Analiza i Verifikacija Softvera

Online Shopping System

Dimitrije Marković 1022/24
Matematički fakultet, Univerzitet u Beogradu

12. septembar 2025.

Sadržaj

---

1 Uvod

Za potrebe kursa *Verifikacija Softvera*, analiziran je projekat **Online Shopping System**. Projekat je razvijen u programskom jeziku C++ kao konzolna aplikacija, sa klijent–server arhitekturom koja koristi `Winsock2` biblioteku za mrežnu komunikaciju.

Aplikacija implementira osnovne funkcionalnosti sistema za online kupovinu, uključujući registraciju i prijavu korisnika, pregled i kupovinu proizvoda, generisanje računa, kao i mogućnost izmene porudžbine. Dodatno, sistem pruža različite privilegije za korisnike, zaposlene i administratore (upravljanje nalozima, popusti, pregled inventara, rešavanje žalbi i automatsko procesuiranje porudžbina).

Cilj ove analize je da se proceni pouzdanost, efikasnost i kvalitet implementacije kroz upotrebu savremenih alata za statičku i dinamičku analizu softvera. U okviru rada korišćeni su sledeći alati:

- **Clang-Tidy** i **Cppcheck** — alati za statičku analizu koda i detekciju potencijalnih problema,
- **Doxygen** — za automatsko generisanje dokumentacije iz izvornog koda,
- **Dr. Memory** — za detekciju problema u radu sa memorijom tokom izvršavanja,
- **Very Sleepy Profiler** — za profilisanje performansi i identifikaciju najintenzivnijih delova koda.

Analiza i dobijeni rezultati omogućavaju uvid u potencijalne slabosti projekta i daju preporuke za dalje unapređenje koda, sa ciljem povećanja njegove stabilnosti, efikasnosti i dugoročne održivosti.

2 Statička analiza

2.1 Clang-Tidy

Alat **Clang-Tidy** je korišćen za proveru koda u direktorijumima `Server` i `Client`. Pokretanje je vršeno pomoću batch skripte koja rekurzivno prolazi kroz sve `.cpp` fajlove i primenjuje sledeće grupe provera: `clang-diagnostic-*`, `clang-analyzer-*`, i `modernize-*`.

Sažetak rezultata

- **Greške (errors):** 2 tipa, sa više pojavljivanja.
- **Upozorenja (warnings):** oko 8022 ukupno.

Greške (errors) i rešenja

(E1) Nedefinisan `std::basic_ostringstream<char>` (#include `<sstream>`) **Opis:** Greška nastaje kada se koristi `std::ostringstream` bez uključivanja hedera `<sstream>`.

Fajlovi (reprezentativno):

- `Server/Headers/Admin.cpp`,
- `Server/Headers/Employee.cpp`,
- `Server/Headers/Person.cpp`,
- `Server/Headers/Server.cpp`,

- `Server/Headers/Customer.cpp`.

Rešenje: Dodati sledeći include u svaku prevodnu jedinicu gde se koristi `ostringstream`:

```
#include <sstream>
```

(E2) Konflikt Winsock zaglavlja (redefinicije tipova / expected identifier) **Opis:** Pojavljuju se višestruke greške (*"expected identifier"*, *"redefinition of 'sockaddr'/'fd_set' ..."*) koje potiču iz Windows SDK hedera (`ws2def.h`, `winsock2.h`). Uzrok je uključivanje `<windows.h>` pre `<winsock2.h>`, što povlači zastareli `winsock.h` i izaziva konflikt.

Fajlovi (posredno): greške su prijavljene u SDK hederima, ali izvorni kod se nalazi u mrežnim modulima, npr. `Server/Headers/Thread.cpp`, `Server/Headers/Server.cpp`.

Rešenje: Osigurati ispravan redosled uključivanja u svim fajlovima koji koriste mrežu:

```
#define WIN32_LEAN_AND_MEAN // spreava povlaenje suvinih API-ja
#include <winsock2.h> // uvek pre <windows.h>
#include <ws2tcpip.h> // za InetPton/InetNtop
#include <windows.h> // tek posle winsock2.h
```

Upozorenja (warnings) i preporuke

(W1) Zastarele/nesigurne CRT i Winsock funkcije **Opis:** Korišćenje zastarelih ili nesigurnih funkcija.

Primeri:

- `getch()` → koristiti `_getch()` (`<conio.h>`),
- `strcpy()` → koristiti `strcpy_s()` ili preći na `std::string`,
- `ctime()`, `localtime()` → koristiti `ctime_s()`, `localtime_s()`,
- `inet_addr()`, `inet_ntoa()` → koristiti `InetPton()`, `InetNtop()`.

Fajlovi: `Client/Headers/Person.cpp`, `Client/Headers/Admin.cpp`, `Client/Headers/Customer.cpp`, `Server/Headers/Employee.cpp`, `Server/Headers/Complaint_Base.cpp`, `Server/Headers/Complaint_E.cpp`, `Server/Headers/Server.cpp`, `Server/serverProgram.cpp`, `Client/clientProgram.cpp`.

Napomena: ukupno oko 105 slučajeva, od čega `getch()` 69, `strcpy()` 28, ostalo pojedinačni slučajevi.

(W2) Modernizacija C++ koda (modernize checks)

- `modernize-use-nullptr` (~15 puta): zameniti `NULL` sa `nullptr`. **Fajlovi:** `Server/Headers/Server.h`, `Server/Headers/Thread.cpp`, `Server/Headers/Server.cpp`, `Client/Headers/Person.cpp`.
- `modernize-loop-convert` (~8 puta): zameniti klasične `for` petlje po kontejnerima range-based `for`.

  Fajlovi: `Server/Headers/Person.cpp`, `Server/Headers/Admin.cpp`.

(W3) Sigurnosna upozorenja (`clang-analyzer-security.insecureAPI.strcpy`) **Opis:** Korišćenje `strcpy` je prijavljeno kao nesigurno (CWE-119). **Fajlovi:** `Server/Headers/Complaint_Base.cpp`, `Server/Headers/Complaint_E.cpp`. **Rešenje:** koristiti `strcpy_s()` sa granicama bafera ili preći na `std::string`.

(W4) Neispravan poziv funkcije **Opis:** U `Server.cpp` je pozvana funkcija `WSAGetLastError` bez zagrada, što ispisuje pokazivač umesto vrednosti.

Rešenje:

```
std::cerr << "Error:␣" << WSAGetLastError() << std::endl;
```

(W5) Potencijalno neinicijalizovani objekti **Opis:** Detektovan je neinicijalizovani član u klasi `Client`.

Fajl: `Client/Headers/Client.h`.

Rešenje: inicijalizovati sve članove u konstruktoru, npr:

```
Client(const std::string& ipAddr = "127.0.0.1", int port = 7777)
  : m_ipAddress(ipAddr), m_port(port), client(INVALID_SOCKET) {}
```

Zaključak

Greške: najkritičniji problemi su nedostajući `<sstream>` i konflikt Winsock zaglavlja.

Upozorenja: odnose se na zastarele i nesigurne funkcije, potrebu za modernizacijom koda, sigurnosne rizike sa `strcpy`, neispravne pozive funkcija i neinicijalizovane članove. Implementacijom predloženih rešenja smanjiće se rizik od sigurnosnih propusta i poboljšaće se čitljivost i stabilnost koda.

2.2 Cppcheck

Alat **Cppcheck** je korišćen za statičku analizu koda sa ciljem da se otkriju potencijalni problemi koji nisu uvek detektovani kompajlerom. Analiza je obuhvatila i `Server` i `Client` deo projekta.

2.2.1 Glavne kategorije detektovanih problema

- **Nedostajući sistemski hederi (missingIncludeSystem)**
  Prijavljeno u velikom broju fajlova, npr. `Server/Headers/Person.h`, `Server/Headers/Server.h`, `Client/Headers/Person.h`, `Client/Headers/Client.h`.
  Ovi upozorenja ne predstavljaju realan problem jer Cppcheck ne zahteva standardne biblioteke, ali ukazuju na zavisnost projekta od platforme (Windows-specifični hederi poput `conio.h`, `windows.h`, `ws2tcpip.h`).

- **Neinicijalizovane promenljive (uninitMemberVar)**
  `Thread::data` u `Server/Headers/Thread.h`, kao i `Person::sex`, `Person::age`, i `Person::DOB` u `Client/Headers/Person.h`.
  Rešenje: dodati inicijalizaciju u konstruktorima:

  ```
  Person::Person() : sex('U'), age(0), DOB("") {}
  ```

- **Klase bez konstruktora (noConstructor)**
  Klase `Goods`, `Cash`, `Complaint_Base`, `Complaint_E` nemaju deklarisan konstruktor iako poseduju privatne članove.
  Rešenje: eksplicitno deklarisati konstruktore i inicijalizovati sve članove.

- **Pogrešan redosled u initializer listi (initializerList)**
  Detektovano u `Server::m_port` (`Server/Headers/Server.h`). Članovi klase se uvek inicijalizuju redosledom deklaracije.
  Rešenje: prilagoditi redosled initializer liste redosledu deklaracije članova.

- **Funkcije koje mogu biti `const` (functionConst)**
  Na primer: `Admin::view`, `Admin::accounts`, `Cash::get_final_cash`, `Complaint_Base::view`, `Complaint_Base::see`, `Person::transfer_to_file`.
  Rešenje: dodati ključnu reč `const` kada funkcija ne menja unutrašnje stanje objekta.

- **Funkcije koje mogu biti `static` (functionStatic)**
  Detektovano u `Person::initialize_goods`, `Person::initialize_cash`, `Server::updateActivity`, `Server::getActivity`, `Server::closeSocket`.
  Rešenje: ako funkcija ne koristi stanje objekta, može biti deklarisana kao `static` radi performansi.

- **Konstruktori koji nisu `explicit` (noExplicitConstructor)**
  Klase `Server`, `Employee`, `Admin`, `Customer`, `Client`.
  Rizik: implicitne konverzije.
  Rešenje: koristiti `explicit` za jednoargumentne konstruktore.

- **Opseg promenljivih (variableScope)**
  Promenljive poput `ans`, `previous`, `Balance` imaju širi opseg nego što je potrebno (npr. `Admin.cpp`, `Complaint_Base.cpp`, `Person.cpp`).
  Rešenje: deklarisati promenljive što bliže mestu korišćenja.

- **Problemi sa prenosom argumenata (constParameterReference)**
  Na primer: parametri funkcija `Person::update_cash`, `Thread::Rec`, `Thread::Client` mogu biti prosleđeni kao `const &`.
  Rešenje: koristiti `const T&` za velike objekte da bi se izbeglo nepotrebno kopiranje.

- **Nepotrebne ili sumnjive operacije (constStatement)**
  Detektovano u `Admin.cpp`, `Person.cpp` (izrazi tipa `+` ili `-` bez efekta).
  Rešenje: proveriti da li je to bug ili nepotreban kod.

- **Problemi sa prenosom argumenata funkcija (funcArgNamesDifferent)**
  Na primer: `search` u `Admin.cpp` koristi različita imena parametara u deklaraciji i definiciji (`str` vs `key`).
  Rešenje: ujednačiti imena parametara.

- **Portabilnost: korišćenje `fflush(stdin)`**
  Detektovano u `Admin.cpp` (više puta). Ovo ponašanje nije definisano na svim sistemima.
  Rešenje: koristiti druge metode za čišćenje bafera ulaza (`cin.ignore()` u C++).

- **Stilski problemi**
  - **C-style kastovi** (`(type)ptr`) u `Server.cpp`. Preporuka je korišćenje C++ kastova (`static_cast`, `reinterpret_cast`).
  - **Unused variables** (`index` u `Admin.cpp`). Ukloniti ako nisu potrebne.

2.2.2 Zaključak analize Cppcheck-a

Analiza je otkrila značajan broj stilskih i sigurnosnih problema. Najkritičniji su neinicijalizovani članovi klasa (`Person`, `Thread`), upotreba nesigurnih funkcija (`fflush(stdin)`), kao i implicitni konstruktori bez `explicit` modifikatora. Preporučuje se uvođenje konstruktora sa inicijalizacijom, dodavanje `explicit`, kao i korišćenje modernih C++ tehnika (`const`, `auto`, range-based petlje) kako bi se unapredio kvalitet i stabilnost koda.

3 Doxygen

3.1 Doxygen (generisana dokumentacija)

Za izradu tehničke dokumentacije korišćen je **Doxygen**. Dokumentacija je generisana iz izvornog koda (Server/ i Client/ sloj) u HTML formatu. Ovaj odeljak opisuje *kako je pokrenut* i *kako je organizovan* izlaz, tj. šta je dostupno za pregled.

Kako je pokrenut?

```
doxygen Doxyfile
```

Ključne postavke:

- `PROJECT_NAME = "Online Shopping System"`
- `RECURSIVE = YES`
- `GENERATE_HTML = YES`, `GENERATE_LATEX = NO`
- `EXTRACT_ALL = YES`, `EXTRACT_PRIVATE = YES`
- `GENERATE_TREEVIEW = YES`
- `HAVE_DOT = NO`

HTML početna strana je `index.html`.

Struktura generisane dokumentacije

- **Main Page** – ulazna stranica projekta (naziv, kratak opis).
- **Classes**
  - **Class List** – kompletna lista svih klasa/struktura u projektu (npr. `Admin`, `Cash`, `Client`, ...).
  - **Class Index** – azbučni indeks svih klasa.
  - **Class Hierarchy** – hijerarhijski prikaz odnosa klasa (bazne/izvedene), tekstualno organizovan.
  - **Class Members** – objedinjeni pregled članova svih klasa, podeljeno na:
    - **All** – svi članovi na jednom mestu,
    - **Functions** – samo metode,
    - **Variables** – samo polja/atributi,
    - **Related Symbols** – prijateljske funkcije, aliasi, tipovi i sl.
- **Files**
  - **File List** – lista svih `.h`/`.cpp` fajlova (npr. u okviru čvora `online-shopping-system`); za svaki fajl postoji posebna stranica.
  - **File Members** – objedinjeni pregled globalnih funkcija, promenljivih, `typedef`-ova, `enum`-a, `#define`-ova.
- **Search** – polje za pretragu simbola (klase, funkcije, fajlovi).

Sadržaj stranica (šta se vidi po klikovima)

- **Stranica klase** (npr. `Client`, `Server`): kratak opis (*brief*), detaljniji opis (*details*) ako postoji, tabele članova po vidljivosti (public/protected/private), sekcije za dokumentaciju konstruktora/destruktora, metoda i atributa, kao i linkovi ka fajlovima u kojima je klasa definisana.
- **Stranica člana klase** (metoda/polje): potpis, opis parametara/povratne vrednosti, napomene i veze ka pozivima/pomenima u kodu (ukoliko je uključeno izlistavanje izvora).
- **Stranica fajla** (npr. `Server/Headers/Server.cpp`): opis, lista uključenih hedera, spisak globalnih funkcija/promenljivih/`define`/`enum`/`typedef`; linkovi ka klasama koje fajl definiše ili koristi.
- **Agregatne liste** (*Class Members*/*File Members*): brza filtracija svih simbola po tipu (funkcije/varijable/ostalo) bez otvaranja pojedinačnih stranica.

3.2 Very Sleepy Profiler (analiza performansi)

Very Sleepy je sampling profiler za Windows. U ovoj analizi profilisane su odvojeno binarke `server.exe` i `client.exe`. Podaci su eksportovani u `.callgrind` i `.csv` formate i korišćeni za identifikaciju *hotspot* funkcija (Inclusive/Self vreme) i kritičnih putanja poziva.

Metodologija

- **Profilisanje:** sampling (*stack snapshots*); trajanje i interval uzorkovanja prema podrazumevanim postavkama Very Sleepy.
- **Artefakti:** `server.callgrind`, `server.csv`, `client.callgrind`, `client.csv`.
- **Napomena:** CRT start funkcije (`__tmainCRTStartup`, `mainCRTStartup`) su očekivano visoko u listi zbog ulazne tačke procesa i ne predstavljaju cilj optimizacije.

Rezultati (Server)

Tabela 1: Server — Top funkcije po % ukupnog vremena

Funkcija	Vreme [s]	Udeo %
`[00007FFD1F4D8C5C]`	657.783	23.24%
`__tmainCRTStartup`	330.008	11.66%
`mainCRTStartup`	330.008	11.66%
`Server::run`	330.008	11.66%
`Thread::Rec`	329.855	11.65%
`NSPStartup`	329.855	11.65%
`Tcpip4_WSHAddressToString`	329.370	11.63%
`worker`	96.620	3.41%

Tabela 2: Server — Projektne funkcije

Funkcija	Vreme [s]	Udeo %
`Server::run`	330.008	11.66%
`Thread::Rec`	329.855	11.65%
`worker`	96.620	3.41%
`Person::login`	96.620	3.41%
`Person::buy`	0.131	0.00%
`std::__basic_file<char>::open`	0.089	0.00%
`std::basic_filebuf<...>::open`	0.089	0.00%

Top funkcije (ukupno vreme, iz `server.callgrind`).

Zapažanja (Server).

- **Mrežni sloj dominira:** `Server::run`/`Thread::Rec` nose glavninu vremena ⇒ trošak je fokusiran na petlju koja prihvata/obradjuje i čita sa soketa.
- **Sistemske funkcije:** prisutni `NSPStartup`/`Tcpip4_WSHAddressToString` i jedna anonimna adresna tačka; to su spoljne (OS/network) rutine, nisu primarni kandidati za refaktorisanje.
- **I/O i fajlovi:** pojavljuju se `std::basic_file*::open` (zanemarljiv udeo).

Rezultati (Client)

Tabela 3: Client — Top funkcije po % ukupnog vremena

Funkcija	Vreme [s]	Udeo %
`__tmainCRTStartup`	325.177	21.14%
`mainCRTStartup`	325.177	21.14%
`Person::login`	295.432	19.20%
`__gnu_cxx::stdio_sync_filebuf<...>::underflow`	279.252	18.15%
`std::istream::sentry::sentry`	191.348	12.44%
`Customer::start`	62.839	4.08%
`Person::buy`	37.594	2.44%
`Person::profile`	4.260	0.28%

Tabela 4: Client — Projektne funkcije

Funkcija	Vreme [s]	Udeo %
`Person::login`	295.432	19.20%
`Customer::start`	62.839	4.08%
`Person::buy`	37.594	2.44%
`Person::profile`	4.260	0.28%
`heading`	0.806	0.05%
`Client::Rec`	0.140	0.01%

Top funkcije (ukupno vreme, iz `client.callgrind`).

Zapažanja (Client).

- **Ulaz sa konzole:** veliki udeo vremena u

  `stdio_sync_filebuf::underflow` i `std::istream::sentry::sentry`

  ukazuje da su `std::cin`/konzolni ulazi glavni *hotspot*.

- **Tokovi aplikacije:** `Person::login`/`Customer::start`/`Person::buy` čine glavne korisničke scenarije.

Napomena

Prikazani procenti i vremena su rezultat *sampling* profilisanja, te predstavljaju procene zasnovane na uzorkovanim stanjima steka u datom scenariju i okruženju (trajanje, interval uzorkovanja, build konfiguracija). Kao takvi, pouzdano ističu *hotspot* funkcije za konkretno opterećenje, ali nisu apsolutne metrike važeće za sve slučajeve.

Statičke optimizacije iz odeljka Clang-Tidy (npr. zamena `strcpy`, `_getch`, `inet_*` → `InetPton/Ntop`) i modernizacije (`nullptr`, range-based `for`) primarno unapređuju ispravnost, bezbednost i održivost koda; njihov efekat na performanse je posredan i slučajan. Nasuprot tome, **profilisanje** ukazuje gde se troši najveći deo *uzorkovanog* CPU vremena u realnom radu, pa stoga služi za ciljanje optimizacija nakon što je kod stabilizovan statičkim ispravkama.

3.3 Dr. Memory

Alat **Dr. Memory** je korišćen za dinamičku analizu izvršavanja servera i klijenta. Alat detektuje curenja memorije, neinicijalizovane ili nevažeće pristupe memoriji, *use-after-free*, dvostruko oslobađanje i oštećenja stoga/heap-a.

3.3.1 Dr. Memory — analiza servera

Rezultati. U fajlu sa rezultatima zabeleženo je preko **61 prijava** . Najčešće kategorije su:

- **UNINITIALIZED READ** (preko 50 prijava): čitanja neinicijalizovanih registara/memorije (npr. `eax`, `r9`, `eflags`) u `WSAStartup`/`accept`/`closesocket`, pozivane iz

  `Server::start`/`Server::Listen`/`main`.

- **UNADDRESSABLE ACCESS** (1 prijava): pristup izvan stoga (*beyond top of stack*) u toku `Server::start`.

- **Potential errors (blocklist)**: deo UNINIT čitanja u `KERNELBASE.dll`/`WS2_32.dll` alat klasifikuje kao verovatne lažno pozitivne (OS/CRT).

Diskusija grešaka. Dominantne prijave su neinicijalizovana čitanja u WinSock toku, tipično:

- `replace_memset` → `WSAStartup` → `Server::start` → `main` — UNINIT pri inicijalizaciji WinSock-a.

- `WS2_32.dll!closesocket` → `Server::Listen` → `Server::start` → `main` — UNINIT na registru `r9` pri zatvaranju konekcije.

- *Beyond top of stack* — upućuje na pogrešnu upotrebu lokalnih bafera/pokazivača ili prelazak opsega.

Preporuke za ispravku.

1. **Inicijalizacija i provere:** pre svakog WinSock poziva (`WSAStartup`/`accept`/`closesocket`) inicijalizovati strukture (`sockaddr_in`, `WSADATA`) i proveriti povratne kodove.
2. **RAII za sokete i memoriju:** uvesti omotače koji automatski zatvaraju `SOCKET` u destruktoru; posle `closesocket` postaviti deskriptor na `INVALID_SOCKET`.
3. **Stog i životni vek objekata:** ne vraćati/prosleđivati pokazivače na lokalne bafer-e; za veće strukture koristiti dinamičku alokaciju ili statički trajni objekat.
4. **Filtriranje šuma:** potencijalne greške iz OS/CRT (blocklist) tretirati kao lažno pozitivne ukoliko ne postoji uticaj na korisnički kod.

Zaključak. Nisu uočena tipična curenja ili dvostruka oslobađanja, ali su detektovana brojna UNINIT čitanja i jedno oštećenje stoga. Uz doslednu inicijalizaciju, RAII upravljanje soketima i reviziju upotrebe lokalnih bafera, stabilnost serverskog dela se značajno unapređuje.

3.3.2 Dr. Memory — analiza klijenta

Rezultati. Analiza više pokretanja (*clientProgram.exe*) pokazuje ponovljiv obrazac:

- **UNINITIALIZED READ** tokom WinSock inicijalizacije i DNS/servisnih upita:

  `WSALookupServiceNextW`/`WSAStartup` (registri `eax`, `eflags`; čitanja iz memorijskih slotova oko `0x7ef930`).
- **UNINITIALIZED READ** pri gašenju/rezoluciji: `closesocket` (reg. `r9`) i `getnameinfo`/`WSAUnhookBlockingHook` (reg. `rcx`).
- **UNADDRESSABLE ACCESS** u ranoj CRT/relocation fazi (*beyond top of stack*), pre korisničke logike.
- **Potential errors (blocklist)**: deo UNINIT prijava iz `KERNELBASE.dll`/`WS2_32.dll` klasifikovan kao OS/CRT šum.

Diskusija grešaka. Ključne tačke:

- `Client::start` → `main` — UNINIT u `WSAStartup`/`WSALookupServiceNextW` tipično nastaje kada strukture/parametri nisu nulto inicijalizovani ili se oslanjaju na nedefinisane vrednosti.
- `Client::Connect` — UNINIT na `r9` pri `closesocket` sugeriše dvostruko zatvaranje ili upotrebu nevalidnog `SOCKET`-a; UNINIT u `getnameinfo` ukazuje na ulazne parametre/dužine.
- *Beyond top of stack* u CRT startupu najverovatnije nije iz korisničkog koda, ali opravdava proveru build okruženja/flagova i upotrebe velikih lokalnih bafera.

Preporuke za ispravku.

1. **Deterministička inicijalizacija:** nulto inicijalizovati `WSADATA`, `sockaddr_in`/`addrinfo` i sve pomoćne strukture; validirati `len` vrednosti pre poziva API-ja.
2. **Bezbedno rukovanje soketima:** RAII omotač za `SOCKET`; pre `closesocket` proveriti `socket != INVALID_SOCKET`, a potom setovati `INVALID_SOCKET`.

3. **Ulazno/izlazni baferi:** koristiti `std::vector<char>` ili `std::array` i osigurati potpuni inicijalizacioni opseg pre `send`/`recv`/`getnameinfo`.

4. **Smanjenje šuma:** blocklistovane *potential errors* iz sistemskih DLL-ova ignorisati osim ako ne postoje indikacije na korisničke parametre.

Zaključak. Klijent deli obrazac sa serverom: dominiraju UNINIT čitanja u WinSock stazama, uz nekoliko relevantnih tačaka u korisničkom kodu (zatvaranje soketa, validacija parametara i bafera). Primena preporuka (inicijalizacija, RAII, proverene dužine/parametri, bezbedni baferi) umanjuje rizik od oštećenja memorije i povećava pouzdanost klijentskog dela.

4 Zaključak

Analiza projekta **Online Shopping System** kombinovanjem statičkih alata (*Clang-Tidy*, *Cppcheck*), generisane dokumentacije (*Doxygen*), dinamičke analize memorije (*Dr. Memory*) i profajlisanja performansi (*Very Sleepy*) obezbedila je sveobuhvatan uvid u kvalitet i pouzdanost implementacije.

Pozitivni aspekti. Arhitektura klijent–server je jasno razdvojena i funkcionalno koherentna. Doxygen dokumentacija (hijerarhija tipova, call grafovi) olakšava razumevanje uloga ključnih komponenti, dok rezultati profilisanja potvrđuju da se najveći deo vremena troši u očekivanim tačkama (mrežni I/O na serveru, konzolni I/O na klijentu), bez neočekivanih uskih grla.

Uočeni problemi. Statička analiza ukazuje na upotrebu zastarelih/nesigurnih API-ja i na potrebu modernizacije C++ stila (npr. `nullptr`, `explicit`, `const`-ispravnost, range-based `for`), kao i na pojedine neinicijalizovane članove i nedosledan redosled u *initializer* listama. Dinamička analiza detektuje *UNINITIALIZED READ* događaje i pristupe izvan važećeg opsega stoga u stazama vezanim za `Winsock` i lokalne bafer; deo tih nalaza potiče iz sistemskih biblioteka i može biti blok-listiran, ali deo ukazuje na realne rizike u tretmanu `SOCKET` deskriptora i validaciji bafera pri `send`/`recv`.

Preporuke.

- Ispravan redosled uključivanja zaglavlja (`winsock2.h` pre `windows.h`, `WIN32_LEAN_AND_MEAN`) i zamena zastarelih funkcija sigurnijim alternativama (`strcpy` → `strcpy_s`/`std::string`, `getch` → `_getch`, `inet_*` → `InetPton/Ntop`).
- RAII pristup za sokete i resurse (jednoznačno vlasništvo; posle `closesocket` postaviti `INVALID_SOCKET`).
- Inicijalizacija i validacija svih struktura/bafera pre mrežnih poziva; izbegavanje prosleđivanja pokazivača na objekte sa ograničenim životnim vekom.
- Postupna modernizacija koda radi čitljivosti i održivosti: `explicit` konstruktori, `const`-korektnost, bezbedni kastovi, sužavanje opsega promenljivih.

Sumirano, projekat je funkcionalno zaokružen i dobro dokumentovan, uz jasno identifikovane tehničke tačke koje unapređuju stabilnost i kvalitet koda. Primena preporučenih izmena podiže pouzdanost sistema, olakšava buduće održavanje i usmerava dalji razvoj na najuticajnije delove koda.