aplikaciju za sigurnu razmenu poruka, fajlova i glasovne/video pozive.  
Server je napisan u **Python-u**, koristi **FastAPI** za backend i **WebSocket** tehnologiju za komunikaciju u realnom vremenu.  
Glavna svrha ovog servera:

* Registracija korisnika i čuvanje njihovih javnih ključeva za enkripciju.
* Upravljanje sobama za razgovor (chat rooms).
* Slanje obaveštenja kada se neko pridruži ili napusti sobu.
* Prosleđivanje poruka, fajlova i signala za pozive između korisnika.
* Ne čuva istoriju poruka – server služi samo kao „posrednik“.

## Biblioteke I konfiguracija

|  |
| --- |
| **Izvorni kod 1.0: Biblioteke** |
| 1. import asyncio 2. import json 3. import uuid 4. from typing import Dict, Any, Set, Optional 5. from datetime import datetime, timezone 6. from fastapi import FastAPI, WebSocket, WebSocketDisconnect 7. from fastapi.responses import HTMLResponse 8. from fastapi.middleware.cors import CORSMiddleware 9. import uvicorn |

* asyncio – omogućava istovremeno rukovanje više konekcija bez blokiranja.
* json – serijalizacija podataka (pretvara Python objekte u JSON i obrnuto).
* uuid – kreiranje jedinstvenog identifikatora za svakog korisnika.
* typing – koristi se za tipizaciju podataka (čitljiviji kod).
* datetime – obezbeđuje tačno vreme u ISO formatu za poruke.
* fastapi i WebSocket – moderni Python web framework i WebSocket podrška.
* uvicorn – server za pokretanje FastAPI aplikacije.
* CORSMiddleware – omogućava pristup aplikaciji sa različitih domena (za testiranje i razvoj).

## Inicijalizacija aplikacije

|  |
| --- |
| **Izvorni kod 1.1: Inicijalizacija** |
| 1. app = FastAPI(title="SecureTalk Server") 2. app.add\_middleware( 3. CORSMiddleware, 4. allow\_origins=["\*"], 5. allow\_credentials=True, 6. allow\_headers=["\*"], 7. allow\_methods=["\*"], 8. ) |

**Objašnjenje**:

* Kreira se FastAPI aplikacija.
* CORS podešavanja omogućavaju da aplikacija radi i kada je frontend na drugom serveru/domeni (bitno za testiranje).

## Struktura podataka

|  |
| --- |
| **Izvorni kod 1.2: Strukture** |
| 1. USERS: Dict[str, Dict[str, Any]] = {} 2. ROOMS: Dict[str, Set[str]] = {} |

USERS je rečnik gde se za svakog korisnika čuva:

* username
* WebSocket konekcija (ws)
* ime sobe
* javni ključevi za enkripciju
* vreme pridruživanja

ROOMS čuva aktivne sobe i listu korisnika u njima.

|  |
| --- |
| **Izvorni kod 1.2b: Primer** |
| 1. USERS = { 2. „user123“: {„username“: „Milan“, „room“: „lobby“, „pub\_dh“: „...“, ...} 3. } 4. ROOMS = { 5. „lobby“: {„user123“, „user456“} 6. } |

## Pomoćne funkcije

|  |
| --- |
| **Izvorni kod 1.3: Vraća vreme u iso formatu** |
| 1. def now\_iso() -> str: 2. return datetime.now(timezone.utc).isoformat() |

|  |
| --- |
| **Izvorni kod 1.4: Šalje JSON poruku klijentu preko WebSocket-a.** |
| 1. async def send\_json(ws: WebSocket, payload: dict): 2. await ws.send\_text(json.dumps(payload, ensure\_ascii=False)) |

|  |
| --- |
| **Izvorni kod 1.5: Šalje poruku u sobi** |
| 1. async def notify\_room(room: str, payload: dict, exclude\_user\_id: Optional[str] = None): 2. # Šalje poruku svima u sobi osim jednom korisniku (npr. onom koji je poslao) |

Upravljanje sobama

|  |
| --- |
| **Izvorni kod 1.6: Dodavanje u sobu** |
| 1. def join\_room(user\_id: str, room: str): 2. if room not in ROOMS: 3. ROOMS[room] = set() 4. ROOMS[room].add(user\_id) 5. USERS[user\_id]["room"] = room |

def join\_room(user\_id: str, room: str):

* Definiše funkciju join\_room koja prima ID korisnika i ime sobe. Tipovi podataka str su samo type hinting (informacija da su oba parametra stringovi).

if room not in ROOMS:

* Proverava da li soba postoji u rečniku ROOMS. Ako ne postoji, napravi novu sobu.

ROOMS[room] = set()

* Ako soba nije postojala, kreira prazan set (skup) u kojem će biti korisnici. Set se koristi jer sprečava duplikate (isti korisnik ne može dva puta u istu sobu).

ROOMS[room].add(user\_id)

* Dodaje korisnika (po user\_id) u izabranu sobu.

USERS[user\_id]['room'] = room

* U podacima za korisnika čuva informaciju u kojoj je sobi trenutno.

Ukoliko soba ne postoji funkcija je kreira.

|  |
| --- |
| **Izvorni kod 1.7: Izlazak iz sobe** |
| 1. def leave\_room(user\_id: str): 2. room = USERS[user\_id].get("room") 3. if room and room in ROOMS: 4. ROOMS[room].discard(user\_id) 5. if not ROOMS[room]: 6. del ROOMS[room] 7. USERS[user\_id]["room"] = None |

room = USERS[user\_id].get("room")

* Uzima ime sobe u kojoj je korisnik trenutno.
* .get("room") vraća vrednost iz rečnika ili None ako nema.

if room and room in ROOMS:

* Proverava da li korisnik uopšte ima sobu i da li ta soba postoji u ROOMS.

ROOMS[room].discard(user\_id)

* Uklanja korisnika iz skupa korisnika u toj sobi.
* .discard() ne baca grešku ako korisnika nema, za razliku od .remove().

if not ROOMS[room]:

* Ako je soba sada prazna (skup je prazan), briše sobu.

del ROOMS[room]

* Briše unos za tu sobu iz rečnika ROOMS.

USERS[user\_id]["room"] = None

* Označava da korisnik više nije ni u jednoj sobi.

|  |
| --- |
| **Izvorni kod 1.8: Prikaz korisnika iz sobe** |
| 1. def room\_roster(room: str): 2. members = [] 3. for uid in ROOMS.get(room, set()): 4. u = USERS.get(uid) 5. if not u: 6. continue 7. members.append({ 8. "user\_id": uid, 9. "username": u.get("username"), 10. "pub\_dh": u.get("pub\_dh"), 11. "pub\_sign": u.get("pub\_sign"), 12. }) 13. return members |

Funkcija room\_roster služi za dobijanje liste korisnika koji se trenutno nalaze u određenoj sobi. Funkcija prima jedan parametar room, koji predstavlja ime sobe. Prvo se kreira prazna lista members u koju će se dodavati podaci o korisnicima. Zatim se koristi petlja for koja prolazi kroz sve ID-jeve korisnika (uid) u sobi. Sobe i njihovi korisnici čuvaju se u globalnom rečniku ROOMS, a ROOMS.get(room, set()) omogućava da se u slučaju da soba ne postoji vrati prazan skup, što sprečava greške. Za svaki ID korisnika, funkcija pokušava da dobije podatke korisnika iz globalnog rečnika USERS pomoću USERS.get(uid). Ako korisnik ne postoji (not u), petlja prelazi na sledeći ID. Ako korisnik postoji, kreira se rečnik sa osnovnim informacijama: user\_id (ID korisnika), username (korisničko ime), pub\_dh (javni ključ za Diffie-Hellman), i pub\_sign (javni ključ za potpis). Taj rečnik se dodaje u listu members. Na kraju, funkcija vraća listu members, što predstavlja kompletnu roster listu svih članova sobe sa njihovim ključnim podacima. Ovaj pristup omogućava efikasno prikupljanje i prikazivanje podataka o korisnicima u realnom vremenu, što je korisno za aplikacije poput chat soba ili sistema za kolaboraciju.

## Osnovna ruta

|  |
| --- |
| **Izvorni kod 1.9: Osnovna ruta** |
| 1. @app.get("/") 2. async def index(): 3. return HTMLResponse("<h3>SecureTalk signaling server radi ✅</h3>") |

**Objašnjenje:**

* @app.get("/") → ovo znači da je ovo HTTP GET ruta na početnoj adresi (/).
* async def index() → definisali smo asinhronu funkciju koja će se izvršiti kad neko poseti ovu rutu.
* return HTMLResponse(...) → server vraća jednostavnu HTML poruku da radi.

**Zašto ovako:**  
Ovo je najjednostavniji način da proverimo da server radi pre nego što počnemo sa WebSocket konekcijama.

## WebSocket endpoint

|  |
| --- |
| **Izvorni kod 1.10: Websocket endpoint** |
| 1. @app.websocket("/ws") 2. async def ws\_endpoint(ws: WebSocket): 3. await ws.accept() 4. user\_id = str(uuid.uuid4()) 5. USERS[user\_id] = { 6. "username": None, 7. "ws": ws, 8. "room": None, 9. "pub\_dh": None, 10. "pub\_sign": None, 11. "joined\_at": now\_iso(), 12. } |

**Objašnjenje:**

* @app.websocket("/ws") → ovo je WebSocket ruta, koristi se za stalnu vezu između klijenta i servera.
* await ws.accept() → prihvatamo WebSocket konekciju od klijenta.
* user\_id = str(uuid.uuid4()) → generišemo jedinstveni ID za svakog korisnika.
* USERS[user\_id] = {...} → pravimo zapis u globalnom USERS rečniku za praćenje podataka o korisniku:
  + username → ime korisnika
  + ws → WebSocket objekat za slanje/primanje poruka
  + room → koja soba trenutno
  + pub\_dh i pub\_sign → javni ključevi za sigurnu komunikaciju
  + joined\_at → vreme kada se korisnik pridružio

**Zašto ovako:**  
Ovo je osnova za praćenje korisnika u realnom vremenu i omogućava serveru da zna kome da šalje poruke ili datoteke.

## Slanje prvog “pozdrava”

|  |
| --- |
| **Izvorni kod 1.11: Slanje prvog „pozdrava“ korisniku** |
| 1. await send\_json(ws, {"type": "hello", "user\_id": user\_id, "server\_time": now\_iso()}) |

**Objašnjenje:**

* send\_json je funkcija (pretpostavljam definisana negde u kodu) koja šalje JSON podatke preko WebSocket-a.
* Šaljemo korisniku tip poruke "hello", njegov user\_id i trenutno vreme servera.

**Zašto:**  
Da klijent zna da je konekcija uspešno uspostavljena i da dobije svoj ID koji će koristiti u daljoj komunikaciji.

## Primanje i slanje poruka

|  |
| --- |
| **Izvorni kod 1.12: Glavna petlja za primanje i slanje poruka** |
| 1. buffer = "" 2. while True: 3. text = await ws.receive\_text() 4. buffer += text 5. data = json.loads(buffer) 6. buffer = "" 7. mtype = data.get("type") |

**Objašnjenje:**

* while True: → beskonačna petlja koja stalno čeka poruke od korisnika.
* text = await ws.receive\_text() → čeka sledeću poruku od korisnika.
* buffer += text → dodajemo primljeni tekst u buffer.
* data = json.loads(buffer) → pretvaramo JSON string u Python rečnik.
* mtype = data.get("type") → čitamo tip poruke, koji definiše šta korisnik želi (npr. "register", "join\_room", "message", itd.).

**Zašto:**  
WebSocket radi sa stalnom vezom, pa petlja omogućava da stalno slušamo klijenta i reagujemo na njegove zahteve.

## Registracija korisnika

|  |
| --- |
| **Izvorni kod 1.13: Registracija korisnika** |
| 1. if mtype == "register": 2. USERS[user\_id]["username"] = data.get("username") 3. USERS[user\_id]["pub\_dh"] = data.get("pub\_dh") 4. USERS[user\_id]["pub\_sign"] = data.get("pub\_sign") 5. await send\_json(ws, {"type": "register\_ok", "user\_id": user\_id}) |

**Objašnjenje:**

* Klijent šalje "register" poruku sa username, pub\_dh i pub\_sign.
* Server čuva te podatke u USERS[user\_id].
* Server potvrđuje registraciju slanjem "register\_ok" nazad.

**Zašto:**  
Da server zna identitet korisnika i njegove javne ključeve za šifrovanje poruka.

|  |
| --- |
| **Izvorni kod 1.14: Pridruživanje sobi ukoliko ona postoji** |
| 1. elif mtype == "join\_room": 2. room = data.get("room") or "lobby" 3. prev = USERS[user\_id].get("room") 4. if prev and prev != room: 5. leave\_room(user\_id) 6. join\_room(user\_id, room) |

**Objašnjenje:**

* "join\_room" → korisnik želi da se pridruži sobi.
* room = data.get("room") or "lobby" → ako nije specificirano, default soba je "lobby".
* Ako je korisnik već u nekoj sobi (prev), poziva se leave\_room da izađe.
* join\_room(user\_id, room) → dodaje korisnika u novu sobu.

**Zašto:**  
Ovo omogućava korisnicima da komuniciraju unutar različitih “sobnih grupa” i prati ko je u kojoj sobi.

|  |
| --- |
| **Izvorni kod 1.15: Notifikacija prilikom pristupanja sobi** |
| 1. await notify\_room(room, { 2. "type": "presence", 3. "event": "join", 4. "user": { 5. "user\_id": user\_id, 6. "username": USERS[user\_id]["username"], 7. "pub\_dh": USERS[user\_id]["pub\_dh"], 8. "pub\_sign": USERS[user\_id]["pub\_sign"], 9. } 10. }, exclude\_user\_id=user\_id) |

**Objašnjenje:**

* Obaveštavamo sve korisnike u sobi da je novi korisnik došao.
* "exclude\_user\_id=user\_id" → ne šaljemo notifikaciju korisniku koji je upravo ušao.

**Zašto:**  
Da svi vide da je novi korisnik prisutan i da mogu da šalju poruke ili datoteke.

|  |
| --- |
| **Izvorni kod 1.16: Notifikacija prilikom pristupanja sobi** |
| 1. elif mtype == "message": 2. cipher\_dict = data.get("cipher\_dict") or {} 3. if data.get("to") and data.get("cipher"): 4. cipher\_dict = {data.get("to"): data.get("cipher")} 5. for peer\_id, cipher in cipher\_dict.items(): 6. uinfo = USERS.get(peer\_id) 7. if uinfo: 8. payload = { 9. "type": "message", 10. "from": user\_id, 11. "cipher": cipher, 12. "msg\_id": data.get("msg\_id"), 13. "ttl\_ms": data.get("ttl\_ms"), 14. "ts": now\_iso(), 15. } 16. try: 17. await send\_json(uinfo["ws"], payload) 18. except Exception: 19. pass |

**Objašnjenje:**

* "message" → korisnik šalje šifrovanu poruku drugom korisniku.
* Podržava se više primalaca (cipher\_dict) i stari format (to + cipher).
* Poruka se šalje ciljnom korisniku preko send\_json.

**Zašto:**  
Omogućava **peer-to-peer šifrovanu komunikaciju** preko servera.

# Client.py

Na početku klijent fajla imamo veliki blok import linija. To su biblioteke I moduli koje ovaj program koristi

|  |
| --- |
| **Izvorni kod 2.0: Registracija korisnika** |
| 1. import asyncio 2. import json 3. import logging 4. import os 5. import secrets 6. import sys 7. import uuid 8. from pathlib import Path 9. from typing import Optional, Tuple 10. import aiohttp 11. import websockets 12. from cryptography.hazmat.primitives.asymmetric import dh 13. from cryptography.hazmat.primitives.serialization import ( 14. Encoding, 15. ParameterFormat, 16. PublicFormat, 17. load\_pem\_parameters, 18. load\_pem\_public\_key, 19. ) 20. from cryptography.hazmat.primitives import hashes, serialization 21. from cryptography.hazmat.primitives.asymmetric import ec 22. from cryptography.hazmat.primitives.kdf.hkdf import HKDF 23. from cryptography.hazmat.primitives.ciphers.aead import AESGCM |

**Objašnjenje:**

1. **asyncio – omogućava pisanje asinkronog koda (program može da radi više stvari istovremeno, npr. sluša poruke dok šalje nove).**
2. **json – rad sa JSON formatom za slanje podataka ka serveru i primanje podataka.**
3. **logging – za logovanje informacija i grešaka.**
4. **os, sys, secrets, uuid, Path – različite biblioteke za rad sa sistemom, generisanje ID-jeva, sigurnih ključeva, fajlovima, itd.**
5. **typing.Optional, typing.Tuple – za tipizaciju podataka (nije obavezno, ali pomaže za čitljivost).**
6. **aiohttp, websockets – biblioteke za komunikaciju sa serverom:**
   * **aiohttp – za HTTP zahteve (npr. registracija korisnika).**
   * **websockets – za real-time razmenu poruka (chat, signalizacija za pozive).**
7. **cryptography.\* – biblioteke za kriptografiju:**
   * **dh, ec – Diffie-Hellman i Elliptic Curve algoritmi za generisanje ključeva.**
   * **HKDF – za izvlačenje tajnog ključa iz razmene ključeva.**
   * **AESGCM – za enkripciju i dekripciju poruka.**

**Ovaj deo samo uveze alate koje koristimo. Ne sadrži logiku, ali je temelj svega jer odavde dolaze sve funkcije za mrežu, kriptografiju i asinhroni rad.**

## Globalne promenljive

|  |
| --- |
| **Izvorni kod 2.1: Globalne promenljive** |
| 1. SERVER\_URL = "http://localhost:8000" 2. WS\_URL = "ws://localhost:8000/ws" 3. DATA\_DIR = Path("./client\_data") 4. DATA\_DIR.mkdir(exist\_ok=True) 5. USER\_ID\_FILE = DATA\_DIR / "user\_id.txt" 6. PRIV\_KEY\_FILE = DATA\_DIR / "priv\_key.pem" 7. SIGN\_KEY\_FILE = DATA\_DIR / "sign\_key.pem" 8. logger = logging.getLogger("client") 9. logger.setLevel(logging.INFO) |

**Objašnjenje:**

* SERVER\_URL i WS\_URL – adrese servera:
  + HTTP server (SERVER\_URL) za registraciju i upload fajlova.
  + WebSocket server (WS\_URL) za razmenu poruka i poziva u realnom vremenu.
* DATA\_DIR – direktorijum gde se čuvaju korisnički podaci (ključevi, ID...).
* USER\_ID\_FILE, PRIV\_KEY\_FILE, SIGN\_KEY\_FILE – putanje do fajlova u kojima čuvamo podatke korisnika.
* logger – logger za ispis informacija i grešaka u konzolu (umesto običnih print() koristimo logging).

Ovaj deo **priprema okruženje**: zna gde se nalaze podaci korisnika, gde se čuvaju ključevi, i kako će se pratiti događaji.

## EncryptedAudioCall

**Šta radi EncryptedAudioCall?**

* Omogućava **razmenu zvuka uživo preko UDP-a** (brz protokol za zvuk/video).
* Svaki **paket zvuka se šifruje** pomoću **NaCl Box** (X25519 Diffie-Hellman + XSalsa20-Poly1305).
* Jednostavno rečeno: svaka poruka je mala šifrovana kutija, i samo tvoj sagovornik je može otvoriti.

|  |
| --- |
| **Izvorni kod 2.2: Init** |
| 1. def \_\_init\_\_(self, box: Box, local\_port: int = 0): 2. self.box = box 3. self.running = False 4. self.udp = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM) 5. self.udp.bind(('', local\_port)) 6. self.local\_port = self.udp.getsockname()[1] 7. self.peer = None 8. self.play\_stream = None 9. self.cap\_stream = None |

Kad napravimo objekat EncryptedAudioCall, on:

1. **Pamti Box objekat** (self.box) – to je naš šifrovani kanal (nastaje iz tvog privatnog i tuđeg javnog ključa).
2. **Priprema UDP socket** na nekom portu (0 = OS bira slobodan port).
3. Čuva port (self.local\_port) koji će slati drugoj strani.
4. Postavlja peer adresu i audio streamove na None.

|  |
| --- |
| **Izvorni kod 2.3: set\_peer** |
| 1. def set\_peer(self, host: str, port: int): 2. self.peer = (host, int(port)) |

Kada saznamo gde je peer (IP i port), ovde ga setujemo. To se obično radi kad dobijemo **call\_answer** od servera.

|  |
| --- |
| **Izvorni kod 2.4: Pokretanje poziva** |
| 1. def start(self): 2. if self.running: 3. return 4. self.running = True 5. # create audio streams 6. self.play\_stream = sd.OutputStream(samplerate=AUDIO\_RATE, channels=1, dtype=AUDIO\_DTYPE, blocksize=BLOCK) 7. self.play\_stream.start() 8. self.cap\_stream = sd.InputStream(samplerate=AUDIO\_RATE, channels=1, dtype=AUDIO\_DTYPE, blocksize=BLOCK, callback=self.\_capture\_cb) 9. self.cap\_stream.start() 10. threading.Thread(target=self.\_recv\_loop, daemon=True).start() |

Šta se dešava:

* Ako je već pokrenuto, ne radi ništa.
* Podiže sounddevice audio **OutputStream** (slušanje) i **InputStream** (mikrofon).
* callback=self.\_capture\_cb znači: kad mikrofon ima nove podatke, poziva se funkcija \_capture\_cb.
* Kreće recv\_loop u posebnoj niti – to je beskonačna petlja za primanje zvuka.

|  |
| --- |
| **Izvorni kod 2.5: Hvatanje i slanje zvuka** |
| 1. def \_capture\_cb(self, indata, frames, time, status): 2. if not self.running or not self.peer: 3. return 4. pcm = indata.tobytes() 5. nonce = np.random.bytes(24) 6. cipher = self.box.encrypt(pcm, nonce) 7. try: 8. self.udp.sendto(cipher, self.peer) 9. except Exception: 10. pass |

1. Dobijemo **PCM zvučne podatke** iz mikrofona (indata).
2. Generišemo **nonce** (nasumičan broj za ovaj paket).
3. self.box.encrypt(pcm, nonce) pravi šifrovani paket.
4. Šalješ ga preko UDP-a ka self.peer.

**Svaki paket je zasebno šifrovan i ima svoj nonce.**

|  |
| --- |
| **Izvorni kod 2.6: Primanje i puštanje zvuka** |
| 1. def \_recv\_loop(self): 2. while self.running: 3. try: 4. data, \_ = self.udp.recvfrom(8192) 5. pcm = self.box.decrypt(data) 6. audio = np.frombuffer(pcm, dtype=AUDIO\_DTYPE) 7. self.play\_stream.write(audio) 8. except Exception: 9. continue |

Proces:

1. Čeka da stigne UDP paket.
2. Dešifruje ga pomoću istog Box objekta.
3. Pretvara bajtove nazad u numpy array zvučnih podataka.
4. Pusti zvuk kroz zvučnike (self.play\_stream.write).

|  |
| --- |
| **Izvorni kod 2.7: Zaustavljanje poziva** |
| 1. def stop(self): 2. self.running = False 3. try: 4. if self.cap\_stream: 5. self.cap\_stream.stop(); self.cap\_stream.close() 6. if self.play\_stream: 7. self.play\_stream.stop(); self.play\_stream.close() 8. except Exception: 9. pass 10. try: 11. self.udp.close() 12. except Exception: 13. pass |

Zaustavlja sve audio streamove i zatvara socket.

**Ukratko: kako to funkcioniše**

1. Svaki korisnik generiše **Box** koristeći svoj privatni i tuđi javni ključ → zajednička tajna.
2. Kad poziv počne:
   * Kreira UDP socket i pokreće **mikrofon → šifruj → pošalji**.
   * Paralelno radi **primi → dešifruj → pusti**.
3. Sve ide u realnom vremenu, svaki UDP paket ima **nonce**, pa su paketi nezavisni.
4. Zato je komunikacija brza i sigurna.

## SecureTalkClient

|  |
| --- |
| **Izvorni kod 2.8: Init** |
| 1. class SecureTalkClient: 2. def \_\_init\_\_(self, username: str, room: str): 3. # identity 4. self.username = username 5. self.room = room or "lobby" 6. self.user\_id = None 7. # crypto 8. self.dh\_priv = PrivateKey.generate() 9. self.dh\_pub = self.dh\_priv.public\_key 10. self.sign\_priv = SigningKey.generate() 11. self.sign\_pub = self.sign\_priv.verify\_key 12. # peer state 13. self.peers\_dh = {} # user\_id -> PublicKey 14. self.peers\_sign = {} # user\_id -> VerifyKey 15. self.peer\_boxes = {} # user\_id -> Box (our\_priv, their\_pub) 16. self.username\_to\_uid = {} # username -> user\_id 17. # call state 18. self.call: EncryptedAudioCall | None = None 19. self.active\_call\_with: str | None = None 20. # transport queue (thread-safe) 21. self.sync\_send\_q = queue.Queue() 22. # GUI 23. self.root = tk.Tk() 24. self.root.title(f"SecureTalk — {self.username}") 25. self.\_build\_gui() 26. # background loop thread handle 27. self.ws\_thread = None |

**Objašnjenje za ceo blok**

Ovaj blok definiše **konstruktor klase SecureTalkClient** i priprema sve osnovne komponente klijenta:

1. **Identitet korisnika**
   * username i room pamte osnovne informacije o korisniku i sobi.
   * user\_id će biti dodeljen od servera nakon registracije.
2. **Kriptografski ključevi**
   * dh\_priv / dh\_pub → Diffie-Hellman ključevi za šifrovanje poruka.
   * sign\_priv / sign\_pub → ključevi za digitalni potpis poruka.
   * Omogućava **sigurnu end-to-end enkripciju** sa drugim korisnicima.
3. **Stanje peer-ova (drugih korisnika)**
   * peers\_dh → javni DH ključevi drugih korisnika.
   * peers\_sign → javni ključevi za verifikaciju potpisa.
   * peer\_boxes → objekti Box za šifrovanje/dešifrovanje sa svakim peer-om.
   * username\_to\_uid → preslikavanje username → user\_id, olakšava pronalaženje peer-ova.
4. **Stanje audio poziva**
   * call → objekat EncryptedAudioCall za aktivan poziv.
   * active\_call\_with → čuva user\_id osobe sa kojom trenutno pričamo.
5. **Transportna queue**
   * sync\_send\_q je **thread-safe red** za slanje poruka ka serveru.
   * Omogućava asinhronu i bezbednu razmenu poruka između GUI niti i mrežnog sloja.
6. **GUI**
   * self.root = tk.Tk() kreira glavni prozor.
   * self.\_build\_gui() gradi sve GUI elemente (tekstualni chat, dugmad, liste korisnika).
   * self.root.title() postavlja naslov prozora.
7. **Thread za WebSocket**
   * self.ws\_thread čuva referencu na nit koja će pokrenuti asinhroni WebSocket loop u pozadini.
   * Inicijalno je None jer nit još nije pokrenuta.

|  |
| --- |
| **Izvorni kod 2.9: Pravljenje GUI** |
| 1. def \_build\_gui(self): 2. frm\_top = ttk.Frame(self.root, padding=8) 3. frm\_top.pack(fill="x") 4. ttk.Label(frm\_top, text=f"Korisnik: {self.username}").pack(side="left") 5. self.room\_var = tk.StringVar(value=self.room) 6. ttk.Entry(frm\_top, textvariable=self.room\_var, width=20).pack(side="left", padx=8) 7. ttk.Button(frm\_top, text="Join sobu", command=self.join\_room).pack(side="left") 8. ttk.Button(frm\_top, text="Osveži članove", command=self.refresh\_members).pack(side="left", padx=6) 9. frm\_mid = ttk.Frame(self.root, padding=8) 10. frm\_mid.pack(fill="both", expand=True) 11. self.chat = tk.Text(frm\_mid, state="disabled", height=18, wrap="word") 12. self.chat.pack(side="left", fill="both", expand=True) 13. sb = ttk.Scrollbar(frm\_mid, command=self.chat.yview) 14. self.chat.configure(yscrollcommand=sb.set) 15. sb.pack(side="left", fill="y") 16. right = ttk.Frame(frm\_mid) 17. right.pack(side="left", fill="y", padx=8) 18. ttk.Button(right, text="Pozovi korisnika", command=self.call\_user).pack(fill="x", pady=2) 19. ttk.Button(right, text="Završi poziv", command=self.end\_call).pack(fill="x", pady=2) 20. ttk.Button(right, text="Pošalji fajl", command=self.send\_file).pack(fill="x", pady=2) 21. frm\_bottom = ttk.Frame(self.root, padding=8) 22. frm\_bottom.pack(fill="x") 23. self.msg\_var = tk.StringVar() 24. self.ttl\_var = tk.StringVar(value="0") 25. ttk.Entry(frm\_bottom, textvariable=self.msg\_var, width=60).pack(side="left", padx=(0,6)) 26. ttk.Label(frm\_bottom, text="TTL(ms):").pack(side="left") 27. ttk.Entry(frm\_bottom, textvariable=self.ttl\_var, width=8).pack(side="left", padx=(0,6)) 28. ttk.Button(frm\_bottom, text="Pošalji", command=self.send\_message).pack(side="left") 29. self.root.protocol("WM\_DELETE\_WINDOW", self.on\_close) |

Ova funkcija kreira **grafički interfejs (GUI)** aplikacije koristeći **Tkinter** i **ttk** (moderni stilizovani widgeti).  
GUI je podeljen u **tri glavna dela**: top, middle i bottom frame.

**Objašnjenje - top:**

* **frm\_top** → kontejner za elemente na vrhu (ime korisnika, soba, dugmad).
* **ttk.Label** → prikazuje tekst sa korisničkim imenom (Korisnik: Milan).
* **self.room\_var** → Tkinter StringVar koji čuva trenutno ime sobe i vezuje ga sa Entry poljem.
* **ttk.Entry** → polje gde korisnik može promeniti sobu.
* **Dugmad:**
  + "Join sobu" → poziva metodu self.join\_room, koja šalje zahtev serveru da se korisnik pridruži sobi.
  + "Osveži članove" → poziva self.refresh\_members, koji traži listu članova sobe.

Top frame služi za **kontrolu sobe i prikaz identiteta korisnika**.

**Objašnjenje - mid:**

* **frm\_mid** → glavni deo prozora gde se prikazuje chat i desni panel sa akcijama.
* **self.chat** → widget za prikaz poruka.
  + state="disabled" → korisnik ne može direktno tipkati u chat.
  + height=18 → visina widgeta u linijama.
  + wrap="word" → linije se prelamaju po rečima.
* **Scrollbar** → vertikalni scroll za chat.
* Povezan sa chatom preko yscrollcommand.

Ovo omogućava pregled svih poruka u sobi i automatsko skrolovanje.

**Objašnjenje - right:**

* **right** → frame desno od chat-a, vertikalno poravnat.
* Dugmad:
  + "Pozovi korisnika" → inicira audio poziv (self.call\_user).
  + "Završi poziv" → završava poziv (self.end\_call).
  + "Pošalji fajl" → šalje fajl drugim korisnicima (self.send\_file).
* fill="x" → dugme popunjava horizontalni prostor.
* pady=2 → razmak između dugmadi.

Desni panel je **kontrolni centar za akcije koje nisu tekstualne poruke**.

**Objašnjenje - bottom:**

* **frm\_bottom** → frame na dnu za unos poruka i TTL.
* **self.msg\_var** → veza sa input poljem za poruku.
* **self.ttl\_var** → opcionalni "Time To Live" u milisekundama, kontroliše koliko dugo poruka važi (peer-to-peer).
* **Dugme "Pošalji"** → šalje sadržaj self.msg\_var metodom self.send\_message.

Bottom frame je **glavni input korisnika**.

**Kontrola zatvaranja prozora:**

* Kada korisnik zatvori prozor (X dugme), poziva se metoda on\_close.
* Ova metoda osigurava da se:
  + Pozivi zaustave,
  + Peer-ovi obaveste da korisnik napušta sobu,
  + GUI se uredno zatvori.

**Zaključak za \_build\_gui**

* Funkcija kreira **kompletan GUI za chat aplikaciju**.
* Deli prozor na **tri logička dela**: kontrola sobe (top), chat + akcije (middle), unos poruka (bottom).
* Svaki widget je povezan sa **metodama klase**, što omogućava interakciju sa serverom i peer-ovima.
* Koristi **Tkinter + ttk** radi modernijeg izgleda i jednostavnog layout-a.

|  |
| --- |
| **Izvorni kod 2.10: Log** |
| 1. def log(self, text: str): 2. self.chat.config(state="normal") 3. self.chat.insert("end", f"{text}\n") 4. self.chat.see("end") 5. self.chat.config(state="disabled") |

**Objašnjenje:**

* **Cilj:** prikazivanje poruka u chat widgetu (self.chat).
* state="normal" → omogućava unos/izmenu teksta u Text widgetu.
* insert("end", f"{text}\n") → dodaje novu liniju na kraj chata.
* see("end") → automatski skroluje na poslednju poruku.
* state="disabled" → ponovo onemogućava ručno tipkanje u chatu.

Funkcija služi za **sigurno i jednostavno logovanje svih događaja i poruka** u GUI.

|  |
| --- |
| **Izvorni kod 2.11: ensure\_box** |
| 1. def \_ensure\_box(self, peer\_id: str): 2. if peer\_id in self.peer\_boxes: 3. return self.peer\_boxes[peer\_id] 4. pub = self.peers\_dh.get(peer\_id) 5. if not pub: 6. return None 7. box = Box(self.dh\_priv, pub) 8. self.peer\_boxes[peer\_id] = box 9. return box |

* **Cilj:** kreira ili vraća **NaCl Box** (simetrični kripto kanal) za komunikaciju sa datim peer-om.
* **Box** koristi **Diffie-Hellman privatni ključ (self.dh\_priv)** i **peer javni ključ (pub)** da bi kreirao **sigurnu sesiju**.
* Rezultat se kešira u self.peer\_boxes da se ne kreira više puta.
* Ako peer javni ključ nije poznat, vraća None.

Ovo je temelj za **šifrovanje i dešifrovanje poruka**.

|  |
| --- |
| **Izvorni kod 2.12: \_encrypt\_for** |
| 1. def \_encrypt\_for(self, peer\_id: str, plaintext: bytes) -> str: 2. box = self.\_ensure\_box(peer\_id) 3. if not box: 4. raise RuntimeError("Peer key missing") 5. nonce = np.random.bytes(24) 6. ct = box.encrypt(plaintext, nonce) 7. return base64.b64encode(ct).decode() |

* Šifruje plaintext za datog peer-a.
* **Koraci:**
  1. Dobije Box za peer-a.
  2. Generiše **nonce** od 24 bajta (za jedinstvenu enkripciju).
  3. Šifruje poruku (box.encrypt).
  4. Kodira rezultat u **base64** da može da se šalje preko WebSocket-a.

Svaka poruka je **jedinstveno šifrovana** i sigurna od presretanja.

|  |
| --- |
| **Izvorni kod 2.13: \_decrypt\_from** |
| 1. def \_decrypt\_from(self, peer\_id: str, b64cipher: str) -> bytes: 2. box = self.\_ensure\_box(peer\_id) 3. if not box: 4. raise RuntimeError("No session with peer") 5. ct = base64.b64decode(b64cipher.encode()) 6. pt = box.decrypt(ct) 7. return pt |

* Dešifruje poruku primljenu od peer-a.
* **Koraci:**
  1. Dobije Box za peer-a.
  2. Base64 dekodira ciphertext.
  3. Dešifruje pomoću Box-a.
  4. Vraća originalne bajtove (bytes).

Ovo je suprotnost \_encrypt\_for – omogućava **siguran prijem poruka**.

|  |
| --- |
| **Izvorni kod 2.14: WebSocket loop** |
| 1. async def \_ws\_loop(self): 2. async with websockets.connect(SERVER\_WS, max\_size=20\_000\_000) as ws: 3. self.ws = ws |

* **Pokreće konekciju ka serveru** koristeći websockets i asyncio.
* max\_size=20\_000\_000 → dozvoljava velike poruke (fajlovi, pozivi).
* async with automatski zatvara konekciju na kraj.

|  |
| --- |
| **Izvorni kod 2.15: Prijava i registracija** |
| 1. msg = json.loads(await ws.recv()) 2. self.user\_id = msg["user\_id"] 3. self.log(f"• Povezano (user\_id={self.user\_id})") 4. await ws.send(json.dumps({ 5. "type": "register", 6. "username": self.username, 7. "pub\_dh": base64.b64encode(bytes(self.dh\_pub)).decode(), 8. "pub\_sign": base64.b64encode(bytes(self.sign\_pub)).decode() 9. })) |

* Server šalje početni hello sa user\_id.
* Klijent registruje svoje **javno DH i sign ključeve** kod servera.
* Na taj način server može da **distribuira javne ključeve peer-ovima**.

|  |
| --- |
| **Izvorni kod 2.16: Pridruživanje sobi** |
| 1. await ws.send(json.dumps({"type": "join\_room", "room": self.room})) |

* Klijent odmah ulazi u željenu sobu.
* Server kasnije šalje članove sobe i njihove ključeve.

|  |
| --- |
| **Izvorni kod 2.17: SenderTask** |
| 1. async def sender\_task(): 2. while True: 3. payload = await loop.run\_in\_executor(None, self.sync\_send\_q.get) 4. try: 5. await ws.send(json.dumps(payload)) 6. except Exception: 7. break |

* **Thread-safe queue** (sync\_send\_q) omogućava slanje podataka sa GUI-a u async WebSocket loop.
* run\_in\_executor koristi **blokirajući .get()** bez zamrzavanja event loop-a.
* Svaka stavka u queue-u se šalje serveru kao JSON.

|  |
| --- |
| **Izvorni kod 2.18: ReciverTask** |
| 1. async def receiver\_task(): 2. while True: 3. raw = await ws.recv() 4. data = json.loads(raw) 5. await self.\_handle\_server(data) |

* Prima podatke od servera.
* Prosleđuje ih funkciji \_handle\_server koja **obradi različite tipove događaja** (poruke, fajlovi, pozivi, prisustvo).

|  |
| --- |
| **Izvorni kod 2.19: Pokretanje obe Task funkcije** |
| 1. await asyncio.gather(sender\_task(), receiver\_task()) |

* Obezbeđuje **paralelno izvršavanje slanja i prijema** poruka.
* Sender task šalje poruke iz queue-a.
* Receiver task obrađuje poruke koje dolaze sa servera.

## Handle Server

|  |
| --- |
| **Izvorni kod 2.20: HandleServer** |
| 1. async def \_handle\_server(self, data: dict): 2. t = data.get("type") 3. if t == "room\_joined": 4. room = data.get("room") 5. members = data.get("members", []) 6. self.room = room 7. # rebuild maps 8. self.username\_to\_uid.clear() 9. for m in members: 10. uid = m["user\_id"] 11. uname = m.get("username") 12. if uname: 13. self.username\_to\_uid[uname] = uid 14. # store peer public keys if not us 15. if uid != self.user\_id and m.get("pub\_dh"): 16. try: 17. self.peers\_dh[uid] = PublicKey(base64.b64decode(m["pub\_dh"])) 18. self.peers\_sign[uid] = VerifyKey(base64.b64decode(m["pub\_sign"])) 19. # remove old box to force re-derive 20. self.peer\_boxes.pop(uid, None) 21. except Exception: 22. pass 23. self.log(f"• Ušao si u sobu '{room}'. Članova: {len(members)}") 24. # show roster 25. if members: 26. names = [m.get("username", m["user\_id"][:6]) for m in members] 27. self.log("Roster: " + ", ".join(names)) 28. elif t == "presence": 29. ev = data.get("event") 30. u = data.get("user", {}) 31. uid = u.get("user\_id") 32. uname = u.get("username") 33. if ev == "join": 34. if uname: 35. self.username\_to\_uid[uname] = uid 36. if uid != self.user\_id and u.get("pub\_dh"): 37. try: 38. self.peers\_dh[uid] = PublicKey(base64.b64decode(u["pub\_dh"])) 39. self.peers\_sign[uid] = VerifyKey(base64.b64decode(u["pub\_sign"])) 40. self.peer\_boxes.pop(uid, None) 41. except Exception: 42. pass 43. self.log(f"✅ {uname or uid[:6]} se priključio.") 44. elif ev == "leave": 45. if uname: 46. self.username\_to\_uid.pop(uname, None) 47. self.peers\_dh.pop(uid, None) 48. self.peers\_sign.pop(uid, None) 49. self.peer\_boxes.pop(uid, None) 50. if self.active\_call\_with == uid: 51. self.\_end\_call\_local() 52. self.log(f"❌ {uname or uid[:6]} je napustio sobu.") 53. elif t == "message": 54. frm = data.get("from") 55. cipher = data.get("cipher") 56. try: 57. pt = self.\_decrypt\_from(frm, cipher) 58. text = pt.decode("utf-8") 59. except Exception: 60. text = "⚠️ [Neuspešno dešifrovanje]" 61. # show message: if username known map it 62. uname = None 63. for name, uid in self.username\_to\_uid.items(): 64. if uid == frm: 65. uname = name; break 66. label = uname if uname else frm[:6] 67. self.log(f"{label}: {text}") 68. elif t == "file": 69. frm = data.get("from") 70. cipher\_meta = data.get("cipher\_meta") 71. cipher\_chunks = data.get("cipher\_chunks", []) 72. try: 73. box = self.\_ensure\_box(frm) 74. meta = json.loads(box.decrypt(base64.b64decode(cipher\_meta)).decode()) 75. fname = meta.get("filename", f"file\_{now\_iso()}") 76. chunk\_b64 = cipher\_chunks[0] 77. filedata = box.decrypt(base64.b64decode(chunk\_b64)) 78. # save to current dir with unique name 79. outname = f"recv\_{fname}" 80. with open(outname, "wb") as f: 81. f.write(filedata) 82. uname = None 83. for name, uid in self.username\_to\_uid.items(): 84. if uid == frm: 85. uname = name; break 86. self.log(f"📎 Primljen fajl od {uname or frm[:6]}: {outname}") 87. except Exception: 88. self.log("⚠️ Primljen fajl - neuspešno dešifrovanje") 89. elif t in ("call\_invite", "call\_answer", "call\_end", "call\_ice"): 90. # normalized: server forwards "from" as user\_id 91. if t == "call\_invite": 92. frm = data.get("from") 93. udp\_host = data.get("udp\_host", None) or data.get("udp\_host") or data.get("udp\_host") 94. udp\_port = data.get("udp\_port") 95. # automatically accept for MVP (could prompt user) 96. box = self.\_ensure\_box(frm) 97. if not box: 98. self.log("⚠️ Poziv: nemamo kripto sesiju sa peer-om.") 99. return 100. # start call and set peer 101. self.call = EncryptedAudioCall(box) 102. self.call.start() 103. # set peer address (we expect peer to send their host/port) 104. host = udp\_host or self.\_local\_ip\_guess() 105. if udp\_port: 106. self.call.set\_peer(host, udp\_port) 107. self.local\_udp\_port = self.call.local\_port 108. self.active\_call\_with = frm 109. self.log(f"📞 Poziv primljen od {self.\_username\_of(frm) or frm[:6]}") 110. # respond 111. self.sync\_send\_q.put({ 112. "type": "call\_answer", 113. "to": frm, 114. "udp\_host": self.\_local\_ip\_guess(), 115. "udp\_port": self.local\_udp\_port, 116. "ok": True 117. }) 118. elif t == "call\_answer": 119. frm = data.get("from") 120. ok = data.get("ok", False) 121. if ok and self.call and self.active\_call\_with == frm: 122. host = data.get("udp\_host") or self.\_local\_ip\_guess() 123. port = data.get("udp\_port") 124. # set peer and start audio if not yet 125. self.call.set\_peer(host, port) 126. self.log(f"📞 Poziv povezan sa {self.\_username\_of(frm) or frm[:6]}") 127. else: 128. self.log("❌ Poziv odbijen ili greška") 129. elif t == "call\_end": 130. frm = data.get("from") 131. if self.active\_call\_with == frm: 132. self.\_end\_call\_local() 133. self.log("🔴 Poziv završen (peer).") 134. elif t == "room\_members": 135. members = data.get("members", []) 136. names = [m.get("username", m.get("user\_id")[:6]) for m in members] 137. self.log(f"• Članovi sobe ({data.get('room')}): {', '.join(names)}") 138. elif t == "error": 139. self.log(f"⚠️ Server error: {data.get('message')}") |

### room\_joined

**Šta radi:**

* Server javlja da je klijent uspešno ušao u sobu.
* Čisti sve stare mape (username\_to\_uid) i učitava nove članove sobe.
* Sprema javne ključeve drugih korisnika (peers\_dh, peers\_sign) za enkripciju.
* Briše stare peer\_boxes da bi se sesija obnovila.
* Loguje poruku i ispisuje roster (spisak korisnika).

### Presence

**Šta radi:**

* Obrada događaja kada neko uđe ili izađe iz sobe.
* Ako je **join** → dodaje korisnika u mape i sprema njegove javne ključeve.
* Ako je **leave** → briše korisnika iz mape i završava lokalni poziv ako je bio aktivan.
* Loguje događaj u GUI.

### Message

**Šta radi:**

* Prima šifrovanu poruku od servera.
* Pokušava da dešifruje koristeći peer\_boxes.
* Ako ne može da dešifruje → prikazuje grešku.
* Ispisuje poruku u chat-u sa imenom pošiljaoca (ako je poznato).

### File

**Šta radi:**

* Prima fajl od drugog korisnika.
* Dešifruje metapodatke (ime fajla) i sadržaj fajla.
* Snima fajl lokalno.
* Loguje da je fajl primljen ili grešku ako nije moguće dešifrovati.

### Ostale funkcije

**call\_invite / call\_answer / call\_end**

* **Pozivi su VoIP preko UDP-a, enkriptovani.**

**call\_invite**

* Prima poziv, kreira EncryptedAudioCall, startuje lokalno.
* Postavlja peer adresu i port.
* Automatski prihvata poziv i šalje odgovor serveru.

**call\_answer**

* Ako peer prihvati poziv, konektuje audio kanal.
* Ako odbije, loguje grešku.

**call\_end**

* Ako peer završi poziv, prekida lokalni poziv.

**room\_members**

* Loguje članove sobe (osvežavanje roastera).

**g) error**

* Loguje greške servera.

## Helper

|  |
| --- |
| **Izvorni kod 2.21: Tražimo username** |
| 1. def \_username\_of(self, uid: str): 2. for name, id\_ in self.username\_to\_uid.items(): 3. if id\_ == uid: 4. return name 5. return None |

**Šta radi:**

* Pomoćna funkcija koja vraća username iz user\_id.
* Ako ne postoji → vraća None.

|  |
| --- |
| **Izvorni kod 2.22: Određivanje IP** |
| 1. def \_local\_ip\_guess(self) -> str: 2. s = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM) 3. try: 4. s.connect(("8.8.8.8", 80)) 5. ip = s.getsockname()[0] 6. except Exception: 7. ip = "127.0.0.1" 8. finally: 9. s.close() 10. return ip |

**Šta radi:**

* Pokušava da odredi lokalnu IP adresu mašine.
* Ako ne uspe → vraća localhost 127.0.0.1.

## GUI akcije

|  |
| --- |
| **Izvorni kod 2.23: Pridruživanje sobi** |
| 1. def join\_room(self): 2. room = self.room\_var.get().strip() or "lobby" 3. # clear local caches for new room 4. self.peers\_dh.clear(); self.peers\_sign.clear(); self.peer\_boxes.clear(); self.username\_to\_uid.clear() 5. self.sync\_send\_q.put({"type": "join\_room", "room": room}) 6. self.log(f"• Tražim pristup sobi '{room}'...") |

**Objašnjenje:**

* Ova metoda omogućava korisniku da se pridruži chat sobi.
* self.room\_var.get().strip() preuzima ime sobe iz GUI polja, a ako nema imena koristi "lobby".
* Pre ulaska u novu sobu, briše sve lokalne podatke o korisnicima i kriptografskim ključevima:
  + self.peers\_dh.clear() → briše javne Diffie-Hellman ključeve drugih korisnika.
  + self.peers\_sign.clear() → briše javne ključeve za potpisivanje.
  + self.peer\_boxes.clear() → briše kriptografske sesije.
  + self.username\_to\_uid.clear() → briše mapu korisničkih imena i njihovih ID-jeva.
* Zatim u **sinhronizovani red (sync\_send\_q)** ubacuje zahtev za pridruživanje sobi ({"type": "join\_room", "room": room}).
* Na kraju loguje poruku u chat: „Tražim pristup sobi“.

**Zaključak:**  
Ova metoda resetuje lokalni keš ključeva i inicira komunikaciju sa serverom da se korisnik pridruži željenoj sobi.

|  |
| --- |
| **Izvorni kod 2.24: Lista clanova sobe** |
| 1. def refresh\_members(self): 2. self.sync\_send\_q.put({"type": "room\_members"}) |

**Objašnjenje:**

* Jednostavna metoda koja traži od servera listu članova trenutne sobe.
* Ubacuje zahtev {"type": "room\_members"} u red za slanje.
* Na serveru će to pokrenuti odgovor sa ažuriranom listom korisnika.

**Zaključak:**  
Koristi se za osvežavanje liste članova u realnom vremenu.

|  |
| --- |
| **Izvorni kod 2.25: Slanje poruke svim clanovima sobe** |
| 1. def send\_message(self): 2. msg = self.msg\_var.get().strip() 3. if not msg: 4. return 5. # create per-peer ciphertext mapping 6. cipher\_dict = {} 7. for uname, uid in self.username\_to\_uid.items(): 8. if uid == self.user\_id: 9. continue 10. try: 11. cipher = self.\_encrypt\_for(uid, msg.encode("utf-8")) 12. cipher\_dict[uid] = cipher 13. except Exception: 14. # skip peers without keys 15. continue 16. if not cipher\_dict: 17. self.log("⚠️ Nema drugih članova ili nema kripto ključeva.") 18. return 19. payload = { 20. "type": "message", 21. "cipher\_dict": cipher\_dict, 22. "msg\_id": str(uuid.uuid4()), 23. "ttl\_ms": int(self.ttl\_var.get() or 0) 24. } 25. # show our own message immediately 26. self.log(f"Ja: {msg}") 27. self.msg\_var.set("") 28. self.sync\_send\_q.put(payload) |

**Objašnjenje:**

* Metoda za slanje tekstualne poruke svim članovima sobe.
* msg = self.msg\_var.get().strip() uzima tekst iz GUI polja.
* Ako je poruka prazna, odmah prekida.
* Kreira prazan rečnik cipher\_dict u koji stavlja **šifrovane poruke** za svakog korisnika pojedinačno:
  + Poziva \_encrypt\_for(uid, msg.encode()) da se poruka šifrira za svakog korisnika posebnim ključem.
  + Ako nema ključa za nekog korisnika → preskače ga.
* Ako nema ni jednog primatelja (cipher\_dict prazan) → loguje upozorenje.
* Kreira payload sa:
  + "type": "message" → tip poruke.
  + "cipher\_dict" → šifrovani sadržaj po korisnicima.
  + "msg\_id" → jedinstveni ID poruke.
  + "ttl\_ms" → opcionalni „time-to-live“ za automatsko brisanje poruke.
* Loguje poruku u chat: Ja: ... i prazni polje za unos poruke.
* Na kraju dodaje payload u red sync\_send\_q za slanje serveru.

**Zaključak:**  
Svaka poruka se **posebno šifrira za svakog korisnika** koristeći njegov javni ključ. Ovo povećava sigurnost, jer niko osim ciljanog korisnika ne može pročitati poruku.

|  |
| --- |
| **Izvorni kod 2.26: Slanje fajlova** |
| 1. def send\_file(self): 2. path = filedialog.askopenfilename() 3. if not path: 4. return 5. try: 6. with open(path, "rb") as f: 7. data = f.read() 8. except Exception as e: 9. messagebox.showerror("Greška", str(e)) 10. return 11. # prepare per-peer encryption 12. for uname, uid in self.username\_to\_uid.items(): 13. if uid == self.user\_id: 14. continue 15. box = self.\_ensure\_box(uid) 16. if not box: 17. continue 18. cipher\_meta = base64.b64encode(box.encrypt(json.dumps({"filename": path.split("/")[-1]}).encode(), np.random.bytes(24))).decode() 19. cipher\_chunk = base64.b64encode(box.encrypt(data, np.random.bytes(24))).decode() 20. payload = {"type": "file", "cipher\_dict": {uid: {"cipher\_meta": cipher\_meta, "cipher\_chunks": [cipher\_chunk]}}} 21. self.sync\_send\_q.put(payload) 22. self.log(f"📎 Poslat fajl: {path.split('/')[-1]}") |

**Objašnjenje:**

* Ova metoda šalje fajl svim članovima sobe, koristeći enkripciju.
* Otvara dijalog za izbor fajla, učitava ga u data.
* Za svakog korisnika kreira:
  + cipher\_meta → šifrovane metapodatke (ime fajla).
  + cipher\_chunk → šifrovani sadržaj fajla.
* Svakom korisniku šalje **sopstveno šifrovanu kopiju fajla**.
* Na kraju u chat log dodaje poruku „Poslat fajl: ...“.

**Zaključak:**  
Ova metoda koristi istu logiku kao poruke — fajl je enkriptovan za svakog korisnika posebno.

|  |
| --- |
| **Izvorni kod 2.27: Pokretanje audio poziva** |
| 1. def call\_user(self): 2. uname = simpledialog.askstring("Poziv", "Unesite username:").strip() if self.root else None 3. if not uname: 4. return 5. target\_uid = self.username\_to\_uid.get(uname) 6. if not target\_uid: 7. self.log(f"⚠️ Korisnik {uname} nije dostupan.") 8. return 9. box = self.\_ensure\_box(target\_uid) 10. if not box: 11. self.log("⚠️ Nema kripto sesije sa ciljanim korisnikom.") 12. return 13. # start local audio call 14. self.call = EncryptedAudioCall(box) 15. self.call.start() 16. self.local\_udp\_port = self.call.local\_port 17. self.active\_call\_with = target\_uid 18. # send invite using target\_uid (server accepts uid or username) 19. payload = {"type": "call\_invite", "to": target\_uid, "udp\_host": self.\_local\_ip\_guess(), "udp\_port": self.local\_udp\_port} 20. self.sync\_send\_q.put(payload) 21. self.log(f"📞 Poziv upućen ka {uname}") |

**Objašnjenje:**

* Pokreće **p2p audio poziv** sa određenim korisnikom.
* Traži korisničko ime (uname).
* Proverava da li korisnik postoji i da li imamo kriptografski „box“ sa njim.
* Ako je sve u redu:
  + Kreira objekat EncryptedAudioCall za enkriptovan glasovni poziv.
  + Startuje poziv (self.call.start()).
  + Pamti lokalni UDP port za prenos glasa.
  + Kreira i šalje serveru zahtev za poziv (call\_invite).

**Zaključak:**  
Ova funkcija gradi osnovu za **end-to-end enkriptovane VoIP pozive**.

|  |
| --- |
| **Izvorni kod 2.28: Zavrsetak poziva** |
| 1. def end\_call(self): 2. if not self.active\_call\_with: 3. return 4. payload = {"type": "call\_end", "to": self.active\_call\_with} 5. self.sync\_send\_q.put(payload) 6. self.\_end\_call\_local() 7. self.log("🔴 Poziv završen (ja).") 8. def \_end\_call\_local(self): 9. if self.call: 10. try: 11. self.call.stop() 12. except Exception: 13. pass 14. self.call = None 15. self.active\_call\_with = None |

**Objašnjenje:**

* end\_call šalje serveru zahtev da završi poziv.
* Poziva \_end\_call\_local koji zaustavlja lokalni audio stream i briše podatke o pozivu.
* Nakon prekida, loguje poruku u chat.

**Zaključak:**  
Osigurava da i lokalna i udaljena strana znaju da je poziv završen.

|  |
| --- |
| **Izvorni kod 2.29: Zatvaranje aplikacije** |
| 1. def on\_close(self): 2. if messagebox.askokcancel("Izlaz", "Zatvori klijent?"): 3. try: 4. if self.call: 5. self.call.stop() 6. except Exception: 7. pass 8. # attempt to leave room before exit 9. try: 10. self.sync\_send\_q.put({"type": "leave\_room"}) 11. except Exception: 12. pass 13. self.root.destroy() |

**Objašnjenje:**

* Poziva se kada korisnik zatvori aplikaciju.
* Prvo traži potvrdu (askokcancel).
* Ako postoji aktivan poziv, zaustavlja ga.
* Pokušava da serveru pošalje leave\_room zahtev.
* Uništava GUI prozor (self.root.destroy()).

**Zaključak:**  
Pravilno zatvara aplikaciju, prekida veze i obaveštava server.

|  |
| --- |
| **Izvorni kod 2.30: Pokretanje aplikacije** |
| 1. def run(self): 2. # start websocket thread (runs asyncio loop) 3. def \_thread\_run(): 4. asyncio.run(self.\_ws\_loop()) 5. self.ws\_thread = threading.Thread(target=\_thread\_run, daemon=True) 6. self.ws\_thread.start() 7. # start GUI mainloop 8. self.root.mainloop() |

**Objašnjenje:**

* Glavna metoda za pokretanje aplikacije.
* Pokreće asinhroni WebSocket loop u posebnoj niti (ws\_thread).
* Zatim startuje Tkinter GUI petlju (self.root.mainloop()).

**Zaključak:**  
Omogućava **paralelno pokretanje GUI-ja i mrežne komunikacije**.

## Main

|  |
| --- |
| **Izvorni kod 2.31: Main** |
| 1. if \_\_name\_\_ == "\_\_main\_\_": 2. uname = simpledialog.askstring("SecureTalk", "Unesite username:") 3. if not uname: 4. uname = f"user\_{uuid.uuid4().hex[:6]}" 5. room = simpledialog.askstring("Soba", "Unesite sobu (prazno = lobby)") or "lobby" 6. client = SecureTalkClient(uname, room) 7. client.run() |

**Objašnjenje:**

1. **if \_\_name\_\_ == "\_\_main\_\_":**
   * Ova linija osigurava da se kod ispod nje izvršava **samo ako se fajl pokrene direktno** (npr. python client.py), a ne kada se fajl importuje u drugi fajl.
   * Standardna praksa u Python-u za organizaciju koda.
2. **uname = simpledialog.askstring("SecureTalk", "Unesite username:")**
   * Otvara dijalog prozor u kojem korisnik unosi svoje **korisničko ime**.
   * simpledialog.askstring(title, prompt) je Tkinter funkcija koja kreira mali GUI prozor sa naslovom *"SecureTalk"* i tekstom *"Unesite username:"*.
   * Rezultat unosa (string) se čuva u promenljivoj uname.
3. **if not uname:**
   * Ako korisnik klikne "Cancel" ili ostavi polje prazno, uname će biti None ili prazan string.
   * U tom slučaju se generiše **nasumično korisničko ime**.
4. **uname = f"user\_{uuid.uuid4().hex[:6]}"**
   * Generiše jedinstveni identifikator koristeći biblioteku uuid.
   * uuid.uuid4() stvara slučajni UUID (128-bitni broj).
   * .hex[:6] uzima samo prvih 6 karaktera tog broja za jednostavnost.
   * Na primer, korisničko ime može biti user\_a3f91c.
   * Ovo je **fallback opcija** ako korisnik ne unese ime.
5. **room = simpledialog.askstring("Soba", "Unesite sobu (prazno = lobby)") or "lobby"**
   * Traži od korisnika da unese ime sobe za razgovor.
   * Ako korisnik ništa ne unese, automatski se koristi "lobby" kao podrazumevana soba.
   * Ovo omogućava kreiranje različitih chat soba ili pridruživanje postojećim.
6. **client = SecureTalkClient(uname, room)**
   * Kreira instancu klase SecureTalkClient sa korisničkim imenom i sobom.
   * Ova klasa sadrži sve što je potrebno za povezivanje na server, razmenu kriptovanih poruka, vođenje liste korisnika, GUI, itd.
   * Konstruktor \_\_init\_\_ (koji smo već objašnjavali) postavlja sve osnovne parametre i GUI.
7. **client.run()**
   * Pokreće glavnu logiku aplikacije (verovatno otvara GUI prozor i uspostavlja vezu sa serverom).
   * U klasi SecureTalkClient, metoda run() obično:
     + Povezuje klijenta sa signaling serverom (preko WebSocket-a),
     + Pokreće **background thread** za komunikaciju,
     + Pokreće Tkinter GUI glavnu petlju (root.mainloop()).

# Literatura

FastAPI Documentation. (n.d.). Retrieved from https://fastapi.tiangolo.com/

Python WebSocket Documentation (websockets library). (n.d.). Retrieved from https://websockets.readthedocs.io/

Cryptography.io - Python Cryptography Library Documentation. (n.d.). Retrieved from https://cryptography.io/en/latest/

Python Official Documentation. (n.d.). Retrieved from https://docs.python.org/3/

Tkinter — Python Interface to Tcl/Tk. Python Docs. (n.d.). Retrieved from https://docs.python.org/3/library/tkinter.html

PyNaCl Documentation — Python binding to libsodium. (n.d.). Retrieved from https://pynacl.readthedocs.io/

JSON — JavaScript Object Notation. Python Docs. (n.d.). Retrieved from https://docs.python.org/3/library/json.html

Asyncio — Asynchronous I/O. Python Docs. (n.d.). Retrieved from https://docs.python.org/3/library/asyncio.html

Secure Messaging Protocols Overview. (n.d.). OWASP. Retrieved from https://owasp.org/

RFC 8446 - The Transport Layer Security (TLS) Protocol Version 1.3. IETF. Retrieved from https://datatracker.ietf.org/doc/html/rfc8446