АКАДЕМИЈА ТЕХНИЧКО-УМЕТНИЧКИХ СТРУКОВНИХ СТУДИЈА БЕОГРАД

ОДСЕК ВИСОКА ШКОЛА ЕЛЕКТРОТЕХНИКЕ И РАЧУНАРСТВА

Ђонић Ненад

C++ Апликација за размену порука

- завршни рад -

Београд, септембар 2024.

Кандидат: **Ђонић Ненад**

Број индекса: **РТ-42/19**

Студијски програм: **Рачунарска техника**

Тема: ***C*++ Апликација за размену порука**

Основни задаци:

1. Опис коришћених технологија клијентске и среверске стране.

2. Имплементација *C*++ апликације за размену порука.

3. Приказ корисничког интерфејса.

Ментор:

Београд, септембар 2024 године.

_____

др Перица Штрбац, проф. ВИШЕР

РЕЗИМЕ:

Овај рад представља развој апликације за размену порука засноване на *C++*. Клијентска страна користи *wxWidgets GUI* библиотеку и *TCP* протокол за мрежну комуникацију, док на страни сервера користи *TCP* за мрежне везе и *SQLite* базу података за складиштење корисничких података. Апликација омогућава размену порука у реалном времену између корисника, са функцијама као што су регистрација корисника, пријављивање, слање порука и преглед. Кориснички интерфејс је дизајниран да пружи корисничко искуство, омогућавајући интуитивну навигацију и интеракцију. Овај рад показује ефикасну употребу *C++*-а и релевантних библиотека за изградњу робусне и скалабилне апликације за размену порука.

Кључне речи: : *C++*, апликација за размену порука, *wxWidgets*, *TCP*, *SQLite*, клијент-сервер архитектура.

ABSTRACT:

This paper presents the development of a C++-based messaging application. The client-side utilizes the wxWidgets GUI library and TCP protocol for network communication, while the server-side employs TCP for network connections and SQLite database for user data storage. The application enables real-time messaging between users, with features such as user registration, login, message sending, and viewing. The user interface is designed to provide a user-friendly experience, allowing for intuitive navigation and interaction. This work demonstrates the effective use of C++ and relevant libraries for building a robust and scalable messaging application.

Key words: C++, messaging application, wxWidgets, TCP, SQLite, client-server architecture.

САДРЖАЈ:

1. УВОД

Овај рад представља развој апликације за размену порука засноване на *C++* која омогућава комуникацију у реалном времену између више клијената преко централног сервера. Апликација користи *wxWidgets GUI* библиотеку за кориснички интерфејс на страни клијента и *SQLite* за управљање базом података на страни сервера. *TCP* протокол се користи за комуникацију између клијената и сервера.

1. Преглед технологије пружа свеобухватно истраживање технологија које се користе у апликацији за размену порука. Удубиће се у кључне карактеристике и функционалности *C++*, *wxWidgets*, *SQLite* и *TCP* протокола, расправљајући о њиховој подобности за захтеве апликације.

2. Архитектура система описује архитектонски дизајн апликације за размену порука. Он ће представити детаљан преглед компоненти система, укључујући клијента, сервер и базу података, и истражити комуникационе протоколе и структуре података који се користе за размену и складиштење порука.

3. Имплементација на страни клијента се фокусира на развој апликације на страни клијента користећи *wxWidgets*. Покриваће дизајн и имплементацију корисничког интерфејса, укључујући функције као што су пријављивање, регистрација, слање и примање порука, и дискутовати о интеграцији клијента са *TCP* комуникационим слојем.

4. **Имплементација на страни сервера** истражује имплементацију апликације на страни сервера. Покриваће улогу сервера у руковању клијентским везама, управљању аутентификацијом корисника, обради порука и интеракцији са *SQLite* базом података. У поглављу ће се такође расправљати о архитектури сервера и имплементацији *TCP* комуникационог слоја.

5. Дизајн и управљање базом података се бави дизајном и управљањем *SQLite* базом података коју користи апликација за размену порука. Разговараће се о шеми базе података, укључујући табеле.

6. Безбедносна разматрања баве се безбедносним аспектима апликације за размену порука, разматрајући потенцијалне рањивости и стратегије ублажавања.

7. Закључак Овај рад је успешно развио апликацију за размену порука засновану на *C++* која омогућава комуникацију у реалном времену између више клијената. Апликација користи *wxWidgets*, *SQLite* и *TCP* протокол за пружање корисничке услуге размене порука. Будућа побољшања могу укључивати додавање функција као што су групна ћаскања, дељење датотеке и интеграција са другим комуникационим платформама. Будућа побољшања би требало да имају акценат и на повећању сигурноснх аспеката апликације за размену порука.

2. ПРЕГЛЕД ТЕХНОЛОГИЈЕ

Апликација за размену порука користи пажљиво одабрану комбинацију технологија за пружање робусне и ефикасне комуникационе платформе. У основи апликације лежи програмски језик *C++*, свестран и моћан алат за изградњу сложених софтверских система. Могућности ниског нивоа *C++*-а, објектно оријентисане функције и обимна стандардна библиотека га оспособљавају да се носи свим неопходним аспектима апликације за размену порука, укључујући мрежну комуникацију, интеракције корисничког интерфејса и управљање подацима.

2.1. ПРОГРАМСКИ ЈЕЗИК *C++*

C++ је програмски језик опште намене познат по својој ефикасности, флексибилности и моћним објектно оријентисаним карактеристикама. Његове могућности ниског нивоа чине га веома погодним за задатке програмирања на нивоу система, као што су мрежна комуникација и интеракције са базом података. Обимне стандардне библиотеке *C++*-а пружају богат скуп алата и функционалности за различите задатке програмирања, укључујући структуре података, алгоритме и операције уноса/излаза.

2.2. *WXWIDGETS GUI* БИБЛИОТЕКА

Апликација за размену порука користи *wxWidgets GUI* библиотеку за креирање интуитивних и корисничких интерфејса за десктоп и мобилне платформе. *wxWidgets* је скуп алата за више платформи који омогућава програмерима да направе апликације које изгледају исто на *Windows*, *macOS*, *linux* и другим оперативним системима. Нуди широк спектар контрола корисничког интерфејса, укључујући прозоре, дугмад, текстуална поља, мените и дијалоге, пружајући доследно корисничко искуство на различитим платформама. Архитектура вођена догађајима *wxWidgets* га чини веома погодним за изградњу интерактивних апликација као што су клијенти за размену порука.

2.3. *SQLITE* СИСТЕМ ЗА УПРАВЉАЊЕ БАЗАМА ПОДАТАКА

За складиштење и управљање корисничким подацима и историјом порука, апликација за размену порука користи *SQLite* систем за управљање базом података. *SQLite* је лагана, уграђена база података која је идеална за мале апликације због своје једноставности, поузданости и лакоће коришћења. Пружа интерфејс заснован на *SQL*-у за креирање, испитивање и ажурирање база података, што олакшава интеракцију са ускладиштеним подацима. Компактна величина и минималне зависности *SQLite*-а чине га погодним за примену у оквиру сервера апликације за размену порука.

2.4. *TCP* ПРОТОКОЛ

TCP (*Transmission Control Protocol*) служи као основа за комуникацију између клијената апликације за размену порука и сервера. *TCP* је поуздан протокол оријентисан на везу који гарантује испоруку пакета података, обезбеђује правилно секвенцирање и обезбеђује механизме за проверу грешака. Његове карактеристике поузданости и контроле тока чине га погодним за успостављање стабилне и поуздане везе између клијентског и серверског дела апликације за размену порука.

3. АРХИТЕКТУРА СИСТЕМА

Ово поглавље представља детаљан преглед архитектуре система за апликацију за размену порука. Архитектура се састоји од три примарне компоненте: клијента, сервера и базе података. Ове компоненте комуницирају једна са другом кроз добро дефинисане комуникационе протоколе и структуре података како би олакшале размену и складиштење порука.

3.1. АРХИТЕКТУРА НА СТРАНИ КЛИЈЕНТА

Архитектура на страни клијента је одговорна за обезбеђивање корисничког интерфејса и руковање корисничким интеракцијама. Састоји се од следећих компоненти:

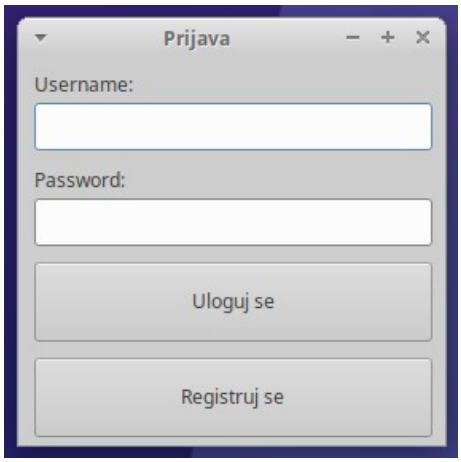
- **Кориснички интерфејс:** Кориснички интерфејс је развијен коришћењем *wxWidgets GUI* библиотеке и пружа визуелно привлачан и интуитиван интерфејс за кориснике за интеракцију са апликацијом за размену порука. Укључује функције као што су пријављивање, регистрација, преглед листе корисника, слање и примање порука.

- **Мрежна комуникација:** Клијент комуницира са сервером користећи *TCP* протокол. Успоставља сталну везу са сервером и шаље и прима податке преко ове везе.

- **Обрада протокола**: Клијент прима податке од сервера и обрађује их на одговарајући начин. Ажурира кориснички интерфејс за приказ долазних података и омогућава корисницима да шаљу поруке другим корисницима.

3.1.1. Кориснички интерфејс

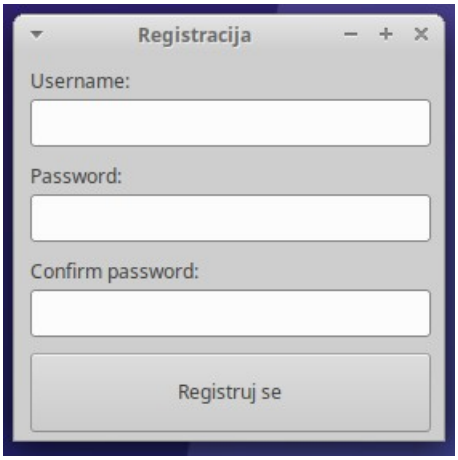
Апликација за размену порука има интерфејс прилагођен кориснику изграђен са *wxWidgets GUI* библиотеком. Овај интерфејс омогућава корисницима да се лако пријаве, региструју за налог, прегледају листу корисника, шаљу поруке и примају поруке од других.



Слика 3.1 – Прозор за пријаву

Прозор за пријаву приказан на слици 3.1 пружа једноставан и интуитиван интерфејс за кориснике да унесу своје акредитиве и приступе апликацији за размену порука. Прозор укључује следеће елементе:

- **Поље корисничког имена:** Поље за текст у које корисници могу да унесу своје корисничко име (*Username*).
- **Поље лозинке:** Поље за лозинку у које корисници могу да унесу своју лозинку (*Password*).
- **Дугме за пријаву:** Дугме које покреће процес пријаве (Uloguj se).
- **Дугме за регистрацију:** Дугме за отварање прозора за регистрацију (Registruj se).



Слика 3.2 – Прозор за регистрацију

Прозор за регистрацију приказан на слици 3.2 омогућава новим корисницима да креирају налоге. Прозор укључује следеће елементе:

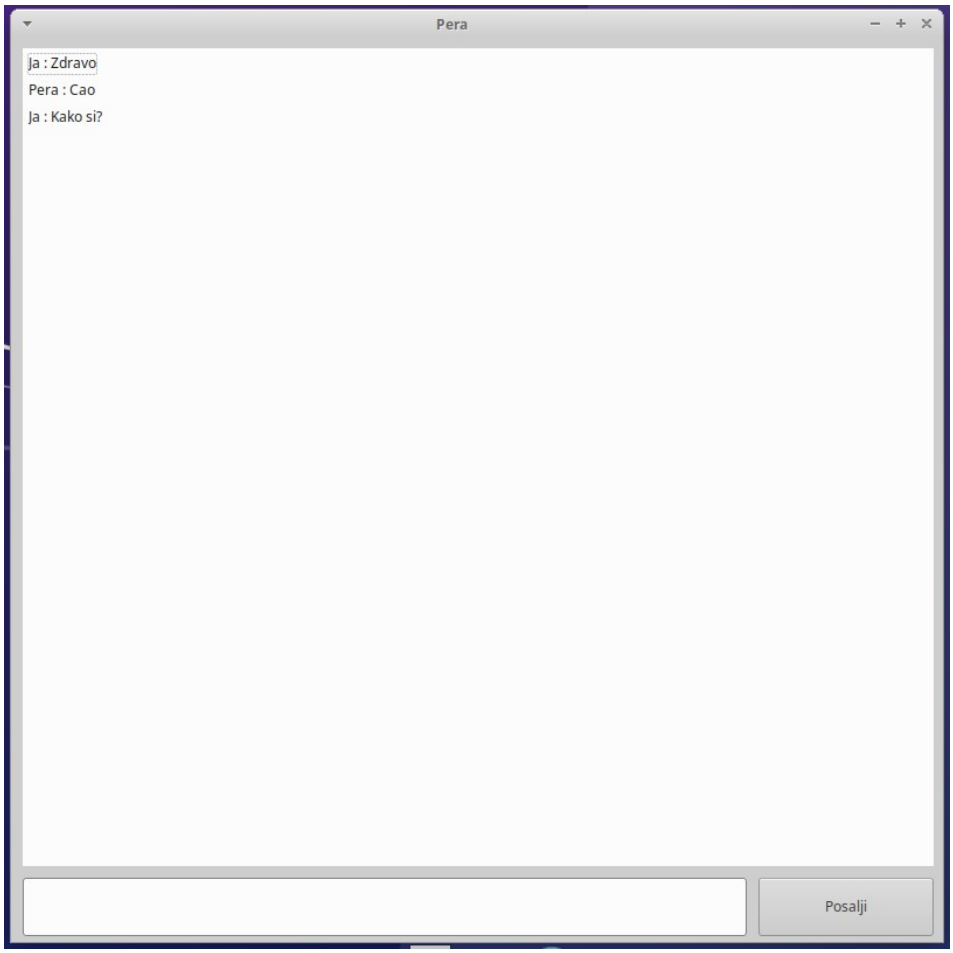
- **Поље корисничког имена:** Поље за текст у којем корисници могу да унесу своје будуће корисничко име (*Username*).
- **Поље за лозинку:** Поље за лозинку у којем корисници могу да унесу своју будућу лозинку (*Password*).
- **Поље за потврду лозинке:** Поље за потврду унете лозинке (*Confirm password*).
- **Дугме за регистрацију:** Дугме које покреће процес регистрације (Registruj se).

Слика 3.3 – Прозор за приказ листе корисника

Прозор листе корисника приказан на слици 3.3 приказује листу регистрованих корисника са којима корисник може да комуницира. Прозор укључује следеће елементе:

Листа корисника: Приказује листу која приказује корисничка имена регистрованих корисника.

Дугме за одјаву: Дугме које покреће процес одјављивања (Izloguj se).



Слика 3.4 – Прозор за размену порука

Прозор за ћаскање приказан на слици 3.4 је место где корисници могу да комуницирају једни са другима. Укључује следеће елементе:

- **Историја ћаскања:** Приказ листе који приказује историју порука између тренутног корисника и изабраног партнера за ћаскање.
- **Поље за унос поруке:** Поље за текст у које корисници могу да унесу своје поруке.
- **Дугме за слање:** Дугме које шаље унету поруку партнеру за ћаскање (Posalji).

3.1.2. Мрежна комуникација

Клијентска апликација се ослања на *TCP* (*Transmission Control Protocol*) да би успоставила поуздану и трајну везу са сервером. Снага *TCP*-а лежи у његовој природи оријентисаној на конекцији, која обезбеђује сигурну испоруку пакета и имплементиране механизме за проверу грешака. Ово гарантује тачан пренос порука између клијента и сервера, минимизирајући ризик од оштећења или губитка података.

Након покретања апликације, клијент иницира везу са сервером. Размена података се затим одвија преко прилагођеног протокола дизајнираног посебно за

апликацију за размену порука. Овај протокол дефинише формат поруке и ток комуникације, обезбеђујући да и клијент и сервер размењују информације на прецизно дефинисан начин, одређене структуре порука. Касније ћемо дубље ући у специфичности овог протокола.

Ако било која непредвиђена грешка поремети *TCP* везу, протокол апликације прецизно прекида комуникацију. Ово спречава недоследности података или неочекивано понашање које произилази из непоуздане везе. Коначно, веза се прекида када корисник намерно затвори клијентску апликацију.

3.2. АРХИТЕКТУРА НА СТРАНИ СЕРВЕРА

Архитектура апликације за размену порука на страни сервера служи као окосница система, руковање клијентским везама, управљање аутентификацијом корисника, обрада порука и интеракција са базом података. Осигурава несметан рад апликације за размену порука, и пружа сигурну и поуздану комуникацијону платформу за кориснике.

3.2.1. Управљање везама са клијентима

Архитектура на страни сервера игра кључну улогу у управљању клијентским везама, обезбеђујући комуникацију и ефикасну испоруку порука. Сервер слуша на унапред одређеном *port*-у, активно тражећи долазне захтеве за повезивање од клијената. По пријему везе, сервер покреће нови *thread* за самостално руковање специфичном интеракцијом клијента, омогућавајући истовремену комуникацију са више клијената.

3.2.2. Регистрација корисника

Сервер спроводи безбедан процес регистрације да би заштитио апликацију за размену порука. Нови корисници морају да обезбеде важеће акредитиве који су у складу са унапред дефинисаним стандардима проткола апликације за размену порука. Након успешне верификације, сервер креира нови налог у бази података, омогућавајући кориснику приступ апликацији.

3.2.3. Аутентификација корисника

Да би заштитио апликацију за размену порука и спречио неовлашћени приступ, сервер користи робустан механизам за аутентификацију корисника. Када клијент покуша да се пријави, сервер верификује њихове акредитиве према унапред дефинисаном протоколу дизајнираном за ову апликацију. Овај протокол описује специфичне кораке укључене и аутентификацију корисника, укључујући размену неопходних информација и валидацију акредитива. Када су акредитиви клијента успешно верификовани, сервер им даје приступ апликацији за размену порука и складишти информације које корисник шаље серверу у бази података за будућу употребу.

3.2.4. Управљање листом корисника

Након успешног пријављивања, клијент шаље захтев серверу и сервер одговара са листом свих регистрованих корисника из базе података и шаље је клијенту. Ово омогућава клијенту да прикаже листу корисника кориснику, омогућавајући му да одабере корисника са којим ће започети комуникацију у виду размене порука.

3.2.5. Обрада порука

Сервер делује као централно чвориште за размену порука, примање порука од клијената, складиштење у бази података и прослеђивање примаоцима када добије захтев за прослеђивање. Када клијент пошаље поруку, сервер је обрађује, обезбеђујући правилно форматирање и валидацију. Порука се затим чува у бази података. Након тога, сервер одређује одговарајућег примаоца на основу одредишта поруке и прослеђује поруку њему када пошаље захтев за преузимање те поруке.

3.2.6. Интеракција са базом података

Архитектура на страни сервера се у великој мери ослања на *SQLite* базу података за ефикасно управљање и складиштење корисничких информација које се односе на корисничке налоге, историју порука и друге битне аспекте апликације за размену порука. Служећи као централизовано складиште, база података омогућава брзо проналажење и манипулацију подацима, обезбеђујући беспрекорну функционалност апликације. Сервер користи *SQL query* за извршавање различитих операција базе података, укључујући креирање нових записа, преузимање постојећих података, и брисање застарелих уноса. Ова робусна интеграција базе података омогућава серверу да одржи интегритет података, олакша ефикасну обраду порука и пружи поуздано корисничко искуство.

3.3. АРХИТЕКТУРА ПРОТОКОЛА

Апликација за размену порука користи прилагођени протокол за комуникацију између клијента и сервера. Овај протокол је дизајниран да обрађује различите аспекте функционалности апликације, укључујући пријављивање, регистрацију, добијање листе корисника, слање порука и преузимање порука.

Протокол се састоји од шест подпротокола, од којих је сваки одговоран за одређени задатак:

- **0 – *End Communication* :** Овај подпротокол се користи за прекид комуникацијске сесије између клијента и сервера.
- **1 – *Login* :** Овај подпротокол се користи за клијенте да се аутентификују на серверу користећи своје акредитиве.
- **2 – *Registration* :** Овај подпротокол се користи за нове кориснике да креирају налоге на серверу.
- **3 - *User List Request* :** Овај подпротокол користе клијенти да захтевају листу свих регистрованих корисника.
- **4 - *Message Send* :** Овај подпротокол користе клијенти за слање порука другим корисницима.

- **5 - *Get Message List* :** Овај подпротокол користе клијенти за преузимање листе порука које су разменили са одређеним корисником.
- **6 - *Get Last Message* :** Овај подпротокол користе клијенти за преузимање поруке коју су задњу разменили са одређеним корисником.

3.3.1. Подпротокол за прекид комуникације

Подпротокол за завршетак комуникације (0 – *End Communication*) се користи за прекид везе између клијента и сервера. Клијент може покренути ову акцију. Када клијент пошаље захтев са кодом за завршетак комуникације, сервер обрађује поруку и затвара везу. Овај подпротокол се обично користи када се клијентска апликација затвара.

Tabela 3.1 – Подпротокол за прекид комуникације

0	Client		Server
1	000_	→	
2		←	000_

У првом кораку подпротокол за прекид комуникације приказаном у табел 3.1 клијент шаље код 0 серверу. Након чега сервер враћа код 0 у другом кораку клијенту ако је дошло до успешног завршетка протокола, ако није, враћа код 1.

3.3.2. Подпротокол за пријаву

Подпротокол за пријаву (1 – *Login*) се користи за аутентификацију клијената на серверу. Клијент шаље поруку са типом пријаве, укључујући корисничко име и лозинку. Сервер проверава акредитиве у односу на сачуване корисничке информације у бази података. Ако су акредитиви важећи, сервер шаље клијенту успешан одговор на пријаву, омогућавајући му да настави са коришћењем апликације за размену порука. Ако су акредитиви неважећи, сервер шаље одговор на грешку при пријављивању, указујући да корисник треба да пружи исправне акредитиве.

Tabela 3.2 – Подпротокол за пријаву

0	Client		Server
1	001_Username_Password_	→	
2		←	000_

У првом кораку подпротокол за пријаву приказаном у табели 3.2 , клијент шаље поруку која садржи код, корисничко име (*username*) и шифру (*password*). Код који клијент шаље серверу је 1. Ако су акредитиви корисника важећи, сервер враћа код 0, у супротном враћа код 1.

3.3.3. Подпротокол за регистрацију

Подпротокол за регистрацију (2 – *Registration*) се користи за нове кориснике да креирају налоге на серверу. Клијент шаље поруку са типом регистрације, укључујући жељено корисничко име и лозинку. Сервер проверава дате акредитиве како би се уверио да испуњавају услове за регистрацију. Ако су акредитиви исправни и корисничко име није већ заузето, сервер креира нови кориснички налог у бази података и шаље клијенту успешан одговор на регистрацију. Ако су акредитиви неважећи или је корисничко име већ у употреби, сервер шаље одговор на грешку регистрације.

Tabela 3.3 – Подпротокол за регистрацију

0	Client		Server
1	002_Username_Password_	→	
2		←	000_

У првом кораку подпротокол за регистрацију приказаном у табели 3.3 , клијент шаље поруку која садржи код, корисничко име (*username*) и шифру (*password*). Код који клијент шаље серверу је 2. Ако су акредитиви које клијент шаље серверу исправни и корисничко име није већ заузето, сервер креира нови кориснички налог и шаље поруку са кодом 0, у супротном ако акредитиви нису одговарајући враћа код 1.

3.3.4. Подпротокол за преузимање листе корисника

Подпротокол за преузимање листе корисника (3 - *User List Request*) клијенти користе за преузимање листе свих регистрованих корисника са сервера. Клијент шаље поруку са типом захтева за листу корисника. Сервер обрађује захтев, преузима листу корисника из базе података и шаље листу клијенту у одговарајућем облику. Клијент тада може да прикаже листу корисника кориснику, омогућавајући му да комуницира са другим корисницима унутар апликације за размену порука.

Tabela 3.4 – Подпротокол за преузимање листе корисника

0	Client		Server
1	003_	→	
2		←	000_2_
3	000_	→	
4		←	000_id1_Username1_
5	000_	→	
6		←	000_id2_Username2_
7	000_	→	
8		←	000_

У првом кораку подпротокола за преузимање листе корисника приказаном у табели 3.4 , клијент шаље захтев за преузимање листе корисника серверу, кодом 3. У другом кораку сервер одговара са поруком која има код 0 који означава да нема грешака и садржи број регистрованих корисника (што је у горе приказаном случају 2). У трећем кораку клијент шање поруку серверу са кодом 0 што означава да нема грешака. У четвртом кораку сервер шање поруку клијенту која садржи код 0 који означава да нема грешака, *id* корисника и корисничко име (*username*). Пети и шести корак су поновљени трећи и четврти, само за следећег корисника. У седмом кораку клијент шаље поруку са кодмом 0 која означава да је клијент успешно преузео листу корисника. У осмом кораку сервер шаље поруку са кодмом 0 која означава да није било грешака ни на серверској страни.

3.3.5. Подпротокол за слање поруке

Подпротокол за слање поруке (4 - *Message Send*) клијенти користе за слање порука другим корисницима. Клијент шаље поруку са типом слања поруке, укључујући *id* примаоца и садржај поруке. Сервер обрађује поруку, чува је у бази података.

Tabela 3.5 – Подпротокол за слање поруке

0	Client		Server
1	004_id_message_	→	
2		←	000_

У првом кораку подпротокола за слање поруке приказаном у табели 3.5 , клијент шаље поруку која садржи код , *id* корисника коме се шаље порука и сам текст поруке. Код који клијент шаље серверу је 4. Ако је порука у адекватном формату и унета је без

грешака у базу података, сервер враћа одговор са кодмом 0 који означава да није било грешака.

3.3.6. Подпротокол за преузимање порука

Подпротокол за добијање листе порука (5 - *Get Message List*) клијенти користе да добију листу порука које су размењили са одређеним корисником. Клијент шаље поруку за преузимање листе порука, која садржи код зе преузимање листе порука и *id* корисника са којим је размењио поруке. Сервер обрађује захтев, преузима поруке из базе података и шаље листу клијенту у одговарајућем облику. Клијент тада може да прикаже поруке кориснику, омогућавајући му да види своју историју послатих и примљених порука са одређеним корисником.

Tabela 3.6 – Подпротокол за добијање листе порука

0	Client		Server
1	005_id_	→	
2		←	000_2_
3	000_	→	
4		←	000_m-id_message_from_
5	000_	→	
6		←	000_m-id_message_from_
7	000_	→	
8		←	000_

У првом кораку подпротокола за преузимање листе порука приказаном у табели 3.6 , клијент шаље поруку која садржи код 5 за преузимање порука и *id* корисника са којим је размењио поруке које би хтео да преузме.У другом кораку сервер одговара кориснику са поруком која садржи код 0 који означава да није било грешака и број порука. У трећем кораку клијент шаље поруку са кодом 0 који означава да није још увек било грешака. У четвртом кораку сервер шаље поруку клијенту која садржи код 0, *id* поруке, текст поруке и поље које означава да ли је поруку послало клијент који преузима листу порука или је том клијенту послата порука. Пети и шести корак су поновљени трећи и четври за другу поруку. У седмом кораку клијент шаље поруку серверу са кодом 0 означавајући да није било грешака на клијенској страни и да је листа успешно преузета. У осмом кораку сервер шање поруку клијенту са кодом 0, означавајући да није било грешака на серверској страни, да је листа успешно послата и да се подпротокол за преузимање порука завршава.

3.3.7. Подпротокол за преузимање последње поруке

Подпротокол за добијање последње поруке (6 - *Get Last Message*) клијент користи да добије последњу поруку коју је разменио са одређеним корисником. Клијент шаље поруку за преузимање последње поруке, која садржи код зе преузимање последње поруке и *id* корисника са којим је разменио поруку. Сервер обрађује захтев, преузима поруку из базе података и шаље поруку клијенту у одговарајућем облику. Клијент тада може да упореди поруку са последњом поруком из листе, и дода је у листу за приказ уколико је дошло до промене.

Tabela 3.7 – Подпротокол за добијање последње поруке

0	Client		Server
1	006_id_	→	
2		←	000_m-id_message_from_

У првом кораку подпротокола за преузимање последње поруке приказаном у табели 3.7 , клијент шаље поруку која садржи код и *id* корисника са којим је разменио поруку. Код који клијент шаље серверу је 6. Сервер обрађује захтев, преузима последњу поруку коју је корисник који захтева поруку разменио са корисником са *id*-јем наведеним у захтеву и шаље поруку клијенту у одговарајућем формату. Формат у коме се порука шаље клијенту је исти као и у подпротоколу за преузимање листе порука.

4. ИМПЛЕМЕНТАЦИЈА НА СТРАНИ КЛИЈЕНТА

Имплементација апликације за размену порука на страни клијента састоји се од две основне групе компоненти:

Компоненте комуникације : Ове компоненте су одговорне за руковање комуникационим протоколом и обезбеђивање интеракције са сервером. Они управљају успостављањем и одржавањем *TCP* везе, као и слањем и примањем порука према дефинисаном протоколу.

Компоненте *GUI*-а : Развијене користећи *wxWidgets GUI* библиотеку, ове компоненте обезбеђују визуелни интерфејс за апликацију за размену порука. Они управљају интеракцијама корисника и приказују информације. Компоненте *GUI* су одговорне за креирање различитих прозора и контрола унутар апликације, као што су прозор за пријаву, прозор за регистрацију, прозор листе корисника и прозор за размену порука.

4.1. *MAIN.CPP FILE*

Код иницијализује апликацију, креира комуникациони објекат и приказује прозор за пријаву. Класа комуникације је одговорна за руковање везом са сервером и управљање комуникационим протоколом. Класа *LogInFrame* обезбеђује кориснички интерфејс за прозор за пријаву, омогућавајући корисницима да унесу своје акредитиве и покушају да се пријаве или оду у прозор за регистрацију.

```
1.    #include <wx/wx.h>
2.    #include "gui_component/login/login.h"
3.
4.    class Aplikacija : public wxApp
5.    {
6.          public:
7.          virtual bool OnInit();
8.    };
9.
10.   wxIMPLEMENT_APP(Aplikacija);
11.
12.   bool Aplikacija::OnInit()
13.   {
14.         communication* server;
15.         server = new communication();
16.
17.         LogInFrame *frame = new LogInFrame(server);
18.         frame->Show(true);
19.
20.         return true;
21.   }
```

4.2. *COMMUNICATION_OBJECT.H FILE*

C++ код наведен у наставку овог подпоглавља дефинише комуникациону класу за апликацију за размену порука, обухватајући логику за интеракцију са сервером и руковање различитим комуникационим задацима. Класа одржава *socket* везу са сервером, користећи *TCP* протокол за поуздану комуникацију. Пружа методе за регистрацију, пријављивање, слање порука, примање порука и преузимање листе регистрованих корисника. Класа која користи прилагођени протокол, осигуравајући да се подаци преносе и примају исправно. Поред тога, класа управља корисничким информацијама, чува и преузима корисничке податке из базе података. Све у свему, овај код служи као основна компонента апликације за размену порука, олакшавајући комуникацију између клијента и сервера и омогућавајући основне функционалности апликације.

```
1.  #ifndef COMMUNICATION_H
2.  #define COMMUNICATION_H
3.
4.  #include <iostream>
5.  #include <cstring>
6.  #include <unistd.h>
7.  #include <arpa/inet.h>
8.  #include <sys/socket.h>
9.  #include <string>
10.
11. #define SERVER_IP "127.0.0.1"
12. #define PORT 12345
13. #define BUFFER_SIZE 1024
14.
15. #define NO_CODE 0
16. #define CODE_FOR_END 0
17. #define CODE_FOR_LOGIN 1
18. #define CODE_FOR_REGISTER 2
19. #define CODE_FOR_USER_LIST 3
20. #define CODE_FOR_MSG_SEND 4
21. #define CODE_FOR_MSGS 5
22. #define CODE_FOR_MSG 6
23.
24. typedef struct {
25.     char username[32];
26.     int id;
27. } User;
28. typedef struct {
29.     char* text;
30.     int to_user_id;
31. }Msg;
32.
33. void put_code(char* buffer,int n);
34. int get_code(char* buffer);
```

```
35.    void     append_username_password(char*     buffer,char     username[32],char
password[32]);
36.    int get_num_after_code(char* buffer);
37.    int get_id_after_username(char* buffer);
38.    void get_username(char* buffer,void* username);
39.    void put_id_msg(char* buffer,int id,char* msg);
40.    void append_id_after_code(char* buffer,int id);
41.    void get_msg(char* buffer,char* msg);
42.    int get_msg_len(char* buffer);
43.    int get_id_after_msg(char* buffer,int msg_len);
44.
45.    class communication
46.    {
47.        private:
48.            int sock = 0;
49.            struct sockaddr_in serv_addr;
50.
51.            bool posting_msg;
52.            bool getting_msg;
53.
54.        public:
55.            User* user_list;
56.            int number_of_user_in_list;
57.
58.            Msg* msgs_list;
59.            int number_of_msg;
60.
61.            Msg last_msg;
62.
63.            communication()
64.            {
65.                posting_msg = false;
66.                getting_msg = false;
67.            };
68.
69.            bool start_communication();
70.            bool try_registration(char username[32],char password[32]);
71.            bool try_login(char username[32],char password[32]);
72.            bool end_communication();
73.            bool try_get_users();
74.            void print_user_list();
75.            int get_id_from_username(char username[32]);
76.            bool try_to_send_msg(int id,char msg[500]);
77.            bool try_to_get_msgs(int id);
78.            bool try_to_get_msg(int id);
79.    };
80.    #endif
```

4.3. *LOGIN.H FILE*

C++ код наведен у наставку овог подпоглавља дефинише класу *LogInFrame*, која представља прозор за пријаву у апликацији за размену порука. Ова класа наслеђује класу *wxFrame*, пружајући основну структуру прозора. Класа *LogInFrame* садржи : дугме за пријаву, дугме за регистрацију, текстуална поља корисничког имена и лозинке, ознаке и референце на комуникациони објекат, класе *RegisterFrame* и *MainFrame*. Конструктор *LogInFrame* узима комуникациони објекат као параметар, успостављајући везу између прозора за пријаву и комуникационе компоненте одговорне за интеракцију са сервером. Методе *OnClose*, *LoginFunction* и *RegisterFunction* обрађују догађаје који се односе на затварање прозора, пријављивање и регистрацију за нови налог. Ове методе ступају у интеракцију са комуникационим објектом и шаљу одговарајуће поруке серверу и обрађују одговоре. Класа *LogInFrame* пружа кориснички интерфејс за уношење акредитива за пријаву и навигацију до прозора за регистрацију или главног прозора апликације, у зависности од радњи корисника.

```
1.    #ifndef LOGIN_H
2.    #define LOGIN_H
3.
4.    #include <wx/wx.h>
5.    #include "../../communication_component/communication_object.h"
6.    #include "../register/register.h"
7.    #include "../main/mainFrame.h"
8.
9.    class LogInFrame : public wxFrame
10.   {
11.       public:
12.           LogInFrame(communication *server);
13.       private:
14.           void OnClose(wxCloseEvent& event);
15.           void LoginFunction(wxCommandEvent& event);
16.           void RegisterFunction(wxCommandEvent& event);
17.
18.           wxButton* loginButton;
19.           wxButton* registerButton;
20.           wxTextCtrl* usernameTextInput;
21.           wxTextCtrl* passwordTextInput;
22.           wxStaticText* usernameLabel;
23.           wxStaticText* passwordLabel;
24.           RegisterFrame *register_frame;
25.           MainFrame *main_frame;
26.
27.           communication *l_server;
28.   };
29.   #endif
```

4.4. *MAINFRAME.H FILE*

C++ код наведен у наставку овог подпоглавља дефинише класу *MainFrame*, која представља главни прозор апликације за размену порука. Ова класа наслеђује класу *wxFrame*, пружајући основну структуру прозора. *MainFrame* конструктор узима комуникациони објекат као параметар, успостављајући везу између главног прозора и комуникационе компоненте одговорне за интеракцију са сервером. Класа *MainFrame* садржи : дугме за одјаву, оквир са листом корисника, референцу на класу *ChatFrame* и референцу на објекат комуникације. Методе *OnClose*, *LogoutFunction* и *OpenChat* рукују догађајима који се односе на затварање прозора, одјављивање и отварање прозора за ћаскање. Метода *OnEnable* се користи за руковање догађајима у вези са активацијом или деактивацијом главног прозора. Класа *MainFrame* обезбеђује кориснички интерфејс за главну апликацију, омогућавајући корисницима да се одјаве, прегледају листу корисника и отворе прозоре за ћаскање са одређеним корисницима.

```
1.    #ifndef MAIN_H
2.    #define MAIN_H
3.
4.    #include <wx/wx.h>
5.    #include "../../communication_component/communication_object.h"
6.    #include "../chat/chat.h"
7.
8.    class MainFrame : public wxFrame
9.    {
10.       public:
11.           MainFrame(communication *server);
12.
13.           wxFrame* perent;
14.       private:
15.           void OnClose(wxCloseEvent& event);
16.           void LogoutFunction(wxCommandEvent& event);
17.           void OpenChat(wxCommandEvent& event);
18.           wxButton* logoutButton;
19.           wxListBox* listBox;
20.
21.           bool start;
22.
23.           ChatFrame* chat_frame;
24.
25.           void OnEnable(wxActivateEvent& event);
26.
27.           communication *l_server;
28.   };
29.   #endif
```

4.5. *REGISTER.H FILE*

C++ код наведен у наставку овог подпоглавља дефинише класу *RegisterFrame*, која представља прозор за регистрацију у апликацији за размену порука. Ова класа наслеђује класу *wxFrame*, пружајући основну структуру прозора. *RegisterFrame* конструктор узима комуникациони објекат као параметар, успостављајући везу између прозора за регистрацију и комуникационе компоненте одговорне за интеракцију са сервером. Класа *RegisterFrame* садржи променљиве чланова за дугме за регистрацију, корисничко име, лозинку и текстуална поља за потврду лозинке, ознаке и референцу на комуникациони објекат. Методе *OnClose* и *RegisterFunction* обрађују догађаје који се односе на затварање прозора и регистрацију за нови налог, респективно. Класа *RegisterFrame* пружа кориснички интерфејс за креирање новог налога, омогућавајући корисницима да унесу своје жељено корисничко име, лозинку и да потврде лозинку. Када корисник кликне на дугме за регистрацију, класа шаље информације о регистрацији серверу користећи комуникациони објекат и обрађује одговор са сервера.

```
1.  #ifndef REGISTER_H
2.  #define REGISTER_H
3.
4.  #include "../../communication_component/communication_object.h"
5.  #include <wx/wx.h>
6.
7.  class RegisterFrame : public wxFrame
8.  {
9.      public:
10.         RegisterFrame(communication *server);
11.
12.         wxFrame* perent;
13.
14.     private:
15.         void OnClose(wxCloseEvent& event);
16.         void RegisterFunction(wxCommandEvent& event);
17.
18.         wxButton* registerButton;
19.
20.         wxTextCtrl* usernameTextInput;
21.         wxTextCtrl* passwordTextInput;
22.         wxTextCtrl* confirmPasswordTextInput;
23.
24.         wxStaticText* usernameLabel;
25.         wxStaticText* passwordLabel;
26.         wxStaticText* confirmPasswordLabel;
27.
28.         communication *l_server;
29. };
30. #endif
```

4.6. *CHAT.H FILE*

C++ код наведен у наставку овог подпоглавља дефинише класу *ChatFrame*, која представља прозор за размену порука у апликацији. Ова класа наслеђује класу *wxFrame*, пружајући основну структуру прозора. *ChatFrame* конструктор узима комуникациони објекат, *id* корисника и корисничко име као параметре, успостављајући везу између прозора за ћаскање и комуникационе компоненте и пружајући информације о тренутном кориснику. Класа *ChatFrame* садржи : дугме за слање, поље за унос поруке, приказ листе порука, тајмер и референце на комуникациони објекат, *id* корисника и корисничко име. Методе *OnClose*, *SendFunction*, *AddMessage* и *OpenChat* управљају догађајима који се односе на затварање прозора, слање порука, додавање порука у историју порука и отварање нових прозора за ћаскање. Метода *OnTimer* се користи за руковање догађајима тајмера, који се могу користити за периодичну проверу нових порука са сервера. Класа *ChatFrame* обезбеђује кориснички интерфејс за интеракције размена порука, омогућавајући корисницима да шаљу и примају поруке, прегледају историју ћаскања и комуницирају са другим корисницима у оквиру апликације за размену порука.

```
1.    #ifndef CHAT_H
2.    #define CHAT_H
3.
4.    #include "../../communication_component/communication_object.h"
5.
6.    #include <wx/wx.h>
7.    #include <wx/stc/stc.h>
8.    #include <wx/listctrl.h>
9.
10.   class ChatFrame : public wxFrame
11.   {
12.       public:
13.           ChatFrame(communication *server,int id,char username[32]);
14.
15.           wxFrame* perent;
16.
17.       private:
18.           void OnClose(wxCloseEvent& event);
19.           void SendFunction(wxCommandEvent& event);
20.
21.               void AddMessage(wxStyledTextCtrl* chatDisplay, const wxString&
sender, const wxString& message, bool rightAlign);
22.
23.           void OpenChat(wxCommandEvent& event);
24.
25.           wxButton* sendButton;
26.           wxTextCtrl* msgTextInput;
27.           //wxListBox* chatList;
28.           wxListView* chatView;
29.
30.           bool start;
```

```
31.
32.             wxTimer* timer;
33.
34.             void OnTimer(wxTimerEvent& event);
35.
36.             wxDECLARE_EVENT_TABLE();
37.
38.             communication *l_server;
39.
40.             int user_id;
41.             char username[32];
42.     };
43.
44.     #endif
```

5. ИМПЛЕМЕНТАЦИЈА НА СТРАНИ СЕРВЕРА

Имплементација апликације за размену порука на страни сервера служи као централно чвориште за комуникацију, управљање клијентским везама, обраду порука и интеракцију са базом података. Архитектура сервера се састоји од три примарне компоненте: *main* функције, функције *handle_client* и *handle_client_components*. Главна функција је одговорна за иницијализацију сервера, везивање за одређени *port* и слушање долазних веза клијената. Он креира *socket*, везује га за *port* и улази у петљу да прихвати долазне везе. Када се нови клијент повеже, главна функција покреће нови *thread* за управљање комуникацијом клијента. Функција управљања клијентом је одговорна за управљање комуникацијом са одређеним клијентом. Он прима поруке од клијента, обрађује их, шаље одговоре и ажурира информације о клијенту у бази података. Компонента за руковање клијентом обухвата логику за руковање клијентским функцијама, укључујући обраду порука, аутентификацију корисника и интеракцију са базом података. Обезбеђује методе за слање и примање порука, проверу корисничких акредитива, чување и преузимање корисничких информација и управљање везом клијента. Имплементација на страни сервера осигурава да апликација за размену порука функционише ефикасно и поуздано.

5.1. *MAIN* ФУНКЦИЈА

C++ код наведен у наставку овог подпоглавља имплементира главну функцију за апликацију на страни сервера. Он иницијализује серверски *socket*, везује је за наведени *port* и ослушкује долазне клијентске везе. Код улази у бесконачну петљу, континуално прихвата нове клијентске везе и ствара одвојени *thread* за руковање сваким клијентом. Овај приступ омогућава серверу да истовремено рукује са више клијената, побољшавајући перформансе и одзив.

```
1.     int main() {
2.         //delete_db();
3.         //create_db();
4.
5.         int server_fd, new_socket;
6.         struct sockaddr_in address;
7.         int addrlen = sizeof(address);
8.         pthread_t thread_id;
9.
10.        if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
11.            perror("socket failed");
12.            exit(EXIT_FAILURE);
13.        }
14.
15.        address.sin_family = AF_INET;
16.        address.sin_addr.s_addr = INADDR_ANY;
17.        address.sin_port = htons(PORT);
18.
```

```
19.         if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0)
{
20.             perror("bind failed");
21.             close(server_fd);
22.             exit(EXIT_FAILURE);
23.         }
24.
25.         if (listen(server_fd, 3) < 0) {
26.             perror("listen");
27.             close(server_fd);
28.             exit(EXIT_FAILURE);
29.         }
30.
31.         std::cout << "Server listening on port " << PORT << std::endl;
32.
33.         while (true) {
34.             if ((new_socket = accept(server_fd, (struct sockaddr *)&address,
(socklen_t*)&addrlen)) < 0) {
35.                 perror("accept");
36.                 close(server_fd);
37.                 exit(EXIT_FAILURE);
38.             }
39.
40.             if    (pthread_create(    &thread_id,    nullptr,    handle_client,
(void*)&new_socket) != 0) {
41.                 perror("pthread_create failed");
42.                 close(new_socket);
43.             }
44.
45.             pthread_detach(thread_id);
46.         }
47.
48.         close(server_fd);
49.         return 0;
50.     }
```

5.2. *HANDLE_CLIENT* ФУНКЦИЈА

Функција *handle_client* управља комуникацијом са појединачним клијентом. Он успоставља почетну везу са клијентом и улази у петљу која се наставља све док клијент не покрене подпротокол за прекид комуникације са сервером. Унутар петље, функција чита податке са клијентског прикључка и одређује тип комуникације на основу кода примљеног у оквиру података. Различите вредности кода покрећу одређене подпротоколе:

- **0 : Крај комуникације** - покреће подпротокол за крај везе.
- **1 : Пријава** - преузима корисничко име и лозинку, а затим извршава подпротокол за пријаву.
- **2 : Регистрација** - преузима корисничко име и лозинку, а затим извршава подпротокол регистрације.

- **3 : Захтев за листу корисника** - покреће подпротокол за слање листе корисника.
- **4 : Пошаљи поруку** - преузима *id* примаоца и садржај поруке, а затим извршава подпротокол за слање поруке.
- **5 : Преузми поруке** - преузима *id* корисника, а затим извршава подпротокол за преузимање листе порука са корисником.
- **6 : Преузми последњу поруку** - преузима *id* корисника, а затим извршава подпротокол за преузимање последње поруке са корисником.

Функција користи помоћне функције као што су *get_code* и *get_username* за издвајање релевантних информација из примљених порука. Након обраде сваког захтева, функција наставља да чека даљу комуникацију од клијента док се веза не прекине.

```
1.    void* handle_client(void* client_socket) {
2.
3.        char buffer[BUFFER_SIZE] = {0};
4.
5.        client_objects l_user;
6.        l_user.start_connection(client_socket);
7.
8.        while(l_user.work)
9.        {
10.           read(l_user.sock, buffer, BUFFER_SIZE);
11.           int code = get_code(buffer);
12.
13.           if(code == 0)
14.           {
15.               l_user.protocol_for_end_connection();
16.           }
17.
18.           if(code == 1)
19.           {
20.               char username[32] = {0};
21.               char password[32] = {0};
22.
23.               get_username(buffer,&username);
24.               get_password(buffer,&password);
25.
26.               l_user.protocol_for_log_in(&username,&password);
27.           }
28.
29.           if(code == 2)
30.           {
31.               char username[32] = {0};
32.               char password[32] = {0};
33.
34.               get_username(buffer,&username);
```

```
35.                 get_password(buffer,&password);
36.
37.                 l_user.protocol_for_registration(&username,&password);
38.             }
39.
40.             if(code == 3)
41.             {
42.                 l_user.protocol_for_user_list_request();
43.             }
44.
45.             if(code == 4)
46.             {
47.                 int id;
48.                 char msg[500] = {0};
49.
50.                 id = get_user_id_after_code(buffer);
51.                 get_msg(buffer,&msg);
52.
53.                 l_user.protocol_for_send_msg(id,&msg);
54.             }
55.
56.             if(code == 5)
57.             {
58.                 int user_id;
59.                 user_id = get_user_id_after_code(buffer);
60.
61.                 l_user.protocol_for_get_msgs(user_id);
62.             }
63.
64.             if(code==6)
65.             {
66.                 int user_id;
67.                 user_id = get_user_id_after_code(buffer);
68.
69.                 l_user.protocol_for_get_msg(user_id);
70.             }
71.         }
72.
73.         close(l_user.sock);
74.         return nullptr;
75.     }
```

5.3. *HANDLE_CLIENT_COMPONENTS.H* ФУНКЦИЈА

C++ код наведен у наставку овог подпоглавља дефинише класу *client_objects*, која обухвата логику за руковање комуникацијом са одређеним клијентом. Класа укључује промењљиве за чување статуса аутентификације клијента, *id*-а, статуса везе, показивача на *socket* и корисничког имена. Такође пружа методе за руковање аутентификацијом, слање и примање порука и управљање везом. Метода *try_authenticate* верификује клијентове акредитиве помоћу функције; *try_register* која покушава да региструје новог корисника ако је потребно. Метод *put_msg* шаље поруку другом кориснику, док *get_last_msg* метод преузима последњу поруку коју је послао одређени корисник. Метод *start_connection* иницијализује везу са клијентом, а методе *protocol_for_** имплементирају специфичне комуникационе протоколе за различите задатке, као што су пријављивање, регистрација, слање порука и преузимање корисничких листа.

```
1.    #ifndef HANDLE_CLIENT_H
2.    #define HANDLE_CLIENT_H
3.
4.    #include <unistd.h>
5.    #include <arpa/inet.h>
6.    #include <sys/socket.h>
7.
8.    const int PORT = 12345;
9.    const int BUFFER_SIZE = 1024;
10.
11.   int get_code(char* buffer);
12.   void put_code(char* buffer,int n);
13.   void get_msg(char* buffer,void* msg);
14.   void get_username(char* buffer,void* username);
15.   int get_password_start(char* buffer);
16.   void get_password(char* buffer,void* password);
17.   int get_user_id_after_code(char* buffer);
18.   bool try_register(void* username,void* password);
19.   int get_user_num();
20.   int get_msg_num(int id1,int id2);
21.   void get_username_list(void* username,int id);
22.   void get_msg_list(void* msg_list,int id1,int id2);
23.
24.   class client_objects
25.   {
26.       public:
27.           bool is_authenticated;
28.           int id;
29.
30.           bool work;
31.           int sock;
32.
33.           char username[32];
```

```
34.             client_objects():is_authenticated(false),id(0),work(true)
35.             {};
36.
37.             void set_sock(void* client_socket);
38.             bool try_authenticate(void* username,void* password);
39.             bool put_msg(int l_id,void* msg);
40.             void get_last_msg(void *msg,int user_id);
41.             void start_connection(void* client_socket);
42.             void protocol_for_end_connection();
43.             void protocol_for_log_in(void* username,void* password);
44.             void protocol_for_registration(void* username,void* password);
45.             void protocol_for_user_list_request();
46.             void protocol_for_send_msg(int l_id, void* msg);
47.             void protocol_for_get_msgs(int user_id);
48.             void protocol_for_get_msg(int user_id);
49.     };
50.
51.     #endif
```

6. ДИЗАЈН И УПРАВЉАЊЕ БАЗОМ ПОДАТАКА

База података која се користи у апликацији за размену порука састоји се од две табеле: табеле *users* и табеле *messages*. Табела *users* приказана на слици 6.1 чува информације о регистрованим корисницима, укључујући њихов јединствени *id*, корисничко име и лозинку. Табела *messages* приказана на слици 6.2 чува информације о послатим и примљеним порукама, укључујући *id* поруке, временску ознаку, *id* пошиљаоца, *id* примаоца и текст поруке.

	id ▲	username	password
	F...	Filter	Filter
1	1	Korisnik	Sifra
2	2	Nenad	nenad
3	3	Nikola	nikola
4	4	Pera	pera

Слика 6.1 – Слика табеле са корисницима из базе података

	id	date_time	to_user	from_user	msg
	F...	Filter	Filter	Filter	Filter
1	1	1725268948	1	2	Cao
2	2	1725269227	1	2	Cao
3	3	1725269238	1	2	Cao2
4	4	1725269278	1	2	Cao2
5	5	1725269286	1	2	Cao3
6	6	1725540854	2	3	Zdravo.
7	7	1725540884	4	3	Zdravo
8	8	1725540902	3	4	Cao
9	9	1725540920	4	3	Kako si?

Слика 6.2 – Слика табеле са порукама из базе података

7. БЕЗБЕДНОСНА РАЗМАТРАЊА

Апликација за размену порука, иако је функционална, нема снажне безбедносне мере које су неопходне за заштиту корисничких података и спречавање неовлашћеног приступа. Одсуство инфраструктуре за шифровање и складиштење лозинки у облику обичног текста у бази података представљају значајне безбедносне ризике.

Шифровање : Недостатак шифровања значи да су поруке које се преносе између клијента и сервера осетљиве на пресретање и прислушкивање. Осетљиве информације, као што су личне поруке, могу бити угрожене ако их пресретну злонамерни актери. Примена енкрипције би значајно побољшала безбедност апликације тако што би отежала неовлашћеним странама да дешифрују пренете податке. Снажан алгоритам за шифровање, као што је *AES-256*, треба да се користи за шифровање порука пре преноса и дешифровање по пријему.

Хеширање лозинки : Чување лозинки у обичном тексту у бази података представља велики безбедносни недостатак. Ако је база података угрожена, нападачи би могли да добију приступ лозинкама и да их користе за пријаву на корисничке налоге. Хеширање лозинки пре њиховог складиштења у базу података је стандардна безбедносна пракса која нападачима знатно отежава откривање оригиналних лозинки. Снажан алгоритам хеширања, као што је *bcrypt* или *Argon2*, треба да се користи за хеширање лозинки, што их чини рачунарски скупим за разбијање.

Безбедни протокол : Иако је *TCP* поуздан протокол за комуникацију, он сам по себи не обезбеђује шифровање или аутентификацију. То значи да је веза између клијента и сервера подложна нападима човека у средини, где нападач може пресрести и изменити поруке. Коришћење безбедног протокола са могућностима шифровања и аутентификације, као што је *HTTPS*, помогло би да се ублажи овај ризик. *HTTPS* користи *TLS* (*Transport Layer Security*) за успостављање безбедне везе између клијента и сервера, шифровање података у преносу и проверу аутентичности сервера.

***C++* језик :** Иако је *C++* моћан језик за развој апликација, захтева пажљиву праксу кодирања да би се избегле безбедносне рањивости. Неправилно управљање меморијом, валидација уноса и руковање грешкама могу довести до безбедносних проблема. Придржавање безбедних пракси кодирања и коришћење алата за анализу кода може помоћи у идентификацији и адресирању потенцијалних рањивости. Уобичајене рањивости у *C++* апликацијама укључују *buffer overflow*, *memory leak* и *injection attack*. Пратећи смернице за безбедно кодирање и користећи алате за статичку анализу кода, програмери могу да ублаже ове ризике.

Додатна разматрања:

- **Валидација уноса :** Апликација треба да ригорозно проверава кориснички унос како би спречила нападе убризгавањем, као што је *SQL injection*. Валидација уноса треба да обезбеди да су подаци очекиваног типа и формата и да не садрже злонамерни код.

- **Управљање сесијом :** Треба применити одговарајуће управљање сесијом како би се спречио неовлашћени приступ корисничким налозима. Токени сесије треба да буду безбедно генерисани и пренети, и требало би да буду поништени када се корисник одјави или истекне његова сесија.

- **Контроле приступа :** Контроле приступа би требало да буду постављене како би се ограничио приступ осетљивим подацима и функционалности на основу корисничких улога и дозвола. Ово помаже у спречавању неовлашћеног приступа корисничким налозима и осетљивим информацијама.

- **Редовна ажурирања :** Апликацију треба редовно ажурирати како би се решиле безбедносне рањивости и укључила најновија најбоља безбедносна пракса. Ово укључује примену безбедносних закрпа, ажурирање библиотека и зависности и спровођење безбедносних провера.

Како би апликација за размену порука постала значајно сигурнија, неопходно је спровести следеће сигурносне мере:

- **Шифровање :** Применом јаког алгоритма за шифровање, као што је *AES-256*, за шифровање порука које се преносе између клијента и сервера.

- **Хеширање лозинке :** Чувањем лозинки у бази података као хеширане вредности користећи јак алгоритам хеширања као што је *bcrypt* или *Argon2*.

- **Безбедни протокол :** Коришћењем *HTTPS* би се успоставила безбедна веза између клијента и сервера, обезбеђујући шифровање и аутентификацију.

- **Валидација уноса :** Применом валидације уноса би се спречили напади убризгавањем и обезбедили интегритети података.

- **Управљање сесијом :** Спровођењем правилног управљања сесијом би се спречили неовлашћени приступи.

- **Редовна ажурирања :** Одржавајте апликацију ажурном са безбедносним закрпама и ажурирањима.

Уклањањем ових безбедносних недостатака, апликација за размену порука се може учинити безбеднијом и заштитити корисничке податке од неовлашћеног приступа. Имплементација шифровања, хеширање лозинки, коришћење безбедног протокола су суштински кораци да се обезбеди безбедност апликације.

8. ЗАКЉУЧАК

Овај рад је успешно развио апликацију за размену порука засновану на *C++* која омогућава комуникацију у реалном времену између више клијената. Апликација користи *wxWidgets GUI* библиотеку за кориснички интерфејс на страни клијента и *SQLite* базу података на страни сервера. *TCP* протокол се користи за комуникацију између клијената и сервера.

Архитектура апликације састоји се од три главне компоненте: клијента, сервера и базе података. Имплементација на страни клијента се фокусира на пружање корисничког интерфејса и руковање корисничким интеракцијама. Имплементација на страни сервера управља везама клијената, аутентификацијом корисника, обрадама порука и интеракцијом са базом података. База података се користи за чување корисничких информација и историја порука.

Апликација за размену порука укључује прилагођени протокол за комуникацију између клијента и сервера, обезбеђујући ефикасну и поуздану размену података. Протокол укључује подпротоколе за различите задатке, као што су пријава, регистрација, управљање листом корисника, слање порука и преузимање порука.

Иако апликација показује функционалне могућности, важно је напоменути да јој недостају робусне безбедносне мере. Одсуство шифровања и складиштење лозинки у облику обичног текста у бази података представљају значајне безбедносне ризике. Будућа побољшања би требало да дају приоритет примени шифровања, хеширања лозинке и безбедног протокола за заштиту корисничких података и спречавање неовлашћеног приступа.

Овај рад описује апликацију за размену порука која успоставља оквир за комуникацију у реалном времену. Ипак, потребна су даља побољшања како би се ојачала његова безбедност и подигла укупна перформанса и само корисничко искуство.

9. ИНДЕКС ПОЈМОВА

10. ЛИТЕРАТУРА

[1] Julian Smart, Kevin Hock, Stefan Csomor „Cross-Platform GUI Programming with WxWidgets“ https://students.mimuw.edu.pl/~kedar/gui/wxwidgets_book.pdf , преузето : септембар 2024.

[2] Scott Meyers „Effective Modern C++“ http://www.bugs.frozent.pl/Effective%20Modern%20C++%20(%20PDFDrive.com%20).pdf , преузето : септембар 2024.

[3] Michael J. Donahoo , Kenneth L. Calvert „TCP/IP Sockets in C“ http://debracollege.dspaces.org/bitstream/123456789/406/1/TCPIP%20Sockets%20in%20C%20by%20Michael%20J.%20Donahoo.pdf , преузето : септембар 2024.

[4] Jay A. Kreibich „Using SQLite“ https://it.dru.ac.th/o-bookcs/pdfs/17.pdf , преузето : септембар 2024.

[5] SQLite , „An Introduction To The SQLite C/C++ Interface“ , https://www.sqlite.org/cintro.html , преузето : septembar 2024.

11. ИЗЈАВА О АКАДЕМСКОЈ ЧЕСТИТОСТИ

ИЗЈАВА О АКАДЕМСКОЈ ЧЕСТИТОСТИ

Студент (име, име једног родитеља и презиме): **Ненад Срђан Ђонић**

Број индекса: **РТ-42/19**

Под пуном моралном, материјалном, дисциплинском и кривичном одговорношћу изјављујем да је завршни рад, под насловом:

***C++* Апликација за размену порука**

1. резултат сопственог истраживачког рада;
2. да овај рад, ни у целини, нити у деловима, нисам пријављивао/ла на другим високошколским установама;
3. да нисам повредио/ла ауторска права, нити злоупотребио/ла интелектуалну својину других лица;
4. да сам рад и мишљења других аутора које сам користио/ла у овом раду назначио/ла или цитирао/ла у складу са Упутством;
5. да су сви радови и мишљења других аутора наведени у списку литературе/референци који је саставни део овог рада, пописани у складу са Упутством;
6. да сам свестан/свесна да је плагијат коришћење туђих радова у било ком облику (као цитата, парафраза, слика, табела, дијаграма, дизајна, планова, фотографија, филма, музике, формула, веб-сајтова, компјутерских програма и сл.) без навођења аутора или представљање туђих ауторских дела као мојих, кажњиво по закону (Закон о ауторском и сродним правима), као и других закона и одговарајућих аката Академије техничко-уметничких струковних студија Београд;
7. да је електронска верзија овог рада идентична штампаном примерку овог рада и да пристајем на његово објављивање под условима прописаним актима Академије техничко-уметничких струковних студија Београд;
8. да сам свестан/свесна последица уколико се докаже да је овај рад плагијат.

У Београду, __. __. 202_. године

Својеручни потпис студента

_____