

# Politechnika Poznańska

## Wydział Elektryczny



Projekt zespołowy

## **Monitorowanie stacji laboratoryjnych**

<https://github.com/dawid.kulinski/PT-Projekt>

Dawid Kuliński 126820 – [dawid.kulinski@student.put.poznan.pl](mailto:dawid.kulinski@student.put.poznan.pl)

Prowadzący:

Mgr. Inż. Przemysław Walkowiak

## Spis treści

Uzasadnienie wyboru tematu.....	4
Scentralizowane zarządzanie routerami OpenWrt.....	4
Rozpoznawanie obrazu z gry w warcaby oraz wizualizacja stanu gry na komputerze.....	5
Monitorowanie stacji laboratoryjnej.....	5
Wnioski.....	6
Założenia projektu.....	6
Stacja monitorowania.....	6
Stacja kontrolująca.....	7
Podział prac pomiędzy członków zespołu.....	7
Funkcjonalność systemu.....	9
Wybrane technologie.....	11
Część wspólna.....	11
Stacja Monitorująca.....	11
Stacja Kontrolująca.....	12
Narzędzia, środowisko, biblioteki.....	12
Stacja Kontrolująca.....	12
Stacja Monitorująca.....	13
Kontrola wersji.....	13
Narzędzia.....	13
Architektura rozwiązania.....	13
Stacja Komputerowa.....	14
Stacja Kontrolująca.....	16
Interesujące problemy i rozwiązania ich na jakie się natknąłem.....	18
Wnioski.....	19
Opis modelu komunikacji.....	20
Komunikacja bezpośrednia.....	20
Komunikacja za pomocą kluczy trasowania.....	22
Wykorzystanie Topiców.....	23
Ostateczny model komunikacji.....	25

Instrukcja użytkowania aplikacji.....	25
MongoDB.....	25
Elixir.....	26
RabbitMQ.....	26
Visual Studio.....	28
Uruchomienie aplikacji.....	28
Stacja Kontrolująca.....	28
Stacja Monitorowania.....	28
Wyświetlane widoki.....	29
Możliwości rozwoju.....	31
Przetworzenie serwera do obsługi gier sieciowych czasu rzeczywistego.....	31
Rozszerzenie funkcjonalności Stacji Monitorowania.....	31
Praca w przestrzeni jądra.....	31
Wykrywanie podłączenia nowych urządzeń.....	31
Wywoływanie zdalnych procedur.....	32
Podsumowanie.....	32
Źródła wiedzy.....	32

## Uzasadnienie wyboru tematu

Ostateczny wybór tematu polegał na przeanalizowaniu argumentów za i przeciw odnoszących się do trzech wytypowanych projektów na podstawie subiektywnych reakcji:

- Scentralizowane zarządzanie routerami OpenWrt.
- Rozpoznawanie obrazu z gry w warcaby oraz wizualizacja stanu gry na komputerze.
- Monitorowanie pracowni laboratoryjnej.

## Scentralizowane zarządzanie routerami OpenWrt

Jest to temat możliwy do dalszego wykorzystania nawet w warunkach komercyjnych. Wiele firm posiada własne sieci, które często są zbyt duże, żeby można było posiadać pełną kontrolę nad konfiguracją. Oprogramowanie mogłoby rozwiązywać następujące problemy:

- Dynamika infrastruktury sieciowej wymaga jej częstych zmian, które wynikają z częstą zmianą pracy przez pracowników.
- Braki kadrowe wśród specjalistów IT.
- Wymogi biznesowe, które kładą nacisk na jak najszybsze wdrożenie nowej konfiguracji.

Powyższe problemy mogą prowadzić do powstania problemów z bezpieczeństwem. Wiele urządzeń zostaje zapomnianych przez administratorów, przez co ich oprogramowanie nie jest aktualne, a konfiguracja może być przestarzała.

Rozwiązanie byłoby zaprogramowane za pomocą narzędzia służącego do przeprowadzania procesu, ciągłego wdrożenia o nazwie Jenkins. Gdzie każdy router stanowiłby klienta SSH, za pomocą którego dostarczane byłyby informacje dotyczące aktualizacji konfiguracji. Notyfikacja możliwa jest poprzez wykorzystanie serwera GIT, który stanowiłby bazę danych zawierającą konfiguracje wszystkich routerów. Po wykryciu zmiany w repozytorium każdy klient wykonywałby skrypt, którego zadaniem byłoby zaktualizowanie konfiguracji.

Niestety pomysł na przeprowadzenie danego projektu wymagałby bardzo małego nakładu wytworzenia własnego oprogramowania, przez co pojawia się duże ryzyko niepowodzenia projektu wynikające z braku możliwości wprowadzenia własnych zmian w oprogramowaniu. Ze względu na ten sam problem wynikałaby niska możliwość monitorowania przeprowadzonych prac, ze względu na oparcie prac o konfigurację gotowych rozwiązań.

## Rozpoznawanie obrazu z gry w warcaby oraz wizualizacja stanu gry na komputerze

Kolejny temat dawał duże możliwości rozwoju po przeprowadzeniu projektu. Projekt nie posiada dużych możliwości komercyjnych, które umożliwiłyby jego ciągłe udoskonalanie. Jedyna możliwość wykorzystania podczas turnieju w gry w warcaby, gdzie oprogramowanie mogłoby stanowić sędziego gry.

Implementacja danego projektu odbywałaby się za pomocą biblioteki OpenCV (do wyboru język programowania C++ lub Python). Gdzie za pomocą operacji macierzowych przeprowadzane byłyby modyfikacje obrazu w celu rozpoznania poszczególnych pionków, które byłyby zwizualizowane, oraz zweryfikowane, czy ruch odbył się zgodnie z zasadami.

Uwagę należy zwrócić na możliwości rozwoju danej aplikacji, poprzez zastosowanie sieci neuronowych. System mógłby stanowić mechanizm podpowiedzi ruchu jak i funkcję nauki gracza. Nauka odbywałaby się poprzez wprowadzenie bardzo dużej liczby partii prowadzonych przez profesjonalnych graczy, przez co możliwe byłoby uniknięcie błędów osoby początkującej.

## Monitorowanie stacji laboratoryjnej

Temat projektu wydawał się najbardziej interesujący, a zarazem złożony. Monitorowanie pracowni laboratoryjnej może stanowić bardzo dobrą bazę do wprowadzenia własnego rozwiązania. W pierwszej kolejności wspomniany system może przydać się do następujących sytuacji:

- Weryfikacja, czy studenci z odpowiednią intensywnością pracują na zajęciach.
- Weryfikacja, czy studenci nie korzystają z zewnętrznych źródeł informacji podczas kolokwium.
- Weryfikacja listy obecności.

Rozwiązanie powyższych problemów może znacznie ułatwić pracę prowadzącego, przez co mógłby więcej energii przeznaczyć na tworzenie nowych zadań na laboratoria lub odkrywaniem nowych zastosowań dla informatyki.

Rozwiązanie mogłoby być zaimplementowane za pomocą implementacji protokołu AMQP o nazwie RabbitMQ, które odpowiadałoby za cały model komunikacji. Poprzez wykorzystanie przedstawionego protokołu zapewniana jest dostępność, integralność i poufność komunikacji.

## Wnioski

Ostatecznie wybrany został projekt o nazwie „Monitorowanie stacji laboratoryjnej”, ze względu na możliwość wykorzystania bibliotek nowoczesnych, często wykorzystywanych w dużych firmach, oraz odpowiedni balans pomiędzy programowaniem a konfigurowaniem serwerów.

Decydującym argumentem była możliwość dalszego rozwoju aplikacji, która zostanie dokładnie opisana w rozdziale Możliwości rozwoju.

## Założenia projektu

Przewidziane jest utworzenie systemu składającego się z dwóch głównych części:

- Stacja monitorowania
- Stacja kontrolująca

Wymagania stawiane przed projektem koncentrowały się na zapewnieniu odpowiedniej funkcjonalności, która stanowić będzie ostatecznie dowód w postaci „proof-of-concept” możliwości dalszego rozwoju aplikacji. Ostatecznie poziom gotowości technologicznej projektu po jego zakończeniu w ramach zajęć Podstawy Teleinformatyki planowany jest na TRL 4. Przewidziane zostały następujące wymagania funkcjonalne.

## Stacja monitorowania

Stanowić będzie oprogramowanie agentowe, działające w tle systemu operacyjnego. Przeprowadzać będzie operację monitorowania podstawowych własności systemu operacyjnego i przysyłać je do Stacji Kontrolującej.

- Kompatybilny z Windows 10
- Podgląd ekranów stacji komputerowej
  - Tworzenie zrzutów ekranu stacji komputerowej.
  - Wybór rozdzielczości utworzonego zrzutu.
  - Wysyłanie utworzonych zrzutów.
- Podgląd uruchomionych procesów.
  - Utworzenie listy aktualnie uruchomionych procesów w systemie.
  - Monitorowanie utworzenia procesu.
  - Przesyłanie informacji o uruchomionych procesach.
  - Blokowanie niedozwolonych procesów.
- Informowanie użytkownika o połączeniu ze Stacją kontrolującą

## Stacja kontrolująca

Stanowić będzie oprogramowanie serwerowe, działające w trybie ciągłym na serwerze. Jej zadaniem będzie zapisywanie otrzymanych danych ze Stacji Monitorujących do bazy danych, oraz przetwarzanie ich w celu przedstawienia użytkownikowi.

- Kompatybilny z systemem operacyjnym Windows 10
- Kompatybilny z systemem operacyjnym Linux
- Podgląd stanowisk komputerowych
  - Utworzenie lisy aktywnych procesów
  - Zapis listy utworzonych procesów użytkownika.
  - Zapis zrzutów ekranu użytkownika.
  - Podgląd informacji zebranych o użytkowniku.
- Konfiguracja kont klienta
  - Zmiana konfiguracji klienta
- Autoryzacja użytkowników
  - Autoryzacja za pomocą certyfikatów.
  - Podział użytkowników na grupy.

## Podział prac pomiędzy członków zespołu

Ze względu na pracę w grupie jednoosobowej wszystkie zadania zostały przekazane do wykonania mnie, czyli Dawidowi Kulińskiemu. Zadania zlecone zostały na samym wstępie projektu w trakcie przygotowywania prezentacji wraz z przewidzianym ich terminem wykonania, które zostaną przekazane poniżej.

Zadanie	Termin wykonania
Przesyłanie komunikatów za pomocą Rabbit MQ	09.12.2018
Monitorowanie utworzenia nowego procesu	09.12.2018

Zbieranie informacji o utworzonych procesach	09.12.2018
Przesyłanie informacji o utworzonych procesach	09.12.2018
Zapis listy utworzonych procesów użytkownika	09.12.2018
Przesyłanie komunikatów za pomocą Rabbit MQ	09.12.2018
Monitorowanie utworzenia nowego procesu	09.12.2018
Zbieranie informacji o utworzonych procesach	09.12.2018
Przesyłanie informacji o utworzonych procesach	09.12.2018
Zapis listy utworzonych procesów użytkownika	09.12.2018
Zapis zrzutów ekranu użytkownika.	19.01.2019
Podgląd informacji zebranych o użytkowniku.	19.01.2019
Zmiana konfiguracji klienta.	19.01.2019
Blokowanie niedozwolonych procesów, oraz połączeń.	19.01.2019
Obecność komunikatu w postaci "MessageBox" lub "Tray Icon".	19.01.2019

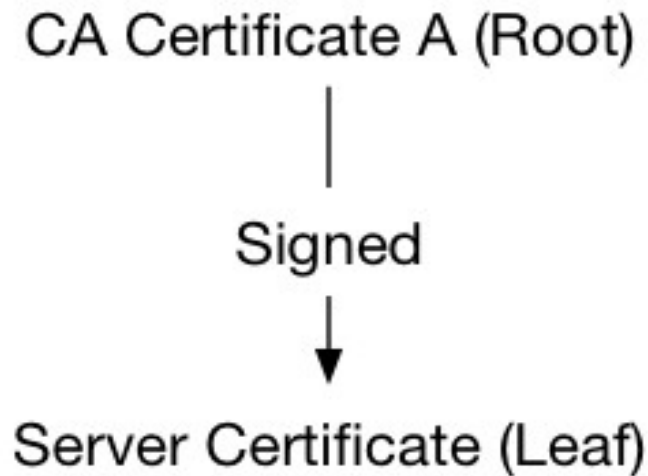
## Funkcjonalność systemu

System jest przygotowany do pracy w systemie operacyjnym Windows 10. Dodatkowo aplikacja serwerowa umożliwia działanie jej na systemie operacyjnym Linux. Umożliwione jest to poprzez



wykorzystanie języka Elixir, który działa w środowisku uruchomieniowym języka Erlang, który jest niezależny od platformy, na której aktualnie pracuje.

System jest odpowiednio odizolowany od sieci zewnętrznej w celu zwiększenia jego bezpieczeństwa. Odbywa się to poprzez mechanizm uwierzytelniania opierającego się o certyfikaty.



*Rysunek 1: Weryfikacja certyfikatu*

Powyższy mechanizm opiera się na weryfikowaniu podpisów cyfrowych złożonych pod certyfikatem serwera przez urząd certyfikacji, któremu ufa. W ten sposób dostępu do systemu nie mogą uzyskać przypadkowe osoby.

W przypadku uwierzytelnienia użytkownik stacji komputerowej nie ma rzeczywistego wpływu na działanie aplikacji, ponieważ działa ona w tle i monitoruje aktualnie wykonywane operacje. Wydobywane są następujące informacje:

- Informacja o aktualnie pracującym użytkowniku
- Informacja o aktualnie uruchomionych procesach działających w systemie operacyjnym Windows.
- Zrzuty ekranu wydobywane podczas pracy.

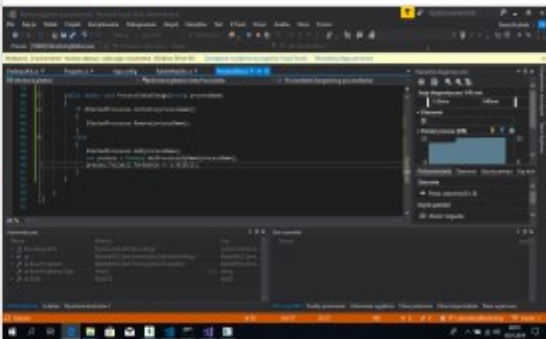
Na podstawie zebranych informacji prowadzący zajęcia może wykonać akcję na Stacji Kontrolującej:

- Podejrzenie listy uruchomionych procesów.
- Zablokowanie podejrzanego procesu.

- Podejrzenie aktualnie przeglądanych informacji na ekranie komputerowym.
- Zwiększenie rozdzielczości przeglądanej obrazu.

Powyższe operacje mogą być dokonywane poprzez interfejs internetowy. W pierwotnej wersji systemu dostępny on będzie tylko poprzez zalogowanie się jako lokalny użytkownik, jednak w kolejnych etapach możliwe będzie wprowadzenie mechanizmów logowania.

## Aktywni użytkownicy

Nazwa użytkownika	Uruchomione procesy	Stan ekranu
client1	<pre>cmd notepad backgroundTaskHost erl epmd</pre>	

Rysunek 2: Przykładowy widok ekranu zarządzania

Na powyższym rysunku przedstawiono przykładowy widok ekranu zarządzania. Ze względu na niskie umiejętności dotyczące programowania interfejsów użytkownika widok został zaprojektowany w postaci tabeli o trzech kolumnach:

- Nazwa użytkownika  
Informacja zawierająca identyfikator użytkownika.
- Uruchomione procesy  
Lista wszystkich uruchomionych procesów. Po kliknięciu na nazwę procesu wysyłany jest sygnał do wskazanej stacji, że proces powinien zostać zamknięty.
- Stan ekranu  
Najświeższy zrzut ekranu otrzymany od użytkownika.

## Wybrane technologie

Technologie zostały wybrane biorąc pod uwagę specyfikę projektu w celu jak najlepszego dopasowania jej do wymogów funkcjonalnych. W szczególności należało zwrócić uwagę, że Stacja Monitorująca powinna działać na systemie Windows 10, a Stacja Kontrolująca być niezależna od platformy sprzętowej.

Ze względu na tą specyfikację dobrany został następujący stos technologiczny.

## Część wspólna

Elementem wspólnym dostępnym dla każdego elementu systemu jest biblioteka RabbitMQ, która stanowi główny element komunikacji.

RabbitMQ jest lekkim i prostym we wdrożeniu oprogramowaniem wspierającym wiele protokołów służących do przekazywania wiadomości (ang. messaging protocol). Implementuje on protokół AMQP (Advanced Message Queuing Protocol), który może być rozszerzany przez różnego rodzaju wtyczki, które rozszerzają możliwości danego oprogramowania. AMQP jest to otwarty standard protokołu warstwy aplikacji dla oprogramowania pośredniczącego zorientowanego komunikatowo. Cechy określające AMQP to zorientowanie komunikatowe, kolejkovanie, trasowanie, niezawodność i bezpieczeństwo. AMQP określa zachowanie usługi oraz klienta komunikacji w stopniu, który powoduje, że implementacje różnych dostawców są interoperacyjne, w taki sam sposób jak SMTP, HTTP, FTP i tym podobne stworzyły interoperacyjne systemy.

## Stacja Monitorująca

Wykorzystane zostało środowisko uruchomieniowe .NET, które jest najbardziej kompatybilne z systemem operacyjnym Windows.

C# jest to obiektowy język programowania zaprojektowany w latach 1998-2001 przez zespół pod kierunkiem Andersa Hejlsberga dla firmy Microsoft. Program napisany w tym języku jest kompilowany do języka Common Intermediate Language (CIL), kodu pośredniego wykonywanego w środowisku uruchomieniowym takim jak .NET Framework.

.NET Framework platforma programistyczna opracowana przez Microsoft, obejmująca środowisko uruchomieniowe (Common Language Runtime – CLR) oraz biblioteki klas dostarczane standardowej funkcjonalności dla aplikacji. Zadaniem platformy .NET Framework jest zarządzanie różnymi elementami systemu: kodem aplikacji, pamięcią i zabezpieczeniami.

## Stacja Kontrolująca

Wykorzystane zostało środowisko uruchomieniowe erlang, na którym interpretowany był kod w języku Elixir.

Elixir jest to funkcyjny i współbieżny język programowania stworzony w 2012 roku przez Jose Valima. Programy napisane w Elixirze uruchamiane są na maszynie wirtualnej Erlanga cechującej się możliwością tworzenia małym kosztem bardzo wielu procesów. Model współbieżności w Elixirze, podobnie jak w Erlangu, bazuje na modelu aktorów. Ze względu na swoją funkcyjną naturę kładzie nacisk na rekurencję oraz funkcje wyższego rzędu, zamiast konstrukcji znanych z paradygmatów imperatywnych, jak np. pętle.

Phoenix framework, który umożliwia stworzenie aplikacji webowej posługując się wzorcem projektowym MVC (ang. Model View Controller). Wykorzystywany jest przez większość firm, które zajmują się produkcją aplikacji webowych. Stanowić będzie on główną bazę dla interfejsu użytkownika systemu.

MongoDB jest to nierelacyjny system zarządzania bazą danych napisany w języku C++. Charakteryzuje się dużą skalowalnością, wydajnością oraz brakiem ściśle zdefiniowanej struktury obsługiwanych baz danych. Zamiast tego dane składowane są jako dokumenty w formacie JSON.

## **Narzędzia, środowisko, biblioteki**

Pełna lista bibliotek, narzędzi, języków programowania użytych podczas implementacji.

### **Stacja Kontrolująca**

- Phoenix
- Phoenix\_pubsub
- Phoenix\_html
- Phoenix\_live\_reload
- gettext
- cowboy
- plug\_cowboy
- amqp
- mongodb
- poolboy

### **Stacja Monitorująca**

- RabbitMQ
- System.Diagnostics

- System
- System.Core
- System.Threading

## Kontrola wersji

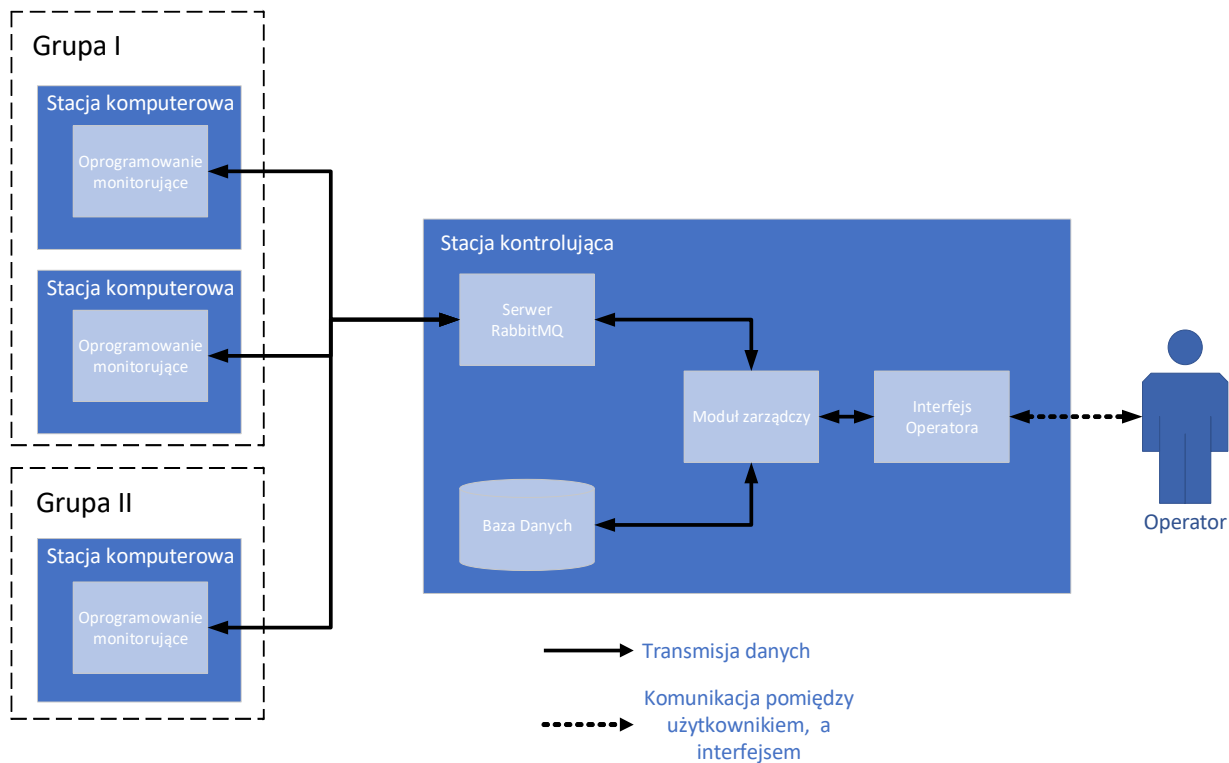
- Git

## Narzędzia

- Visual Studio Enterprise 2017
- Visual Studio Code
- Mongod
- mix
- Przeglądarka Microsoft Edge

## Architektura rozwiązania

Aplikacja została stworzona w celu utworzenia systemu skalowalnego, który umożliwia monitorowanie jak największej skalowalności systemu. Dlatego podzielony został na dwie główne części, które realizują model klient-serwer. Utworzony został następujący projekt aplikacji.

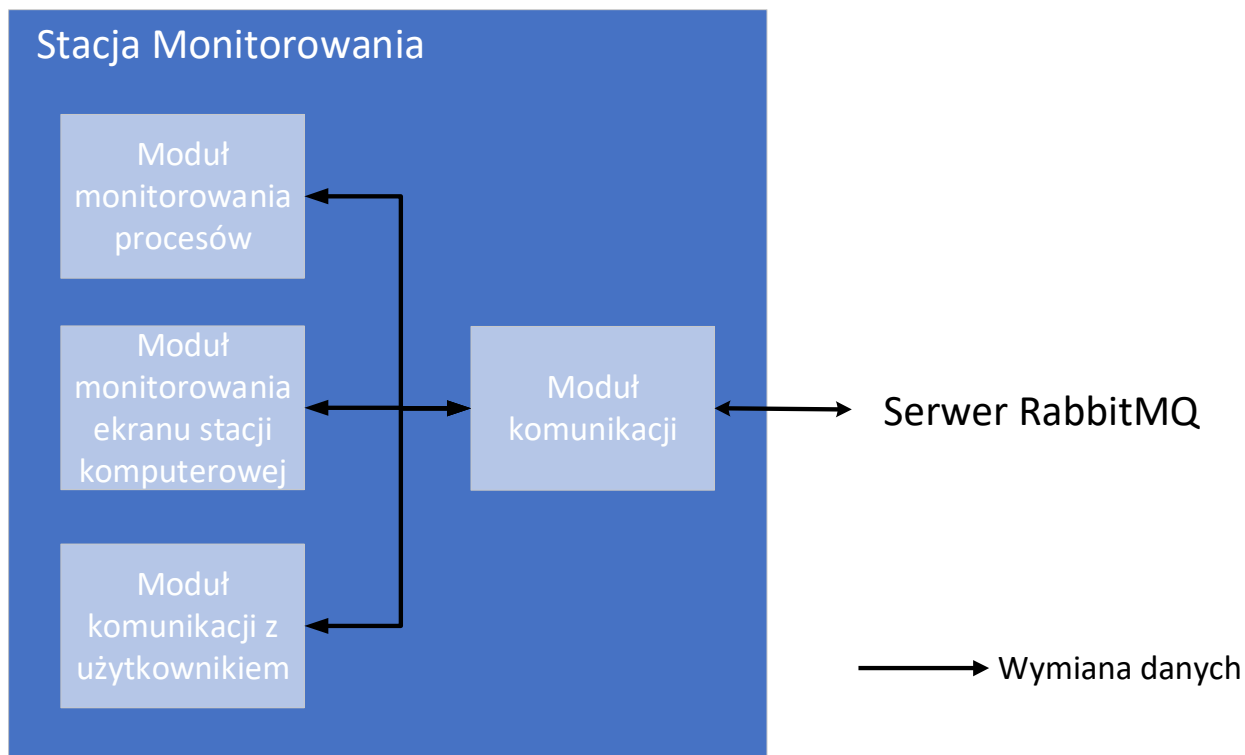


*Rysunek 3: Ogólny schemat architektury*

Na rysunku została przedstawiona wysokopoziomowa architektura systemu. Wyróżnione zostały dwa elementy główne, Stacja Komputerowa, oraz Stacja Kontrolująca.

### Stacja Komputerowa

Stanowi element wprowadzający dane do systemu na podstawie zebranych informacji.



Rysunek 4: Architektura Stacji Monitorowania

- Moduł monitorowania procesów.

Moduł odpowiada za monitorowanie nowo utworzonych procesów. W pierwszej kolejności po uruchomieniu systemu pobierał będzie listę aktualnie uruchomionych procesów za pomocą metody `Process.GetProcesses()`. Utworzona w ten sposób lista pozwoli uniknąć sytuacji w której możliwe jest uruchomienie oprogramowania przed uruchomieniem Oprogramowania Monitorującego. Kolejnym zadaniem wykonywanym przez moduł jest wykrywanie nowo uruchomionego procesu. Odbyna się to poprzez przypisanie funkcji zwrotnej (ang. callback), która będzie wywołana w przypadku wykrycia zmiany w zapytaniu WMI `SELECT * FROM Win32_ProcessStartTrace`.

Dodatkowym elementem wykonywanym przez moduł jest gromadzenie listy procesów, które powinny być blokowane przy uruchomieniu. Podczas dodawania nowej nazwy procesu do listy weryfikowane są wszystkie procesy w celu wyszukania, czy owy proces nie został wcześniej uruchomiony. Weryfikowane są także nazwy nowo uruchomionych procesów w celu blokady w jak najkrótszym czasie.

- Moduł monitorowania ekranu stacji komputerowej

Moduł odpowiada za monitorowanie aktualnie wyświetlanej zawartości ekranu użytkownika. Uruchamiany jest osobny wątek, który w określonym okresie wykonuje zrzuty ekranu, oraz automatycznie wysyła je do Stacji Kontrolującej. Wykonanie zrzutu ekranu odbywa się poprzez wykorzystanie metody `CopyFromScreen`, która jest częścią

obiektu klasy Graphics. Wykonany w ten sposób zrzut ekranu jest zależnie od aktualnej konfiguracji kompresowany w celu zmniejszenia zużycia łącza internetowego.

Możliwość włączenia lub wyłączenia kompresji obrazu wykonywana jest w odpowiedzi na otrzymany komunikat ze Stacji Zarządzania

- Moduł komunikacji z użytkownikiem

Moduł komunikacji z użytkownikiem pełni minimalną rolę w systemie. Ze względu na to, że system jest zarządzany tylko przez operatora obsługującego Stację Kontrolującą. Użytkownik nie ma bezpośredniego wpływu na działanie systemu dlatego nie jest wspomniany jako aktor. Interakcja opiera się na wyświetleniu informacji dla niego, że system aktualnie działa i możliwe że obserwowana jest jego działalność.

## Stacja Kontrolująca

Stanowi najważniejszą część systemu. Jej głównym zadaniem jest zbieranie informacji dotyczących aktualnie wykonywanych operacji w systemie operacyjnym. Kolejnym zadaniem jest zmiana konfiguracji dostępnej dla określonego użytkownika. Wskazana stacja została podzielona na moduły przedstawione na rysunku 1:

- Serwer RabbitMQ

Serwer RabbitMQ stanowi uruchomioną instancję oprogramowania RabbitMQ na Stacji Kontrolującej. Skonfigurowana jest w sposób umożliwiający autoryzację za pomocą certyfikatów, oraz umożliwienie komunikacji grupowej. Serwer stanowi główny moduł komunikacyjny. Umożliwia on odpowiednią skalowalność systemu poprzez utworzenie grupowania (ang. clustering) przez co komunikacja jest równoważona pomiędzy wszystkie dostępne serwery. Ze względu na niewielką liczbę hostów klienta aplikacji funkcjonalność nie jest uruchomiona.

- Baza Danych

Baza danych podzielona jest na dwie części. Pierwsza część stanowi silnik bazodanowy MongoDB, która zawiera informacje o aktualnie aktywnych użytkownikach, oraz procesach które są na nich uruchomione. Ze względu na wykorzystanie danych w formacie JSON informacje mogą być w prosty sposób uaktualniane, oraz przechowywane.

Struktura przechowująca dane dotyczące aktywnych użytkowników.

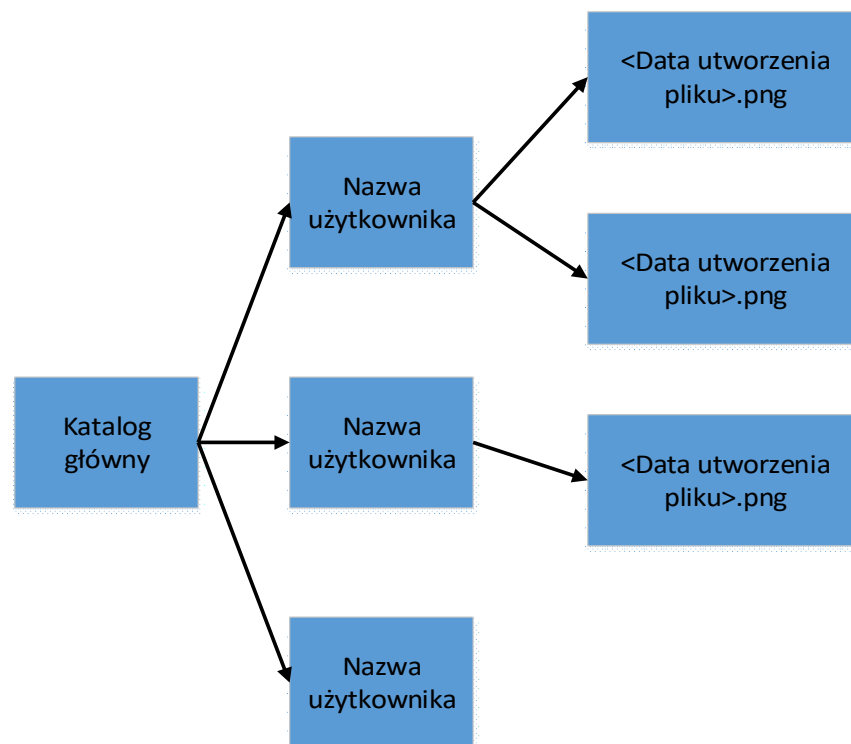
```
{
  „datetime”: <<Data utworzenia listy>>,
  „active”: [<<Lista aktywnych użytkowników>>]
}
```



Struktura przechowująca listę aktywnych użytkowników.

```
{  
  „station”: <<Nazwa użytkownika>>,  
  „processes”: [<<Lista uruchomionych procesów>>]  
}
```

Drugą część bazy stanowi system plików przechowujący wszystkie pobrane zrzuty ekranu w następujący sposób.



Rysunek 5: Struktura katalogów

Na rysunku 3 przedstawiona została struktura katalogów, gdzie umieszczone są pliki zrzutów ekranu wszystkich użytkowników. Wszystkie foldery są umieszczone w katalogu głównym, gdzie znajduje się moduł zarządczy. Dla każdego użytkownika tworzony jest osobny folder, którego nazwę stanowi nazwa użytkownika którego zrzuty ekranu będą gromadzone. W każdym folderze znajdują się pliki w formacie PNG, których nazwa stanowi datę ich utworzenia podaną w sekundach, które minęły od daty 1 stycznia 1970.

Utworzony sposób przechowywania plików znacznie ułatwi późniejsze odwołanie się do najnowszego pliku, który został utworzony, oraz możliwość szybkiego zebrania danych historycznych.

- Moduł zarządczy

Moduł zarządczy jest pośrednikiem pomiędzy Serwerem RabbitMQ a Interfejsem Operatora. Napisany został w języku Elixir, który umożliwia w sposób zrównoleglony wprowadzać informacje dostarczone przez użytkowników. Wszystkie dane zebrane przez moduł są przetwarzane pod względem typu wiadomości, który służy do identyfikacji rodzaju przetwarzanych danych. W systemie wyróżnione są następujące typy danych.

- init

Zawiera informacje dotyczące nowo utworzonego użytkownika. Zbierana jest nazwa użytkownika, która jest gromadzona w liście aktywnych użytkowników.

- update

Zawiera informacje dotyczące aktualnie uruchomionego procesu przez użytkownika.

- screenshot

Zawiera zrzut ekranu, który będzie zapisywany w specjalnej strukturze plików przedstawionej na rysunku 3 „Struktura katalogów”.

- Interfejs Operatora

Interfejs operatora stanowi prosty moduł, który przetwarza dane zapisane w Bazie Danych i przedstawia je użytkownikowi w sposób przyjazny, umożliwiając manipulowanie komputerami działającymi w pracowni komputerowej. Napisany został za pomocą biblioteki „Phoenix Framework”, która umożliwia napisanie strony za pomocą wzorca projektowego Model View Controller (MVC).

## **Interesujące problemy i rozwiązania ich na jakie się natknąłem**

Pierwszym problemem na który można się było natknąć podczas prac była słaba dokumentacja serwera RabbitMQ. Konfiguracja musiała odbywać się poprzez korzystanie z niepewnych źródeł, oraz wykorzystanie metody „prób i błędów”. Ostatecznie rozwiązanie problemu zajęło bardzo dużo czasu przeznaczonego na projekt. Problemy także dotyczyły także konfiguracji wtyczek,

które były wymagane w celu utworzenia wymaganej funkcjonalności do pełnego działania projektu.

Konfiguracja wtyczki <https://www.rabbitmq.com/ssl.html> opisana na oficjalnej stronie nie zawierała wszystkich niezbędnych informacji, które były potrzebne do poprawnego jej skonfigurowania.

Ustawienie wszystkich niezbędnych certyfikatów w początkowych etapach rozwoju systemu stanowiło problem, ze względu na konieczność specjalnej konfiguracji całego łańcucha weryfikacji przy logowaniu. W przypadku wykrycia niezgodności w łańcuchu certyfikatów. Z tego powodu najbardziej problematyczny był język C#, który nie dość że wymagał poprawności weryfikacji łańcucha to nie zezwalał na wykorzystanie własnych wygenerowanych certyfikatów. Przez co wywoływany był wyjątek, który trzeba było ominąć poprzez nadpisanie funkcji weryfikującej aby za każdym razem wskazywała prawidłowość.

Planowana przeze mnie technologia rozszerzająca silnik bazodanowy MongoDB o nazwie GridFS była niemożliwa do wykorzystania. Ze względu na to, że język programowania Elixir stanowi język młody, dopiero rozwijający się aktualna wersja użytkowa sterownika bazodanowego nie wspierała pożądanej technologii. Po analizie dostępnego kodu można było dojść do wniosku, że dana funkcjonalność jest dopiero implementowana. Świadczyło o tym dużo błędów wynikających z prób wykorzystania wspomnianej wtyczki, oraz całkowitego braku dokumentacji opisującej możliwości jej wykorzystania.

Problemem wynikającym ze specyfiki projektu była konieczność implementacji niektórych funkcjonalności w sposób redundantny. Utworzenie oprogramowania w językach C#, oraz Elixir wywołały konieczność utworzenia tej samej funkcjonalności dotyczącej komunikacji z serwerem RabbitMQ. W przypadku poznawania technologii wywołało to dużo problemów dotyczących drobnych różnic pomiędzy implementacjami przez co praca została podwojona.

Ostatnim problemem na którego można było natknąć podczas wykonywania zadań projektowych był wybór nowych technologii nigdy wcześniej nie wykorzystywanych przez twórcę. Wiele problemów wynikło ze specyfikacji języka, którego mechanizmy znacznie różnią się od standardowych (deklaratywnych) programowania. Jednak ze względu na wzrastającą popularność wspomnianego języka warto było spędzić więcej czasu na naukę nowego języka programowania, który znacznie poszerza horyzonty.

## Wnioski

Powyższe problemy nie stanowiły w rzeczywistości prawdziwego zagrożenia do prawidłowego działania systemu. Stanowiły one naturalny element poszerzania swojej wiedzy o nowe technologie, które na samym początku zawsze wywołują wiele problemów które rozwiązywane są poprzez spędzenie większej ilości godzin na przeglądaniu dokumentacji oraz gotowych przykładów i rozwiązań.

## Opis modelu komunikacji

Model komunikacji opiera się na wykorzystaniu mechanizmu kolejek zaimplementowanych w serwerze RabbitMQ. Kolejka może być utworzona przez następujące podmioty biorące udział w komunikacji:

- Nadawca

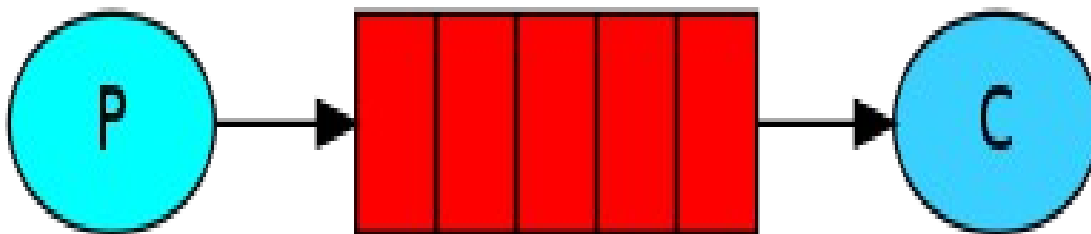
Po utworzonej kolejce możliwe jest wprowadzenie automatycznie danych. Są wtedy one gromadzone na serwerze, przez co nawet w przypadku niedostępności odbiorcy w komunikacji możliwe jest późniejsze odebranie wszystkich informacji.

- Odbiorca

Po utworzonej kolejce możliwe jest automatyczne nasłuchiwanie. W przypadku jeżeli kolejka istniała wcześniej możliwe jest pobranie wszystkich zgromadzonych informacji.

W związku z powyższymi informacjami dotyczącymi kolejkowania zaimplementowane zostały następujące mechanizmy komunikacji.

## Komunikacja bezpośrednia



*Rysunek 6: Komunikacja bezpośrednia*

Wskazany model komunikacji jest wykorzystywany w dwóch przypadkach:

- Wysyłanie informacji ze Stacji Monitorującej do Stacji Kontrolującej.

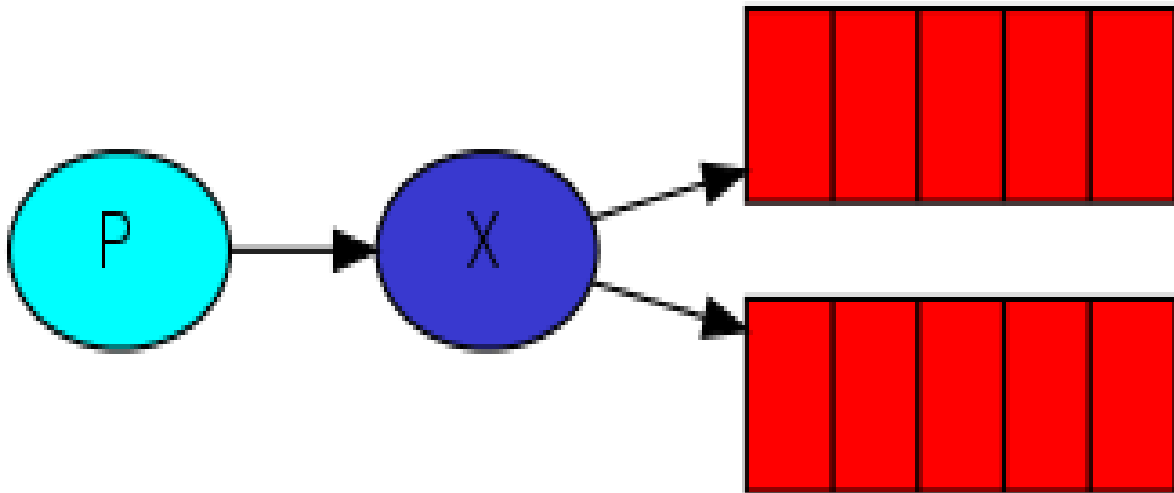
Jest to najczęściej wykorzystywany przypadek komunikacji, ze względu na częste zmiany w systemie operacyjnym. W ten sposób przesyłane są także zrzuty ekranu. Wykorzystany jest fakt, że komunikaty są przesyłane w formie binarnej, przez co nie jest wymagany dodatkowy kanał komunikacji.

- Bezpośrednia zmiana konfiguracji Stacji Kontrolującej

Metoda wykorzystywana w razie konieczności zareagowania przez operatora w trakcie działania systemu w przypadku wykrycia działalności mogącej wpłynąć negatywnie na przebieg przeprowadzanych zajęć lub w przypadku wykrycia korzystania z niedozwolonych na dany moment stron internetowych. Konwencja nazw opiera się na

utworzenia kolejki dla każdej Stacji Monitorującej o nazwie takiej samej jak nazwa użytkownika do którego zostaje przesyłana wiadomość. Możliwe przez to jest dzięki temu tworzenie wyjątków w konfiguracjach użytkowników.

### Komunikacja za pomocą kluczy trasowania



*Rysunek 7: Komunikacja za pomocą kluczy trasowania*

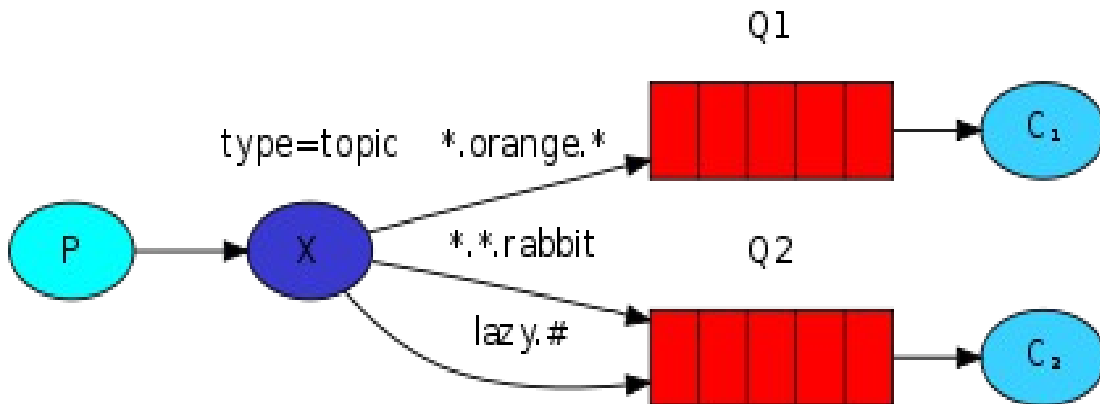
Jest to sposób komunikacji umożliwiający wysyłanie informacji do określonej grupy użytkowników systemu. Opiera się on na utworzeniu tzw. kluczy trasowania (ang. routing keys). Serwer RabbitMQ udostępnia dodatkowy mechanizm trasowania, który umożliwia tworzenie grup użytkowników. Zależnie od przeznaczenia danej grupy możliwe jest ustawienie odpowiedniej konfiguracji. Użytkownik może zapisać się na subskrypcję danego klucza trasowania przez co otrzymywać będzie interesujące go informacje.

Możliwe jest przypisanie więcej niż jednego klucza trasowania do danej Stacji Monitorującej. Przez co możliwe jest utworzenie grup ogólnych jak i specjalistycznych. Przykładowymi grupami mogą być:

- Grupa studentów biorących udział w przedmiocie Podstawy Teleinformatyki
- Grupa studentów piszących kolokwium
- Wszyscy aktualnie obecni na sali.

Taki rozkład umożliwi bardzo szybką możliwość zarządzania informacjami, które są uzyskiwane w trakcie działania systemu.

## Wykorzystanie Topiców



Rysunek 8: Wykorzystanie topiców

Ostatnim modelem komunikacji jest wykorzystanie Topiców. Jest to nazwa własna mechanizmu udostępnionego w serwerze RabbitMQ. Umożliwia on tworzenie podgrup w ramach wymiany informacji. Mechanizm opiera się na omówionym wcześniej mechanizmie za pomocą kluczy trasowania ze specyficznym warunkiem. Nazwa przypisanego Topicu musi zawierać co najmniej dwa słowa oddzielone od siebie znakiem „.”. Umożliwia to zwiększenie liczby wydobywanych informacji za pomocą jednego przypisania. Możliwe są do wykorzystania znaki specjalne:

- Gwiazdka (\*)  
Może być zastąpiona dokładnie jednym słowem.
- Hash (#)  
Może być zastąpiony przez zero lub więcej słów.

W systemie mechanizm jest wykorzystywany w celu wydobycia informacji z uruchomionej wtyczki o nazwie `rabbitmq_event_exchange`, który udostępnia dla subskrybentów informacje dotyczące działania systemu. Wykorzystywanymi informacjami przez system są:

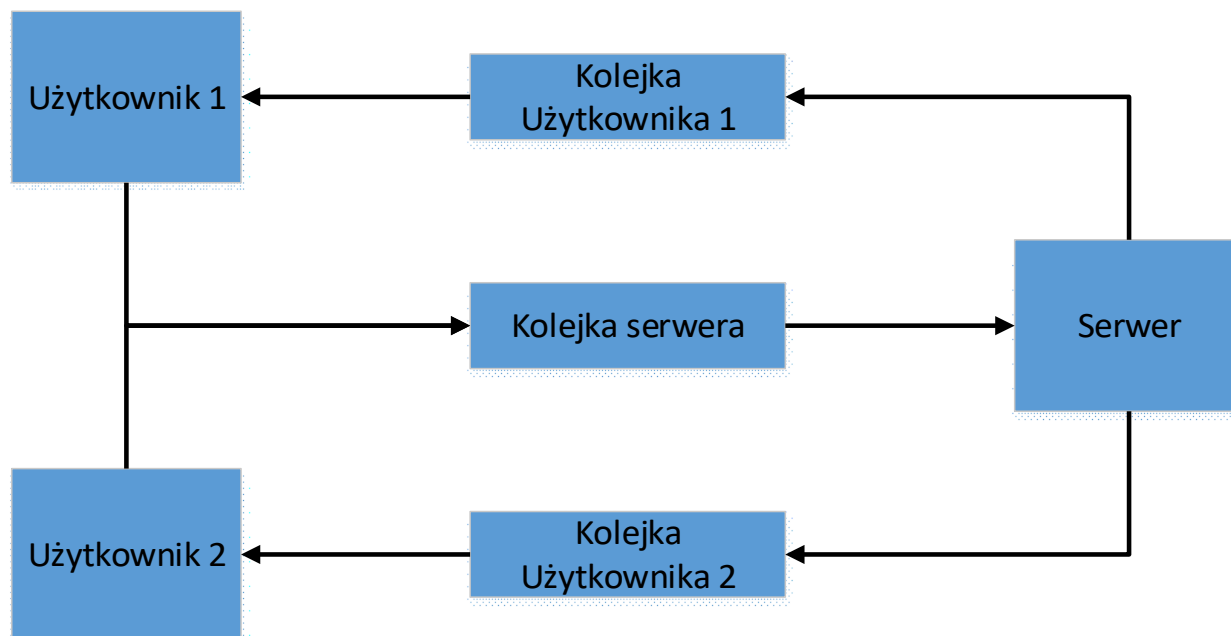
- `connection.created`
- `connection.closed`

Informują one użytkownika o nowych klientach, którzy podłączyli się do monitorowanego serwera. Na ich podstawie można utworzyć listę aktualnie pracujących użytkowników systemu i monitorować ich zachowania w systemie. Ciekawymi wydarzeniami, które można pobierać, jednak nie znalazły zastosowania w systemie są informacje dotyczące wszystkich kolejek,

wymian, oraz przypisać. A także operacje na użytkownikach jak próby uwierzytelnienia, zmiany hasła.

### Ostateczny model komunikacji

Na podstawie powyższych fragmentów opracowany został następujący model komunikacji.



Rysunek 9: Model komunikacji

Przedstawiony został model komunikacji, który zbudowany jest na modelu cyklicznym. W celu zmniejszenia zapotrzebowania na pamięć związaną z utrzymaniem kolejki wyjściowej dla użytkowników zastosowano tylko jedną kolejkę prowadzącą do serwera. Zmniejszenie liczby kolejek wprowadziły konieczność wprowadzenia dodatkowej metody identyfikacji użytkowników. Na szczęście możliwe do wykorzystania są informacje dostępne w metadanych pakietów AMQP (W przypadku próby modyfikacji metadanych, zostanie to wykryte przez serwer i pakiet zostanie automatycznie odrzucony). Informacje wyjściowe są natomiast kierowane bezpośrednio lub za pomocą klucza trasowania.

### Instrukcja użytkowania aplikacji

System składa się z wielu modułów, których instalacja jest niezbędna do jego prawidłowego działania. Ze względu, że cały proces przygotowywania oprogramowania odbywał się na systemie operacyjnym Windows 10, instrukcja zostanie sporządzona właśnie dla tego systemu.

### MongoDB

W pierwszej kolejności należy pobrać serwer baz danych MongoDB ze strony:

<https://www.mongodb.com/download-center/community>

W celu wybrania prawidłowej wersji warto zwrócić uwagę na dopisek w nawiasie zawierający równoważnik zdania „current release”, przez co można uzyskać pewność, że instalowana przez nas wersja oprogramowania jest aktualna.

Po pobraniu należy uruchomić instalator i podążać za jego wskazówkami.

### **WAŻNE!!!**

W trakcie instalacji serwer MongoDB nie tworzy folderu domyślnego, gdzie będą trzymane dane. W tym przypadku istnieje możliwość, że się nie uruchomi. Należy wtedy utworzyć folder C:\data\db.

### **Elixir**

W celu instalacji języka programowania elixir należy wejść na stronę <https://elixir-lang.org/install.html> i pobrać instalator dla systemu operacyjnego Windows. Instalator przeprowadzi przez proces instalacji.

### **RabbitMQ**

W celu instalacji serwera należy pobrać ze storny <https://www.rabbitmq.com/download.html> instalator dla systemu operacyjnego Windows. Instalator przeprowadzi proces instalacji. Po zainstalowaniu serwera należy wprowadzić odpowiednią konfigurację serwera do folderu, który jest umieszczony w folderze %APPDATA%\Roaming\RabbitMQ w pliku rabbitmq.config. Jego zalecana zawartość została przedstawiona poniżej.

```
[
  {rabbit,
    [%%
      {ssl_listeners, [5671]},
      {ssl_options, [{cacertfile, "D:\\rabbitcerts\\ca_certificate_bundle.pem"},
                    {certfile, "D:\\rabbitcerts\\server\\server_certificate.pem"},
                    {keyfile, "D:\\rabbitcerts\\server\\private_key.pem"},
                    {verify, verify_peer},
                    {fail_if_no_peer_cert, false},
                    {ciphers, [
                      "ECDHE-ECDSA-AES256-GCM-SHA384",
                      "ECDHE-RSA-AES256-GCM-SHA384",
                      "ECDHE-ECDSA-AES256-SHA384",
                      "ECDHE-RSA-AES256-SHA384",
                      "ECDH-ECDSA-AES256-GCM-SHA384",
                      "ECDH-RSA-AES256-GCM-SHA384",
                      "ECDH-ECDSA-AES256-SHA384",
```



```
        "ECDH-RSA-AES256-SHA384",  
        "DHE-RSA-AES256-GCM-SHA384",  
        "DHE-DSS-AES256-GCM-SHA384"  
    ]}},  
}  
].
```

Listing 1. Zalecana konfiguracja.

Powyższa konfiguracja umożliwi korzystanie z certyfikatów SSL podczas komunikacji. Żółtym kolorem zostały zaznaczone ścieżki do plików certyfikatów, które będą wykorzystywane do weryfikacji użytkownika.

Generowanie certyfikatów zostało opisane w poradniku na oficjalnej stronie <https://www.rabbitmq.com/ssl.html>

W celu poprawnego działania należy wygenerować główny certyfikat CA. Jeden certyfikat przeznaczony dla serwera, Jeden certyfikat kliencki dla operatora Stacji Kontrolującej, oraz co najmniej jeden certyfikat dla użytkowników korzystających z systemu poprzez Stacje Monitorowania.

Dodawanie użytkowników odbywa się poprzez wywołanie następującej komendy.

```
rabbitmqctl add_user <<nazwa użytkownika>> <<hasło>>  
rabbitmqctl clear_password username  
rabbitmqctl set_permissions -p / „.*” „.*” „.*”
```

Usunięcie hasła użytkownika jest zabiegiem specjalnie przygotowanym ze względu na uwierzytelnienie za pomocą certyfikatów, przez co nie będzie to potrzebne do zachowania odpowiedniego poziomu bezpieczeństwa.

Należy w następnej kolejności uruchomić następujące wtyczki.

```
rabbitmqctl-plugin enable rabbitmq_auth_mechanism_ssl  
rabbitmqctl-plugin enable rabbitmq_event_exchange
```

Uruchomienie powyższych wtyczek uruchamia dwa niezbędne mechanizmy do działania systemu.

- rabbitmq\_auth\_mechanism\_ssl

Umożliwia autoryzację za pomocą certyfikatów.

- rabbitmq\_event\_exchange

Umożliwia notyfikację w związku z połączeniem się nowego klienta.

## Visual Studio

Visual Studio należy pobrać z oficjalnej strony <https://visualstudio.microsoft.com/>. Instalacja powinna odbywać się bez większych problemów.

## Uruchomienie aplikacji

### Stacja Kontrolująca

W celu pełnego uruchomienia Stacji kontrolującej należy wykonać następujące czynności.

- Uruchomienie serwera MongoDB
- Uruchomienie serwera RabbitMQ.
- Uruchomienie Modułu Zarządzania dostępnego w lokalizacji  
\\Management\\controllstation

W tym celu należy wywołać poniższe polecenie.

```
run mix receive.exs
```

- Uruchomienie Interfejsu użytkownika dostępnego w lokalizacji \\Management\\PhoenixFrontend

W tym celu należy wywołać poniższe polecenie.

```
run mix phx.server
```

Po wykonaniu powyższych kroków Stacja Kontrolująca powinna być gotowa do działania.

### Stacja Monitorowania

W celu uruchomienia Stacji Monitorowania zalecane jest uruchomienie jej w środowisku IDE Visual Studio. Projekt jest skonfigurowany, aby bezproblemowo umożliwiać jego uruchomienie.

W celu poprawnego działania konieczne jest wprowadzenie informacji niezbędnych do uwierzytelnienia do pliku app.config. Poniżej przedstawiono przykładową konfigurację.

```
<appSettings>
  <add key="QueueName" value="dataqueue"/>
  <add key="CertPath" value="D:\\rabbitcerts\\client\\client_certificate.p12"/>
  <add key="CertPassword" value="zaq1@WSX"/>
  <add key="UserName" value="client1"/>
</appSettings>
```

Kluczowe są wartości parametrów zaznaczone kolorem żółtym. Powyższe wartości mają następujące znaczenie:

- QueueName

Nazwa kolejki głównej, która stanowi drogę wejściową do Stacji Kontrolującej.

- CertPath

Ścieżka do certyfikatu klienta w formacie p12.

- CertPassword

Hasło służące do poprawnego otworzenia certyfikatu.

- UserName

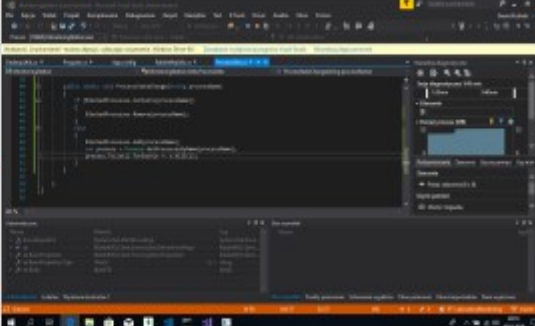
Nazwa użytkownika, która jest zawarta w certyfikacie. (W przypadku wprowadzenia innej nazwy użytkownika serwer nie będzie przyjmował wprowadzanych wiadomości).

## Wyświetlane widoki

W związku z tym, że Stacja monitorowania nie jest zależna od użytkownika nie został zaimplementowany żaden widok. Z założenia stacja miała działać w tle z jak najmniejszym wpływem na działanie systemu operacyjnego, aby nie przeszkadzać w efektywnej pracy.

Stacja Kontrolująca posiada jeden widok, który zawiera wszystkie informacje niezbędne dla operatora stacji.

## Aktywni użytkownicy

Nazwa użytkownika	Uruchomione procesy	Stan ekranu
client1	cmd notepad backgroundTaskHost erl epmd	

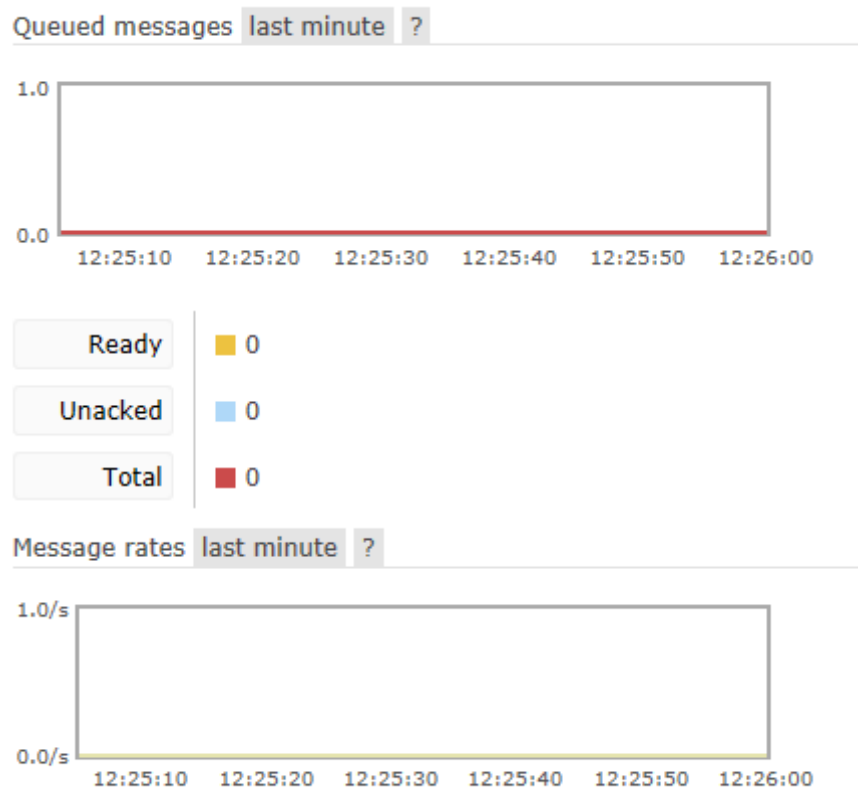
*Rysunek 10: Fragment widoku operatora*

Na Rysunku 10 został przedstawiony widok zawierający następujące informacje:

- Nazwa użytkownika
- Lista uruchomionych procesów
- Aktualny Stan Ekranu

Wszystkie te informacje są zawarte w formie tabeli, przez co w jednym widoku możliwe jest podejrzenie aktywności wszystkich użytkowników systemu.

Dodatkowym widokiem, który może być wykorzystanym jest udostępniony interfejs typu Web-Management dostępnym domyślnie dla każdej instancji serwera RabbitMQ.



*Rysunek 11: Statystyki RabbitMQ*

Przedstawione na Rysunku 11 statystyki nie mają bezpośredniego wpływu na działanie systemu. Przedstawiają one następujące informacje.

- Liczba wiadomości, które są zakolejkowane.
- Liczba wiadomości przesyłanych na sekundę.

Z powyższych statystyk najbardziej przydatna może okazać się liczba wiadomości przesyłanych na sekundę. Pomoże to w ustaleniu, czy dany serwer może zostać przeciążony. Według oficjalnej dokumentacji systemu serwer może obsłużyć maksymalnie 20 tys zapytań na

sekundę. W przypadku zbliżania się do przeciążenia serwera możliwe jest uruchomienie kolejnego serwera w roli load balancera.

## **Możliwości rozwoju**

System posiada wiele możliwości rozwoju. Przedstawione zostały tylko najciekawsze, które będą na pewno wykorzystane w przyszłości.

### **Przetworzenie serwera do obsługi gier sieciowych czasu rzeczywistego**

Implementacja serwera za pomocą języka programowania Elixir umożliwia utworzenie zrównoleglonego systemu, który może bardzo szybko obsługiwać przychodzące wiadomości. Przykładem gry, której serwery są napisane w sposób działający na tym samym środowisku uruchomieniowym jest League of Legends.

### **Rozszerzenie funkcjonalności Stacji Monitorowania**

Stacja Monitorownia ma bardzo wiele możliwości rozwoju jej funkcjonalności. Pierwszym pomysłem jest rozszerzenie działalności to stworzenia oprogramowania antywirusowego, który może być centralnie zarządzanym przez operatora. System taki może wykrywać dodatkowe informacje przez wprowadzenie dodatkowych mechanizmów

### **Praca w przestrzeni jądra**

Możliwe jest rozszerzenie działalności o wprowadzenie sterownika zbierającego dane w przestrzeni jądra. Uzyskane w ten sposób informacje będą bardziej dokładne i istnieje mniejsza szansa, że zostaną one sfalszowane przez złośliwe oprogramowanie.

Rozszerzenie może odbywać się poprzez zbieranie informacji z pamięci RAM, do której dostęp można uzyskać wtedy w sposób bezpośredni. Możliwe jest monitorowanie wszystkich struktur np. EPROCESS.

Możliwe jest także wykorzystanie funkcji zwrotnych udostępnionych przez system operacyjny. Dodatkowo wydobywane będą następujące informacje:

- Informacja o uruchomieniu procesu
- Informacja o załadowaniu nowej biblioteki
- Informacja o utworzeniu klucza rejestru
- Informacja o utworzeniu nowego wątku
- Informacja o załadowaniu nowego sterownika

### **Wykrywanie połączenia nowych urządzeń**

Możliwe jest rozszerzenie funkcjonalności o wprowadzenie możliwości wykrywania połączenia dodatkowych urządzeń służących do przenoszenia danych. Wydobywanie może odbywać się

poprzez monitorowanie kluczy rejestrów odpowiedzialnych za przechowywanie informacji o podłączonych urządzeniach pamięci masowej.

### Wywoływanie zdalnych procedur

Możliwe jest wprowadzenie dodatkowej funkcjonalności polegającej na wywoływaniu zdalnych procedur. Umożliwi to administratorowi sprawdzić informacje na komputerze w przypadku wątpliwości co do działania systemu operacyjnego.

## Podsumowanie

Prace związane z utworzeniem rozwiązania projektu w ramach zajęć Podstawy Teleinformatyki zajęły bardzo dużo czasu ze względu na wykorzystanie nowych technologii.

Wykorzystany język oprogramowania jakim był Elixir przysporzył wiele problemów, które były związane z nikłą znajomością języka przed przystąpieniem do prac projektowych. Wynikło wiele problemów, które wymógł na mnie przypomnienie sobie terminologii związanej z językami funkcyjnymi, które były omawiane podczas przedmiotu Języki i Paradygmaty Programowania 2. Jednak mimo problemów zdobyta wiedza będzie bardzo użyteczna w przyszłej karierze. Umożliwi to rozszerzenie swoich horyzontów uniemożliwiające przywiązanie się tylko do jednego języka programowania. Co wpłynie na o wiele większą jakość wytwarzanych projektów. Podwyższenie jakości będzie polegało na rzeczywistym dopasowaniu języka programowania do problemu programistycznego, którego trzeba rozwiązać, a nie odwrotnie jak zwykle bywa.

Prace podzielone były na dwa etapy. Pierwszy etap polegał na utworzeniu architektury systemu, który będzie mógł być przystosowany do aktualnego zadania, oraz umożliwiać będzie jego rozszerzenie. Konieczne było sprawdzenie możliwości gotowych bibliotek i sprawdzenie najbardziej przystosowanej do zadania określonego w ramach projektu. W wyniku pierwszego etapu powstała architektura systemu.

Kolejny etap polegał na implementacji wszystkich zaplanowanych rozwiązań. Proporcje czasu pomiędzy wyszukiwaniem informacji dotyczących oprogramowania, oraz analiza dokumentacji do czasu implementacji wyniosły 3:1. Dlatego należy zwrócić uwagę, że mimo obecnej ilości kodu w projekt została włożona znacznie większa ilość pracy.

Wszystkie założone elementy, które powinny być wykonane zostały zrobione w sposób spełniający wymagania funkcjonalne przedstawione na pierwszych zajęciach.

## Źródła wiedzy

- <https://www.rabbitmq.com/ssl.html>
- <https://www.rabbitmq.com/getstarted.html>
- <https://www.mongodb.com/>

- <https://github.com/ankhers/mongodb>
- <https://elixir-lang.org/getting-started/introduction.html>
- <https://phoenixframework.org/>
- [https://hexdocs.pm/phoenix/up\\_and\\_running.html](https://hexdocs.pm/phoenix/up_and_running.html)