**Wydział Nauk Informatyczno-Technologicznych**

Kierunek studiów: **Informatyka I stopnia**

Ścieżka rozwoju: Systemy operacyjne

**Damian Piotrowski**

10510

**PROJEKT I IMPLEMENTACJA APLIKACJI**

**DO WYMIANY DANYCH W SYSTEMACH KOMUNIKACJI KLIENT-SERWER**

Praca dyplomowa inżynierska napisana pod kierunkiem:

dr. inż. Andrzeja Sawickiego

Łomża 2024

**Streszczenie**

**Tytuł: Projekt i implementacja aplikacji do wymiany danych w systemach komunikacji klient-serwer**

Celem pracy jest zaprojektowanie i zaimplementowanie aplikacji umożliwiającej wymianę danych w systemach komunikacji typu klient-serwer. Aplikacja została stworzona  
w języku Python, a rolę serwera pełni urządzenie Raspberry Pi z zainstalowanym systemem operacyjnym Linux. Komunikacja pomiędzy klientem a serwerem została zrealizowana za pomocą biblioteki socket, umożliwiającej obsługę protokołów sieciowych, oraz modułu threading, który zapewnia wielowątkowość i efektywne zarządzanie połączeniami. Graficzny interfejs użytkownika aplikacji został zaprojektowany i zaimplementowany z użyciem biblioteki tkinter, oferując intuicyjną i funkcjonalną obsługę aplikacji. Praca opisuje proces projektowania, implementacji oraz testowania aplikacji, uwzględniając wyzwania związane  
z komunikacją sieciową i wielowątkowością.

**Słowa kluczowe**

Python, Raspberry Pi, Komunikacja sieciowa, Wielowątkowość, Linux

**Summary**

**Title: Design and implementation of an application for data exchange in client-server communication systems**

The aim of this project is to design and implement an application enabling data exchange in client-server communication systems. The application was developed in Python, with a Raspberry Pi device running a Linux operating system serving as the server. Communication between the client and the server was implemented using the socket library, which facilitates network protocol handling, and the threading module, which ensures multithreading and efficient connection management. The graphical user interface of the application was designed and developed using the tkinter library, providing an intuitive and functional user experience. This work describes the process of designing, implementing, and testing the application, addressing challenges related to network communication and multithreading.

**Keywords**

Python, Raspberry Pi, Network communication, Multithreading, Linux

Spis treści

[WSTĘP 4](#_Toc188675231)

[Problem inżynierski 4](#_Toc188675232)

[Cel pracy 5](#_Toc188675233)

[Zakres pracy 5](#_Toc188675234)

[1. Analiza tematu, literatury, dostępnych rozwiązań 6](#_Toc188675235)

[1.1. Analiza literatury 6](#_Toc188675236)

[1.1.1. Client-Server Architecture - System Design 6](#_Toc188675237)

[1.2. Analiza istniejących rozwiązań 7](#_Toc188675238)

[1.3. Analiza istniejących technologii 8](#_Toc188675239)

[1.4. Wybór technologii 13](#_Toc188675240)

[2. Projekt systemu informatycznego, aplikacji, Zaproponowanego rozwiązania problemu inżynierskiego 14](#_Toc188675241)

[2.1. Omówienie projektu 14](#_Toc188675242)

[2.1.1. Zasada działania systemu informatycznego / aplikacji 14](#_Toc188675243)

[2.2. Wymagania projektu 14](#_Toc188675244)

[2.3. Projekt systemu, aplikacji, rozwiązania problemu inżynierskiego 15](#_Toc188675245)

[2.3.1. Schemat blokowy systemu 15](#_Toc188675246)

[2.3.2. Diagram klas systemu 16](#_Toc188675247)

[2.3.3. Opis działania i logiki aplikacji 16](#_Toc188675248)

[2.4. Projekt interfejsu użytkownika 17](#_Toc188675249)

[3. Implementacja aplikacji / Realizacja systemu informatycznego / Rozwiązania problemu inżynierskiego 18](#_Toc188675250)

[3.1. Podstawowe funkcjonalności 18](#_Toc188675251)

[3.2. Wybrane algorytmy 18](#_Toc188675252)

[3.3. Działanie systemu 18](#_Toc188675253)

[3.3.1. Przedstawienie wyników działania zrealizowanej aplikacji / systemu 18](#_Toc188675254)

[3.3.2. Przedstawienie wybranych ścieżek działania aplikacji /systemu 18](#_Toc188675255)

[3.4. Badania wybranych elementów systemu rozwiązującego problem inżynierski 18](#_Toc188675256)

[4. Testy systemu informatycznego / Aplikacji 19](#_Toc188675257)

[4.1. Testy systemu 19](#_Toc188675258)

[4.2. Realizacja testów 19](#_Toc188675259)

[4.2.1. Opis testu 1 19](#_Toc188675260)

[4.2.2. Opis testu 2 19](#_Toc188675261)

[4.2.3. Opis testu 2 19](#_Toc188675262)

[4.3. Wyniki testów 19](#_Toc188675263)

[5. PODSUMOWANIE I WNIOSKI 20](#_Toc188675264)

[Bibliografia 21](#_Toc188675265)

[Spis Tabel 21](#_Toc188675266)

[Spis Rysunków 21](#_Toc188675267)

[Spis Listingów 21](#_Toc188675268)

[Spis ZałĄczników 21](#_Toc188675269)

WSTĘP

Współczesne systemy komunikacji oparte na architekturze klient-serwer odgrywają kluczową rolę w budowie aplikacji sieciowych i systemów IoT. Umożliwiają one efektywną wymianę danych między różnymi urządzeniami, co znajduje zastosowanie w wielu dziedzinach, takich jak automatyka domowa, zarządzanie danymi w chmurze czy systemy monitorowania w czasie rzeczywistym.

W pracy tej skupiono się na opracowaniu aplikacji, która w sposób elastyczny i wydajny realizuje wymianę danych w architekturze klient-serwer, wykorzystując popularne narzędzia    
i technologie, takie jak Python, Raspberry Pi oraz biblioteki socket i tkinter. Takie podejście pozwala na stworzenie rozwiązania, które jest zarówno dostępne cenowo, jak i uniwersalne   
w zastosowaniu.

## Problem inżynierski

Głównym problemem inżynierskim, który rozwiązano w tej pracy, jest zaprojektowanie i implementacja systemu wymiany danych w architekturze klient-serwer, który jednocześnie:

* Zapewnia stabilną i bezpieczną komunikację pomiędzy klientem a serwerem,
* Obsługuje wiele równoczesnych połączeń dzięki zastosowaniu wielowątkowości,
* Udostępnia użytkownikom intuicyjny interfejs graficzny,
* Może działać na urządzeniach o ograniczonych zasobach sprzętowych, takich jak Raspberry Pi.

## Cel pracy

W ramach pracy powstanie aplikacja do komunikacji asynchronicznej pomiędzy użytkownikami. Celem pracy jest zaprojektowanie i implementacja aplikacji umożliwiającej przesyłanie tekstu, zdjęć, filmów i wiadomości głosowych między klientami za pośrednictwem serwera. Drugim komponentem systemu będzie aplikacja serwerowa działająca zdalnie na urządzeniu Raspberry Pi z zainstalowanym systemem operacyjnym Linux, do którego użytkownicy mogą połączyć się za pomocą adresu IP oraz portu.

## 

## Zakres pracy

Zakres pracy obejmuje:

* Analizę wymagań dotyczących systemów komunikacji klient-serwer,
* Projekt architektury aplikacji, uwzględniający specyfikę urządzeń o ograniczonych zasobach, takich jak Raspberry Pi,
* Implementację aplikacji serwera oraz klienta z wykorzystaniem języka Python,
* Wykorzystanie bibliotek socket i threading do obsługi połączeń sieciowych oraz wielowątkowości,
* Stworzenie graficznego interfejsu użytkownika za pomocą tkinter,
* Testowanie aplikacji pod kątem stabilności, wydajności oraz łatwości użytkowania,
* Dokumentację procesu projektowania, implementacji i testowania aplikacji, a także wnioski dotyczące możliwości rozwoju systemu.

# Analiza tematu, literatury, dostępnych rozwiązań

## Analiza literatury

### Client-Server Architecture - System Design

W artykule “Client-Server Architecture - System Design” zamieszczonym na stronie geeksforgeeks.org autorzy analizują rozwiązania związane z architekturą klient-serwer.

Autorzy podają, że architektura klient-serwer jest podstawą nowoczesnego projektowania systemów, gdzie infrastruktura składa się z jednego serwera centralnego oraz wielu klientów. Autorzy zaznaczają, że w tym modelu klient rozumiany jest jako urządzenie lub program, który wykonuje żądania usług lub zasobów a serwer jest maszyną lub programem, który na te żądania odpowiada. Komunikacja pomiędzy klientami odbywa się za pomocą protokołów żądanie-odpowiedź, takich jak HTTP/HTTPS dla systemów sieciowych, bądź SQL dla systemów bazodanowych.

Główne cechy systemu klient-serwer wynikające z tekstu artykułu:

* Wydajne zarządzanie danymi oraz przyporządkowywanie zasobów systemu centralizując krytyczne funkcje serwerowi,
* Klient zarządza interakcją z użytkownikiem i wysyła specyficzne żądania do serwera, który przetwarza te żądania i odpowiada w należyty sposób,
* Duża skalowalność systemu, który jest w stanie przyjąć więcej klientów dzięki skalowaniu zdolności serwera, bądź dodanie nowych serwerów,
* Daje podstawy pod bardziej skomplikowane systemy.

Artykuł podkreśla krytyczne znaczenie architektury klient-serwer dla projektowania systemów. Poprzez centralizację zasobów systemu, ta architektura upraszcza w dużym stopniu utrzymanie, aktualizacje oraz bezpieczeństwo systemu. Administratorzy są w stanie efektywnie monitorować i zarządzać danymi, wgrywać aktualizacje oraz egzekwować zasady bezpieczeństwa z jednego miejsca. System pozwala na dużą skalowalność poprzez dodanie nowych serwerów lub zwiększenia możliwości istniejących serwerów, co pozwala rozwijać system, tak aby mógł on obsługiwać dowolną liczbę klientów, nie ingerując przy tym w ogólną architekturę systemu. Autorzy podają także, że takie rozwiązanie niesie za sobą korzyści  
w aspektach optymalizacji zarządzania zasobami, niezawodności oraz bezpieczeństwa. Według autorów centralizowane serwery pozwalają na wprowadzenie lepszych protokołów bezpieczeństwa oraz ochrony danych.

Architektura klient-serwer zakłada istnienie kilku kluczowych składników, które działają razem, aby zapewnić wydajną komunikację. Są to klient, serwer, sieć, protokoły, urządzenia pośrednie, baza danych, interfejs użytkownika i logika aplikacji.

Autorzy podają kilka rozwiązań oraz narzędzi dla architektury klient-serwer. W ich skład wchodzą frameworki oraz programy dla serwera, klienta oraz bazy danych:

* React – biblioteka JavaScript służąca do budowania interfejsów użytkownika, utrzymywany przez Facebook,
* Django – wysokopoziomowy framework Pythona, który zawiera wbudowany interfejs admina oraz mechanizmy potwierdzania tożsamości,
* MySQL – system do prowadzenia baz danych. Działa on w formacie open-source.

Protokoły, API oraz narzędzia do budowania aplikacji:

* REST – styl architektury do projektowania aplikacji sieciowych. Używa standardowego protokołu HTTP,
* WebSocket – protokół pozwalający na pełną transmisję dwukierunkową używając jednego połączenia TCP. Idealne rozwiązanie dla aplikacji czasu rzeczywistego,
* Git – system kontroli wersji, służący do śledzenia zmian w kodzie przy projektowaniu aplikacji. Niezbędny do grupowej pracy nad aplikacjami.

Autorzy podają, że takie rozwiązanie ma szerokie zastosowanie w świecie rzeczywistym, w dziedzinach takich jak: bankowość, telekomunikacja, internet rzeczy czy systemy rządowe.

## Analiza istniejących rozwiązań

**Gotowe frameworki sieciowe**:

Frameworki takie jak Flask czy Django oferują zaawansowane mechanizmy komunikacyjne, ale są stosunkowo ciężkie dla urządzeń o ograniczonych zasobach, jak Raspberry Pi.

**Protokół MQTT**:

Lekki protokół często używany w systemach IoT. Zapewnia wysoką wydajność, ale wymaga dodatkowych brokerów danych, co komplikuje implementację w małych projektach.

**Protokół HTTP/HTTPS:**

Używany w aplikacjach webowych do komunikacji klient-serwer. Serwer odbiera żądania od klienta, przetwarza je, a następnie na nie odpowiada.

**Web Socket:**

Protokół umożliwiający komunikację pełnodupleksową w czasie rzeczywistym, wykorzystywany w aplikacjach wymagających natychmiastowej wymiany danych (np. czaty, powiadomienia).

## Analiza istniejących technologii

**Python**:

Python to jeden z najpopularniejszych języków programowania, ceniony za swoją prostotę, czytelność kodu oraz szeroki ekosystem bibliotek i narzędzi. Dzięki temu jest idealnym wyborem do implementacji aplikacji klient-serwer, zarówno po stronie serwera, jak i klienta. Python wspiera różnorodne protokoły komunikacyjne, takie jak TCP, UDP, czy HTTP, co pozwala na tworzenie systemów dostosowanych do specyficznych potrzeb.

* **Prostota składni**: Python umożliwia szybkie prototypowanie i rozwój aplikacji. Intuicyjna składnia sprawia, że jest przystępny zarówno dla początkujących, jak i doświadczonych programistów.
* **Bogaty ekosystem**: Python oferuje wiele bibliotek i frameworków, takich jak Flask, Django (do aplikacji webowych) czy Asyncio (do obsługi asynchronicznej).
* **Przenośność**: Kod Python działa na różnych systemach operacyjnych, co czyni go wszechstronnym narzędziem dla różnych platform sprzętowych i programowych.

**Raspberry Pi**:

Raspberry Pi to kompaktowy komputer jednopłytkowy, który stał się kluczowym elementem wielu projektów edukacyjnych, IoT (Internet of Things) oraz systemów automatyki. Dzięki pełnej obsłudze systemu Linux oraz wsparciu dla Python, Raspberry Pi doskonale nadaje się do budowy systemów klient-serwer.

* **Wszechstronność sprzętowa**: Raspberry Pi obsługuje różnorodne interfejsy komunikacyjne, takie jak Ethernet, Wi-Fi, Bluetooth, czy GPIO, co pozwala na integrację z sensorami i innymi urządzeniami IoT.
* **Ekosystem Linuksa**: Umożliwia instalację bibliotek sieciowych, serwerów HTTP oraz narzędzi do monitorowania ruchu sieciowego.
* **Niski koszt**: Raspberry Pi jest przystępne cenowo, co czyni je idealnym narzędziem dla małych projektów i prototypowania.

**Biblioteka socket**:

Biblioteka socket w Python pozwala na niskopoziomową komunikację sieciową, umożliwiając tworzenie zarówno serwerów, jak i klientów. Oferuje pełną kontrolę nad procesem przesyłania danych, co pozwala na implementację protokołów dostosowanych do konkretnych wymagań.

##### Kluczowe zalety:

* **Tworzenie gniazd**: socket umożliwia konfigurację gniazd TCP i UDP, które obsługują połączenia niezawodne (TCP) oraz szybką wymianę danych (UDP).
* **Obsługa protokołów**: Oferuje wsparcie dla protokołów IPv4 i IPv6, co pozwala na obsługę nowoczesnych sieci.
* **Elastyczność**: Biblioteka pozwala na wysyłanie i odbieranie danych w różnych formatach (tekst, binarny).
* **Łatwość integracji**: Może być używana razem z innymi bibliotekami, np. do szyfrowania transmisji (SSL/TLS).

**Biblioteka threading**:

Biblioteka threading w Python umożliwia tworzenie wielowątkowych aplikacji, co jest kluczowe w systemach obsługujących wielu klientów jednocześnie. W połączeniu z socket pozwala na implementację serwerów, które mogą efektywnie zarządzać równoczesnymi połączeniami.

##### Kluczowe zalety:

* **Obsługa wielu klientów**: Każde połączenie może być obsługiwane w osobnym wątku, co zapobiega blokowaniu serwera przez długotrwałe operacje.
* **Łatwa implementacja**: Python dostarcza prosty interfejs do tworzenia wątków, co przyspiesza proces projektowania.
* **Integracja z innymi bibliotekami**: threading współpracuje z bibliotekami takimi jak queue, co pozwala na efektywne zarządzanie zadaniami między wątkami.

**Biblioteka Tkinter**:

Tkinter to wbudowana biblioteka Python do tworzenia graficznych interfejsów użytkownika (GUI). Jest lekka, łatwa w użyciu i umożliwia szybkie prototypowanie aplikacji, które wymagają interakcji użytkownika.

##### Kluczowe zalety:

* **Prostota i intuicyjność**: Umożliwia szybkie tworzenie okien, przycisków, pól tekstowych i innych elementów interfejsu.
* **Integracja z innymi bibliotekami**: Tkinter może współpracować z bibliotekami sieciowymi, umożliwiając tworzenie klientów z graficznym interfejsem.
* **Przenośność**: Działa na większości systemów operacyjnych obsługujących Python.

**Biblioteka cryptography:**

Biblioteka **cryptography** w Pythonie oferuje zaawansowane narzędzia do szyfrowania i deszyfrowania danych, co jest niezbędne w aplikacjach, które wymagają bezpiecznej wymiany informacji. Umożliwia korzystanie z różnych algorytmów kryptograficznych, zarówno symetrycznych, jak i asymetrycznych, oraz zapewnia łatwą implementację różnych metod uwierzytelniania i ochrony integralności danych.

#### Kluczowe zalety:

* **Szyfrowanie:** Umożliwia wykorzystanie algorytmów AES,RSA
* **Algorytmy hashujące: cryptography** wspiera różne algorytmy hashujące, takie jak SHA-256, które są wykorzystywane do zapewnienia integralności danych.
* **Wsparcie dla SSL/TLS:** Biblioteka pozwala na użycie protokołów SSL i TLS, co umożliwia tworzenie bezpiecznych połączeń między klientami i serwerami.
* **Łatwość użycia i dokumentacja: cryptography** dostarcza prosty interfejs API i jest dobrze udokumentowana, co ułatwia implementację szyfrowania w aplikacjach Pythonowych. Działa na wielu systemach operacyjnych i jest odpowiednia dla różnych zastosowań, od aplikacji webowych po systemy IoT.

**GitHub:**

GitHub to platforma hostingowa dla projektów programistycznych, która wykorzystuje system kontroli wersji Git. Jest nie tylko miejscem do przechowywania kodu, ale także narzędziem do współpracy zespołowej, zarządzania projektami i automatyzacji procesów deweloperskich.

**Kluczowe zalety:**

* **System kontroli wersji:** GitHub pozwala na śledzenie zmian w kodzie, co umożliwia łatwe cofanie błędnych modyfikacji i analizowanie historii projektu.
* **Współpraca zespołowa:** Funkcje takie jak pull requesty, code review i zarządzanie zgłoszeniami błędów ułatwiają współpracę między członkami zespołu.
* **Automatyzacja:** GitHub umożliwia automatyzację testów, budowania i wdrażania aplikacji.
* **Integracja z narzędziami:** Platforma współpracuje z wieloma narzędziami zewnętrznymi, np. Jenkins, Slack, czy Docker.

**Docker:**

Docker to platforma umożliwiająca konteneryzację aplikacji, co sprawia, że aplikacje mogą być uruchamiane w izolowanym środowisku niezależnie od systemu operacyjnego. Dzięki temu jest szczególnie użyteczny w projektach klient-serwer, gdzie wymagane jest łatwe wdrażanie i skalowanie aplikacji.

**Kluczowe zalety:**

* **Izolacja środowisk:** Każda aplikacja działa w swoim kontenerze, co eliminuje konflikty między zależnościami.
* **Przenośność:** Kontenery można uruchamiać na różnych platformach (serwery lokalne, chmura).
* **Szybkość wdrażania:** Docker pozwala na szybkie uruchamianie środowisk testowych i produkcyjnych.

**Flask:**

Flask to lekki framework webowy dla Pythona, idealny do szybkiego tworzenia aplikacji serwerowych. Dzięki prostocie i elastyczności, Flask jest często wykorzystywany w systemach klient-serwer, które wymagają łatwego zarządzania API.

**Kluczowe zalety:**

* **Minimalistyczna struktura:** Flask pozwala na tworzenie aplikacji bez zbędnych komplikacji.
* **Wsparcie dla API:** Prosta integracja z REST i WebSocket.
* **Rozszerzalność:** Flask obsługuje wiele wtyczek, np. Flask-SocketIO do komunikacji w czasie rzeczywistym.
* **Elastyczność:** Framework daje pełną kontrolę nad strukturą projektu.

**SQLite:**

SQLite to lekka, wbudowana baza danych, która doskonale sprawdza się w mniejszych projektach klient-serwer.

**Kluczowe zalety:**

* **Bezserwerowość:** SQLite działa jako plik lokalny i nie wymaga instalacji serwera bazy danych.
* **Łatwość użycia:** Prosty interfejs API ułatwia integrację z Python.
* **Przenośność:** Baza danych może być łatwo przenoszona między systemami.

## Wybór technologii

Na podstawie analizy dostępnych technologii zdecydowano się na:

* **Python** jako główny język programowania, ze względu na jego prostotę, wsparcie dla wieloplatformowych bibliotek oraz aktywną społeczność.
* **Socket i threading** do realizacji połączeń sieciowych oraz obsługi wielowątkowości, co zapewnia pełną kontrolę nad architekturą aplikacji.
* **Cryptography** jako biblioteka zapewniająca bezpieczeństwo danych poprzez szyfrowanie i uwierzytelnianie. Umożliwia implementację protokołów SSL/TLS, szyfrowania AES/RSA, a także tworzenie hashy dla zapewnienia integralności danych w aplikacji klient-serwer.
* **Tkinter** do budowy graficznego interfejsu użytkownika, jako narzędzie zintegrowane  
  z Pythonem i odpowiednie dla projektów o niewielkiej złożoności wizualnej.
* **Raspberry Pi** jako serwer, dzięki jego niskiemu zużyciu energii, małym wymiarom oraz możliwości pracy w środowisku Linux.
* **GitHub** jako platforma do kontroli wersji. Zapewnia możliwość zarządzania kodem, przechowywania historii zmian, a także automatyzacji procesów co przyspiesza testowanie i wdrażanie aplikacji.
* **MobaXterm** jako narzędzie do zarządzania zdalnymi połączeniami, co umożliwia łatwą konfigurację i monitorowanie Raspberry Pi. Dzięki wbudowanym funkcjom terminal SSH, SCP, czy X11-forwarding, MobaXterm upraszcza pracę w środowiskach sieciowych.
* **Dodatkowe biblioteki takie jak os, sys, time, datetime, traceback oraz errno służące do synchronizacji wymiany danych pomiędzy klientami a serwerem, a także debugowania kodu aplikacji.**

Wybór tych technologii umożliwia stworzenie aplikacji, która jest wygodna w użyciu, funkcjonalna, elastyczna i zoptymalizowana pod kątem działania na urządzeniach o ograniczonych zasobach sprzętowych.

# Projekt systemu informatycznego, aplikacji, Zaproponowanego rozwiązania problemu inżynierskiego

## Omówienie projektu

### Zasada działania systemu informatycznego / aplikacji

Celem projektu jest stworzenie aplikacji umożliwiającej komunikację w architekturze klient-serwer, zapewniającej stabilną wymianę danych pomiędzy użytkownikami. System składa się z dwóch głównych komponentów: serwera działającego na Raspberry Pi oraz aplikacji klienckiej uruchamianej na komputerze lub innym urządzeniu z systemem operacyjnym obsługującym Pythona. Serwer zarządza połączeniami wielowątkowymi oraz wymianą danych, a aplikacja kliencka oferuje użytkownikowi graficzny interfejs do interakcji z systemem.

## Wymagania projektu

**Funkcjonalne**:

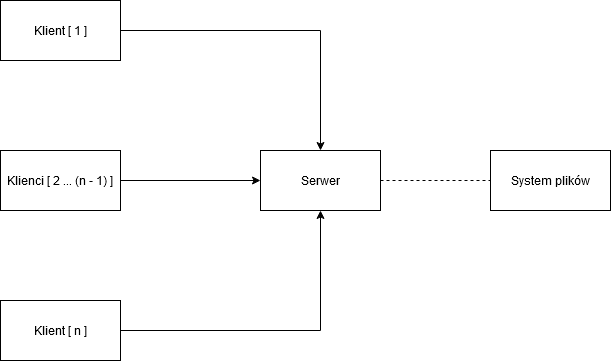
* Obsługa wielu klientów jednocześnie,
* Wysyłanie i odbieranie danych w czasie rzeczywistym,
* Prosty, intuicyjny interfejs graficzny.

**Niefunkcjonalne**:

* Działanie serwera na Raspberry Pi z systemem Linux,
* Minimalne opóźnienia w komunikacji,
* Bezpieczeństwo danych,
* Łatwa instalacja i konfiguracja aplikacji na urządzeniach klienckich.

## Projekt systemu, aplikacji, rozwiązania problemu inżynierskiego

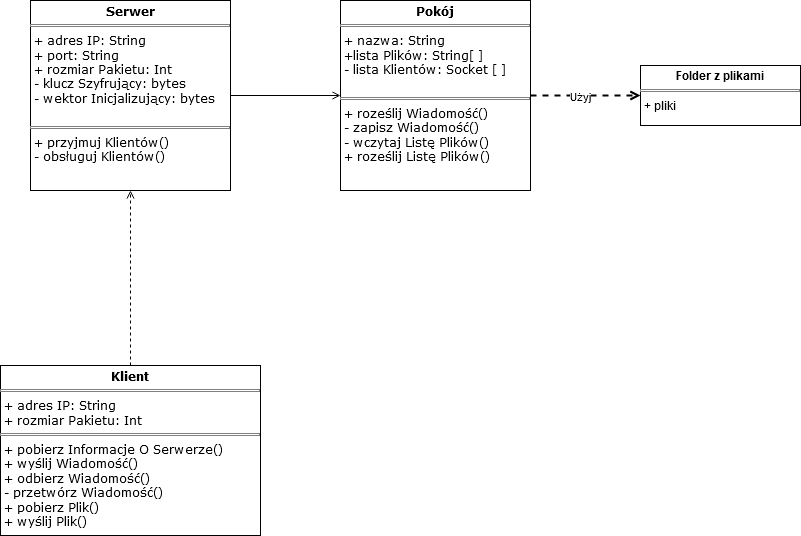
### Schemat blokowy systemu



Rysunek 2.1 Schemat blokowy systemu

Na rysunku 2.1 przedstawiony jest schemat blokowy aplikacji. Serwer jest połączony z systemem plików przez system operacyjny. Serwer może przyjąć n klientów, gdzie n zależy od zasobów systemowych urządzenia, na którym działa serwer.

### Diagram klas systemu



Rysunek 2.2 Diagram klas systemu

Na rysunku 2.2 przedstawiono diagram klas systemu, reprezentujący klasy, które biorą udział w wymianie danych w ramach działania aplikacji. Klasa „Pokój” istnieje wewnątrz klasy „Serwer” przechowuje listę nazw własnych plików oraz listę klientów obecnie zalogowanych do danego pokoju. Klasa „Serwer” przechowuje dane konieczne do odbierania i przetwarzania połączeń oraz wiadomości. Serwer wykorzystuje pokój do rozsyłania wiadomości do odpowiednich klientów. Klient zazwyczaj (nie jest to konieczne) istnieje na osobnym urządzeniu i posiada własny interfejs użytkownika. Przechowuje on klucz oraz wektor szyfrujący i jest w stanie wysyłać oraz odbierać wiadomości od serwera.

### Opis działania i logiki aplikacji

Poniżej znajduje się schemat działania aplikacji:

* Administrator uruchamia serwer,
* Serwer pobiera z pliku dane konfiguracyjne, takie jak adres IP hosta oraz port,
* Serwer generuje klucz AES oraz wektor IV, dane te są unikatowe dla każdego uruchomienia serwera,
* Serwer inicjalizuje się wczytując listę zarejestrowanych użytkowników,
* Serwer inicjalizuje pokoje jako podfoldery aplikacji. Każdy z nich ma osobny plik czatu oraz osobny folder z plikami,
* Klient uruchamia aplikację, rejestruje się i loguje,
* Po zalogowaniu klient otrzymuje klucz AES od serwera, którym będzie szyfrował oraz rozszyfrowywał wiadomości,
* Po nawiązaniu połączenia serwer tworzy nowy wątek dedykowany klientowi,
* Klient wysyła dane do serwera, który je odbiera i odpowiednio przetwarza,
* Serwer reaguje na odebrane wiadomości i wysyła na ich podstawie odpowiednie wiadomości do klientów, co jest wizualizowane w graficznym interfejsie.

## Projekt interfejsu użytkownika

# Implementacja aplikacji / Realizacja systemu informatycznego / Rozwiązania problemu inżynierskiego

## Podstawowe funkcjonalności

## Wybrane algorytmy

## Działanie systemu

### Przedstawienie wyników działania zrealizowanej aplikacji / systemu

### Przedstawienie wybranych ścieżek działania aplikacji /systemu

## Badania wybranych elementów systemu rozwiązującego problem inżynierski

# Testy systemu informatycznego / Aplikacji

## Testy systemu

## Realizacja testów

### Opis testu 1

### Opis testu 2

### Opis testu 2

## Wyniki testów

# PODSUMOWANIE I WNIOSKI

Bibliografia

1. https://www.geeksforgeeks.org/client-server-architecture-system-design/

Spis Tabel

**Nie można odnaleźć pozycji dla spisu ilustracji.**

Spis Rysunków

**Nie można odnaleźć pozycji dla spisu ilustracji.**

Spis Listingów

Nie można odnaleźć pozycji dla spisu ilustracji.

Spis ZałĄczników

1. Kod źródłowy aplikacji – załącznik w APD, plik *Aplikacja\_kod\_zrodlowy.zip*
2. Instrukcja instalacji i uruchomienia aplikacji - załącznik w APD, plik *Aplikacja\_Instrukcja\_instalacji.zip*
3. Płyta DVD z projektem aplikacji, bazą danych i wersją instalacyjna aplikacji.