POLITECHNIKA POZNAŃSKA
WYDZIAŁ ELEKTRYCZNY

PROJEKT ZESPOŁOWY

SAURON

Autorzy:
Patryk Mroczyński *- patryk.mroczynski@student.put.poznan.pl - 126810*
Oskar Rutkowski *- oskar.rutkowski@student.put.poznan.pl - 126845*
Jakub Wiśniewski *- jakub.t.wisniewski@student.put.poznan.pl - 126824*
Daniel Staśczak *- daniel.stasczak@student.put.poznan.pl - 126816*

BEZPIECZEŃSTWO SYSTEMÓW INFORMATYCZNYCH
GRUPA BSI 2 L2

POZNAŃ, 2018

Spis treści:

1. Charakterystyka ogólna

Projekt zespołowy *Sauron* powstał na potrzeby projektu studenckiego w ramach zajęć Podstaw Teleinformatyki na Politechnice Poznańskiej. Nawiązuje do tematu związanego z monitorowaniem urządzeń w sali laboratoryjnej w postaci rozbudowanego systemu.

Temat ten został wybrany ze względu na liczne zastosowania oraz jako pomoc dla prowadzących zajęcia laboratoryjne. Głównym zamierzeniem projektu jest monitorowanie uruchomionych procesów na urządzeniach laboratoryjnych wraz z przesyłaniem zrzutów ekranu na monitorach urządzeń laboratoryjnych. Ma to na celu eliminowanie nieprawidłowości związanych z oszukiwaniem w czasie trwania zaliczeń bądź egzaminów na urządzeniach laboratoryjnych. Zagadnienia związane z tym projektem wydają się ciekawe pod względem koncepcyjnym jak i technicznym. Koncept projektu został starannie opracowany przez autorów dla zorganizowanej pracy oraz zniwelowania przewidywalnych błędów. Tematyka projektu jest także nietrywialnym zagadnieniem programistycznym ze względu na swoje przeznaczenie. Ważnym jest, że system monitorowania jest przeznaczony na wiele platform, co wywiera na autorach znajomości programowania aplikacji webowych jak i budowanie serwera przetwarzającego niezbędne dane.

System monitorowania sali składa się z trzech elementów jakimi są aplikacja kliencka studenta, aplikacja kliencka prowadzącego oraz aplikacja serwerowa.

Głównym założeniem aplikacji klienckiej studenta jest zbieranie informacji na temat wszystkich uruchomionych procesów w danym okresie czasowym. Ma za zadanie również, w równych odstępach czasu, przechwytywać zrzuty ekranów, na których użytkownik aktualnie pracuje. Informacje o tym, jakie procesy są zakazane w czasie trwania sesji, przechowują czarne listy. Składowane są one w bazie danych serwera i wysyłane są do aplikacji klienckich studenta zaraz po lokalnym uruchomieniu oprogramowania. W przypadku wystąpienia niezgodności procesu z pozycjami na czarnej liście, wysyłana zostaje informacja o zaistniałym procederze. Informację tą przetwarza aplikacja serwerowa, po czym aplikacja kliencka prowadzącego zostaje o niej poinformowana.

Następnym elementem składowym systemu jest aplikacja serwerowa. Ma za zadanie przetwarzanie informacji związanych z obsługą i zarządzaniem całym systemem. Przyjmuje dane z aplikacji klienckiej studenta oraz przechowuje je w lokalnej bazie danych. Do przechowywanych informacji ma dostęp aplikacja kliencka prowadzącego, która żąda aktualnych danych dla określonego okresu czasu. Aplikacja serwerowa ma również za zadanie uwierzytelniać użytkowników aplikacji prowadzącego.

Ostatnim elementem systemu jest aplikacja kliencka prowadzącego. Pierwszym a zarazem najważniejszym zadaniem tego modułu jest umożliwienie podglądu do danych składowanych w wyżej opisanej aplikacji serwerowej. Aplikacja prowadzącego działa jako aplikacja webowa. Pobiera i przetwarza dane z serwera w celu zwizualizowania informacji o procesach

oraz zrzutach ekranu dla całej grupy laboratoryjnej aktualnie podłączonej do wybranej wcześniej sali laboratoryjnej. Aplikacja zapewnia również możliwość dodawania, modyfikacji, edytowania oraz wybierania czarnej listy procesów, które są niedozwolone w czasie trwania sesji.

Szczegółowy opis architektury działania całego systemu zostanie przedstawiony w kolejnych rozdziałach.

2. Podział prac

Ze względu na złożoność systemu monitorowania urządzeń w sali laboratoryjnej, zespół został podzielony na trzy działy tematyczne. Pierwszy z nich nawiązuje do aplikacji serwerowej i jest wykonywany przez Patryka Mroczyńskiego. Jest związany z zagadnieniami obsługi serwera oraz bazy danych przetrzymującej niezbędne informacje. Drugim działem zajmują się Jakub Wiśniewski oraz Oskar Rutkowski i jest związany z przetwarzaniem i wizualizacją danych po stronie aplikacji klienckiej prowadzącego. Ostatnim działem jest aplikacja kliencka studenta. Reprezentuje go Daniel Staśczak. Dział ten zajmuje się wszelkimi sprawami związanymi z pozyskiwaniem informacji na temat procesów oraz zrzutów ekranów. Wykonane zadania wraz z przydzielonymi wykonawcami zostały przedstawione w poniższej tabeli.

Osoba	**Zadania**
Patryk Mroczyński	Aplikacja serwerowa - usługi REST z obsługą bazy danych i autoryzacją
	Aplikacja serwerowa - serwis obsługujący blacklisty - dodawanie, usuwanie oraz modyfikacja
	Aplikacja serwerowa - serwis obsługujący użytkowników - dodawanie, usuwanie oraz modyfikacja
	Aplikacja serwerowa - serwis obsługujący procesy - dodawanie oraz pobieranie listy procesów
	Aplikacja serwerowa - serwis umożliwiający wysyłanie oraz pobieranie zrzutów ekranów utworzonych przez aplikację kliencką.
	Zaimplementowanie modułów ułatwiających pracę - pliku zawierającego konfigurację oraz metodę autoryzującą użytkowników oraz context managera ułatwiającego korzystanie z bazy danych

	Projekt i implementacja bazy danych
Jakub Wiśniewski	Dobranie narzędzi, bibliotek i architektury aplikacji klienckiej prowadzącego
	Implementacja uwierzytelniania w aplikacji w aplikacji prowadzącego
	Konfiguracja serwera VPS z serwerem HTTP - nginx
	Implementacja pobierania listy studentów dla podanej grupy wraz z aktualnymi procesami oraz obsługa zrzutów ekranów w aplikacji prowadzącego
	Opracowanie efektów wizualnych na stronie aplikacji prowadzącego (informacje o błędach, ładowanie elementów, wysuwany boczny panel szczegółów)
Oskar Rutkowski	Przygotowanie prototypu graficznego aplikacji prowadzącego
	Utworzenie szkieletu aplikacji prowadzącego
	Przygotowanie zapytań HTTP do serwera w aplikacji prowadzącego
	Opracowanie efektów wizualnych na stronie aplikacji prowadzącego (pasek menu, logo, strona główna po zalogowaniu)
	Implementacja obsługi czarnych list (edycja, usuwanie, dodawanie) w aplikacji prowadzącego
	Implementacja obsługi wyboru sali w aplikacji prowadzącego
Daniel Staśczak	Aplikacja kliencka studenta - umożliwienie pełnej konfiguracji niezbędnych aspektów działania aplikacji, takich jak adres serwera, URI zasobów serwisów REST zapewnianych przez ów serwer, dane identyfikujące użytkownika (w tym kredencjały) oraz częstotliwość komunikacji z serwerem.
	Aplikacja kliencka studenta - zapewnienie rozbudowanego systemu rejestrowania aktywności aplikacji w postaci konfigurowalnych

	logów.
	Aplikacja kliencka studenta - funkcjonalność cyklicznego rejestrowania listy aktywnych procesów w systemie.
	Aplikacja kliencka studenta - funkcjonalność cyklicznego wykonywania zrzutów wszystkich podłączonych ekranów.
	Aplikacja kliencka studenta - funkcjonalność cyklicznego wysyłania zarejestrowanych danych do serwera poprzez udostępniony przez niego interfejs REST API.

Tabela 1. Rozkład prac zespołu programistycznego.

3. Opis wymagań funkcjonalnych

W tym rozdziale zostaną zaprezentowane wymagania funkcjonalne systemu monitorowania urządzeń w sali laboratoryjnej. Wymagania funkcjonalne są podzielone na trzy dziedziny: aplikacja kliencka prowadzącego, aplikacja kliencka studenta oraz aplikacja serwerowa. Poniżej zostaną wyszczególnione wymagania dla każdej z dziedzin.

1. Aplikacja kliencka prowadzącego
    - logowanie
    - wylogowanie
    - wybór grupy w celu nasłuchiwania procesów
    - wybór listy procesów zakazanych
    - podgląd procesów i screenshotów wybranej osoby
    - dodawanie, edycja i usuwanie list procesów zakazanych
    - możliwość użycia szybkiego powrotu do ekranu grupy z studentami

2. Aplikacja kliencka studenta
    - interfejs do konfiguracji adresu serwera docelowego
    - interfejs do konfiguracji danych identyfikujących aplikację
    - interfejs do konfiguracji kredencjałów
    - cykliczne pobieranie listy aktywnych procesów
    - cykliczne wykonywanie zrzutów ekranu
    - cykliczne wysyłanie zebranych informacji do serwera
    - obsługa czarnych list

3. Aplikacja serwerowa
    - możliwość zmiany adresu ip oraz portu, na którym działać ma serwer
    - możliwość filtrowania listy procesów za pomocą parametrów określających przedział czasowy, konkretną grupę oraz nazwę

nazgula
- możliwość filtrowania listy informacji o zrzutach ekranu za pomocą parametrów określających przedział czasowy, konkretną grupę, nazwę nazgula oraz określenie czy mają być pobierane tylko najnowsze informacje dla każdego nazgula osobno
- możliwość filtrowania listy informacji o nazgulach za pomocą parametrów określających przedział czasowy nadawania oraz konkretną grupę
- możliwość filtrowania listy zawierającej informacje o niedozwolonych procesach.
- możliwość aktywowania lub dezaktywowania danej listy procesów niedozwolonych

4. Opis wymagań niefunkcjonalnych

W tym rozdziale zostaną zaprezentowane wymagania niefunkcjonalne systemu monitorowania urządzeń w sali laboratoryjnej. Wymagania niefunkcjonalne są podzielone na trzy dziedziny: aplikacja kliencka prowadzącego, aplikacja kliencka studenta oraz aplikacja serwerowa. Poniżej zostaną wyszczególnione wymagania dla każdej z dziedzin.

1. Aplikacja kliencka prowadzącego
    - aplikacja wieloplatformowa obsługująca WEB API
    - wykorzystanie metody uwierzytelnienia Basic Auth
    - wykorzystany język programowania JavaScript
    - użyty framework JavaScript - Vue.js
    - aplikacja w języku polskim
    - podświetlanie użytkownika oszukującego

2. Aplikacja kliencka studenta
    - odporność na błędy połączenia
    - odporność na próby oszukania aplikacji
    - cross-platformowość

3. Aplikacja serwerowa
    - wykorzystany język programowania Python3.6
    - użyty framework dla Pythona do aplikacji webowych - cherrypy
    - użyty moduł Pythona, dla ułatwienia integracji z bazą danych - pymongo
    - zwracanie informacji uzyskanych przez metody GET w formacie JSON

4. Serwer
    - przechowywanie screenshotów na serwerze VPS
    - serwer HTTP - nginx obsługujący zapytanie o stronę z aplikacją

prowadzącego oraz endpointami do API
- baza danych MongoDB

5. Wybrane technologie

- **Aplikacja kliencka prowadzącego**

W celu stworzenia wieloplatformowej aplikacji, która pozwoli prowadzącym na podgląd procesów oraz zrzutów ekranów studentów została podjęta decyzja o stworzeniu aplikacji webowej. Ze względu na architekturę REST aplikacji serwera oraz możliwości aplikacji SPA (Single Page Application) wybór dotyczył frameworków oraz bibliotek dla języka JavaScript.

Aplikacje Single Page Application pozwalają na ograniczenie ruchu sieciowego oraz zapewnienia większego komfortu użytkowania takiej aplikacji przez użytkownika dzięki swojej strukturze, która symuluje przenoszenie się do podstron poprzez nadpisywanie obecnej zawartości strony zamiast wysyłania kolejnego zapytania do serwera. Dzięki temu wchodząc na stronę użytkownik pobiera całą aplikację na swój komputer, a następnie kod JavaScript renderuje kod HTML do wyświetlania. Pomimo wystąpienia problemów z siecią czy serwerem po uprzednim załadowaniu aplikacji użytkownik może kontynuować pracę w takiej aplikacji o ile działania, które podejmuje nie opierają się na komunikacji sieciowej - w takim przypadku twórca może zaprojektować jak taki błąd zostanie obsłużony i przedstawiony użytkownikowi.

Najpopularniejszymi frameworkami i bibliotekami, które wspomagają proces tworzenia aplikacji SPA w obecnym momencie są React, Angular oraz Vue. Porównanie wyżej wymienionych narzędzi w Google Trends jest trudnym zadaniem ze względu na niejednoznaczność nazewnictwa frameworka od właśnie firmy Google. Konkretnym porównaniem może być jednak przyrównanie zainteresowania tymi projektami na GitHubie. W tym zestawieniu najlepiej wypada framework Vue, który na obecną chwilę ma 105,139 gwiazdek, a tuż za nim React z liczbą 104,647 gwiazdek. Najsłabiej wypada framework Angular - 37,710 gwiazdek.

Nazwy AngularJS, Angular, Angular 2, Angular 4, Angular 5 i w końcu Angular 6 mogą wprowadzić niemałe zamieszanie ze względu na to, że Angular to w dużym skrócie przebudowa i ulepszenie frameworka AngularJS, który był pierwszy. Kolejne cyfry przy nazwach oznaczają kolejne duże zmiany tego frameworka i ze względu taki bieg rzeczy szukając informacji o Angularze bardzo prawdopodobne, że będzie to ciężkie zadanie ze względu na to wprowadzające w zakłopotanie nazewnictwo. Niestety sama dokumentacja jest również niezbyt dokładna i ciężko wyszukać z niej potrzebne informacje.

React jest projektem należącym do Facebooka i w porównaniu do Angulara czy Vue nie jest kompletnym frameworkiem lecz biblioteką do renderowania samych widoków. Skupia się właśnie na tej jednej rzeczy i dlatego jest tak popularna wśród deweloperów. Pozwala na

dobranie innych narzędzi według własnego uznania jednak ma duży próg wejścia i wymaga doświadczenia w tworzeniu aplikacji webowych a także składni standardu EcmaScript 2015. Oprócz tego używa także składni JSX do tworzenia elementów HTML. Niewątpliwą zaletą jest także ogromna społeczność oraz wsparcie i zaangażowanie w projekt dużych firm takich jak Netflix, Dropbox, New York Times czy Amazon.

Mimo tego, że Vue to najmłodszy projekt między wyżej wymienionymi oraz nie stoi za nim żadna duża firma tylko programista Evan You to zaskarbił sobie spore grono fanów. Samo jądro tego projektu skupia się tak jak React też na warstwie widoku, lecz udostępnia również dedykowane narzędzia, które dostarczają potrzebne funkcjonalności, które w przypadku Reacta są raczej tworzone przez społeczność. Dzięki narzędziu vue-cli w bardzo prosty sposób jest możliwe utworzenie projektu dostosowanego do potrzeb. Vue skupia się na tym by być proste, szybkie i eleganckie, a przez wzgląd na świeżość tego projektu nie popełnia błędów starszych frameworków. Tak jak React implementuje wirtualny DOM (Document Object Model) co daje wydajne renderowanie zmian w kodzie HTML. Poza tym framework Vue jest lżejszy niż React czy Angular co również wpływa na wagę aplikacji, która jest przy jego użyciu tworzona. Dokumentacja jest bardzo przejrzysta i wyczerpująca, a ze względu na strukturę tworzenia aplikacji zachowując osobno HTML, CSS i skrypty JavaScript nie potrzeba uczyć się niczego nowego. Ze względu na prostotę architektury tego frameworka nie jest trudny do zrozumienia i zaczęcia z nim pracy.

Jedną z bibliotek, która dostarczana jest przez Vue jest Vuex. Jest to implementacja architektury Flux przechowywania stanu aplikacji więc chcąc wykorzystać zalety takiego rozwiązania jakim jest przechowywanie stanu aplikacji w jednym miejscu, a także funkcje, które odpowiadają za zarządzanie tym stanem oczywistym było wybranie biblioteki Vuex, która jest dedykowanym rozwiązaniem dla tego frameworku. Podobnie sytuacja ma się do biblioteki vue-router, która zapewnia symulację przechodzenia do podstron.

Z racji architektury całego systemu pewnym jest ciągłe wysyłanie zapytań do serwera. Natywnym rozwiązaniem w języku JavaScript jest Fetch API, które dostarcza interfejs na te potrzeby, lecz wygodniejszy interfejs oraz stabilność zapewnia biblioteka axios. Oprócz tego Fetch API nie jest jeszcze w pełni wspierane przez wszystkie przeglądarki.

W celu wydajnej pracy z frameworkiem Vue wybrane zostało także narzędzie vue-cli, które udostępnia interfejs konsolowy do tworzenia projektu oraz zarządzania i edycji jego elementów.

- **Aplikacja serwerowa**

Aplikacja serwerowa wykorzystuje język Python, który jest językiem programowania wysokiego poziomu ogólnego przeznaczenia. Język ten został stworzony przez programistę Guido van Rossum w

roku 1991. Wykorzystywaną implementacją Pythona jest CPython, który jest standardową implementacją napisaną w języku programowania C. Wybraną wersją języka Python jest Python3.6. Został on wybrany z dwóch powodów. Pierwszym z nich jest wycofywane wsparcie dla rodziny wersji Python 2.x, dla którego wsparcie zostanie wycofane w roku 2020. Różnice między wersjami 2 oraz 3 to między innymi zastąpienie wyrażenia print funkcją, która posiada identyczną nazwę oraz wprowadzenie unicode stringów. Poza tymi zmianami należy wspomnieć o dużym wsparciu dla języka Python3 w postaci wielu dostępnych bibliotek, modułów czy poradników, co świadczy o dużym zaangażowaniu społeczności zebranej wokół języka. Drugim powodem, dla którego została wybrana wersja 3.6 języka programowania Python jest fakt, że w dniu rozpoczęcia prac implementacyjnych nad projektem była to najnowsza dostępna wersja stabilna języka. Wersja ta cechuje się kilkoma dodatkowymi elementami, które zostały wprowadzone do języka. Jedną z najistotniejszych zmian jest zmiana implementacji słowników, która oferuje 20-25% zmniejszenie zużycia pamięciowego w porównaniu do wersji 3.5 języka Python.

Wybranym frameworkiem webowym jest framework cherrypy, który znacznie ułatwia tworzenie aplikacji internetowych. Udostępnia on proste narzędzia umożliwiające tworzenie serwisów wraz z ich metodami HTTP. Jedną z najistotniejszych elementów tego frameworka są dekoratory, które w prosty sposób obudowują metody, rozbudowując ich funkcjonalność. Przykładem użyteczności takich dekoratorów mogą być cherrypy.tools.json_out(), który zapewnia, że zwrócona struktura danych zostanie przekonwertowana (o ile to możliwe) do ciągu znaków w formacie JSON, który jest łatwy do przetworzenia i czytelny dla człowieka. Jeśli nie uda się przenieść struktury obiektu do formy tekstowej, zostanie rzucony wyjątek, który opisuje dlaczego się to nie udało. Framework ten umożliwia bardzo prostą konfigurację. Słownik, który zawiera klucze o odpowiednich nazwach może zostać przekazany do odpowiedniej funkcji. Konfiguracja taka może być globalna lub dotyczyć tylko konkretnego punktu końcowego. Można w niej określić adres IP oraz port, na którym ma zostać wystawiona usługa, jaka funkcja ma zostać użyta do autoryzacji użytkownika. Jest również możliwość określenia w jakim miejscu mają się zapisywać logi, które mogą mówić zarówno o błędach i wyjątkach, które się pojawiły oraz o zapytaniach, które zostały wysłane do serwera (wraz z określeniem jaki użytkownik dokonał danego zapytania.

System zarządzania bazą danych, który jest uruchomiony na serwerze to MongoDB. Jest to system napisany w języku C++, dzięki czemu cechować się on może wysoką wydajnością oraz skalowalnością. Przechowywane w bazie danych dokumenty to pliki w formacie JSON, które nie posiadają ściśle zdefiniowanej struktury przechowywanych danych. Baza ta cieszy się dużym uznaniem w środowisku technologicznym. Doskonałym potwierdzeniem tego jest

fakt, że Wielki Zderzacz Hadronów korzysta z tejże bazy danych (informacja na podstawie strony https://pl.wikipedia.org/wiki/MongoDB). Dzięki dużej popularności tego systemu zarządzania bazą danych, powstało wiele bibliotek ułatwiających obsługę bazy danych z poziomu programu komputerowego do popularnych języków programowania takich jak C++ czy Python.

Biblioteką ułatwiającą korzystanie z bazy danych w Pythonie, która została wykorzystana w projekcie, jest moduł Pymongo. Umożliwia on wykonywanie w bardzo prosty sposób operacji na bazie. Jego możliwości to m.in. utworzenie nowej bazy danych, utworzenie nowej kolekcji, utworzenie nowego dokumentu, modyfikowanie pojedynczego lub grupy dokumentów czy usunięcie pojedynczego lub grupy dokumentów. Dostęp do konkretnej bazy danych może odbywać się za pomocą dostępu opartym na atrybucie (np. client.db) lub używając stylu słownikowego (np. client[‘db’]). W identyczny sposób odbywa się dostęp do konkretnej kolekcji.

Systemem operacyjnym, na którym zostały przeprowadzone testy działania aplikacji jest system Ubuntu Server 18.04. Jest to najnowsza stabilna wersja systemu Ubuntu, która została wypuszczona w kwietniu roku 2018. System ten jest dystrybucją Linuxa, nad którą pracuje firma Canonical. Środowisko to zostało wybrane ze względu na stabilność oraz obecność w repozytoriach Ubuntu wersji narzędzi i programów, które są pożądane w procesie programistycznym. Programami i językami programowania, które znajdowały się w repozytoriach są m.in. pożądana wersja MongoDB, Python w wersji 3.6 oraz pip3 dla Pythona. Pip3 jest programem, który ułatwia instalację oraz zarządzanie modułami w języku Python.

- **Aplikacja kliencka studenta**

  Aplikacja kliencka studenta wykorzystuje, podobnie jak aplikacja serwerowa, język Python w wersji 3.x. Implementacja kompatybilna jest z wersją od 3.5.2 wzwyż, dostępną domyślnie w systemie Ubuntu 16.04 LTS, na którym w większości ów implementacja była wykonywana.

  Ze względu na obszerność standardowej biblioteki tego języka, wiele funkcjonalności można było zrealizować bez wykorzystywania zewnętrznych modułów. Należą do nich między innymi: obsługa treści w formacie json, przesyłanej do i odbieranej z REST API udostępnianego przez aplikację serwerową; interfejs do rejestrowania aktywności programu (włącznie z konfiguracją ów interfejsu); obsługa systemu plików w celu tymczasowego zapisu wykonywanych zrzutów ekranu; rozpoznanie systemu operacyjnego, na którym uruchomiona jest aplikacja, w celu obsługi elementów algorytmów specyficznych dla danej platformy; czy obsługa daty oraz czasu, niezbędnej składowej

informacji przesyłanych w całym systemie.

Zewnętrzne moduły wymagane były do zaimplementowania funkcjonalności: pobierania listy aktywnych procesów, wykonywania zrzutów ekranów i wykonywania zapytań HTTP.
W pierwszy przypadku zastosowany został moduł psutil (process and system utilities), cross-platformowa biblioteka udostępniająca interfejs do rejestracji informacji wspomnianych powyżej, a poza tym informacji o całej eksploatacji systemu (statystyki dotyczące między innymi użycia procesora, pamięci RAM oraz dyskowej, a także sieci). Cytując oficjalną dokumentację, implementuje on wiele funkcjonalności oferowanych przez unixowe narzędzia konsolowe, takie jak: ps, top, lsof, netstat, ifconfig, who, df, kill, free, uptime czy pidof. Wśród wspieranych platform są systemy operacyjne Linux, Windows oraz macOS, zarówno w architekturze 32-bit, jak i 64-bit.
Jest to biblioteka stabilna i szeroko stosowane przez organizacje takie jak Facebook oraz Google.

Do implementacji funkcjonalności wykonywania zrzutów ekranu zastosowany został moduł mss (multiple screen shots), przedstawiany przez oficjalną dokumentację jako “ultra szybki, cross-platformowy moduł napisany w czystym Pythonie”, zgodny z Pythonem 2, 3, PEP8 oraz pozbawiony zewnętrznych zależności.
Moduł ten ma kilka niedoskonałości, jak problemy z zapisaniem zrzutów do zmiennej zamiast bezpośrednio do pliku znajdującego się w pamięci dyskowej; nie jest to jednak duża przeszkoda i w ogólnym rozrachunku moduł ten okazał się wygodny i rzeczywiście wydajny pod kątem szybkości.

Do wykonywania żądań HTTP natomiast wykorzystany został popularny moduł Requests, zapewniający bardzo wygodny interfejs do wykonywania ów zapytań, wraz z obsługą takich kwestii jak treść w formacie json czy autoryzacja.

Czyni to aplikację kliencką wieloplatformową, co zostało potwierdzone testami na systemach Linux (dystrybucje Ubuntu 16.04 LTS oraz Antergos), Windows 10 (Education i Education N) oraz macOS (El Sierra), wszystkie w wersjach 64-bit.

W celu zapewnienia dobrej jakości kodu wykorzystano narzędzia pylint (linter sprawdzający zgodność wejściowego kodu z zasadami PEP8) oraz autopep8 (narzędzie formatujące kod zgodnie z wytycznymi PEP8).

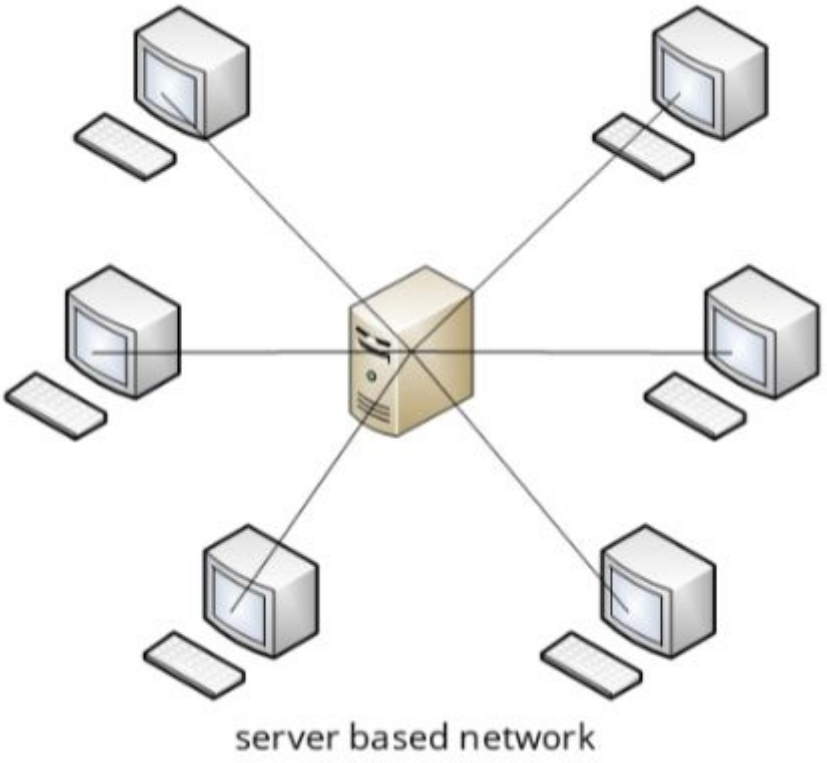
6. Architektura systemu

W tym rozdziale zostanie przedstawiona architektura systemu *Sauron*.

- Architektura aplikacji klienckiej prowadzącego

Architektura oparta jest o model klient-serwer. W tym modelu urządzenia podzielone są na te, które oferują usługi (serwer) oraz

takie, które korzystają z owych usług (klienci). Klientem jest oprogramowanie, które żąda dostępu do danej usługi lub zasobu. Może pracować w trybach: aktywnym, wysłaniu żądania do serwera oraz oczekiwaniu na odpowiedź od serwera. Serwer jest oprogramowaniem świadczącym usługi lub udostępniającym zasoby. Może działać w trybach pasywnym lub oczekiwaniu na żądanie od klienta.



Rys. 6.1. Przykład podstawowej architektury klient-serwer.

Sieć ta jest zarządzana przez specjalnego użytkownika nazwanego administratorem sieci, który jest niedostępny z poziomu dostępu aplikacji prowadzącego. Ów aplikacja korzysta z utworzonych w bazie danych użytkowników prowadzących, którzy są w stanie manipulować danymi w bazie danych. Mają oni możliwość zalogowania się do systemu za pomocą ekranu logowania. Do zadań aplikacji prowadzącego należą: pobieranie danych o procesach oraz zrzutach ekranów użytkowników podłączonych jako studenci, pobieranie danych o listach zakazanych procesów, możliwość edycji, dodawania oraz usuwania zakazanych list. Poniżej zostaną przedstawione mikroserwisy wraz z metodami, z których korzysta aplikacja prowadzącego. Warto zwrócić uwagę na endpoint whitelist, który służy do obsługi list zakazanych, jednak problem ten został opisany w rozdziale Napotkane problemy.

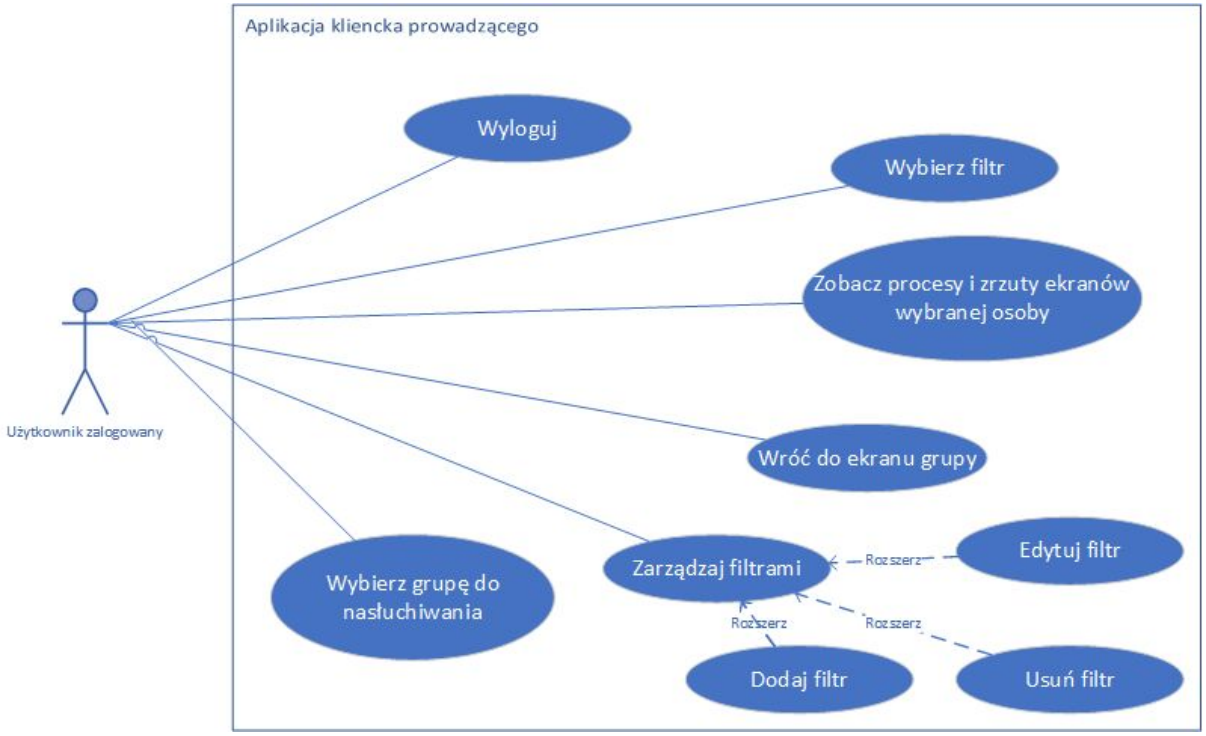
Nazwa serwisu	Endpoint	Opis
UserApi	**/api/auth**	Metody: **GET** - pobiera kod poprawnego zalogowania lub błędnego, w przypadku złych danych logowania, używa następującego nagłówka: ● **Authorization** - określa token autoryzacyjny,
BlacklistApi	**/api/whitelist**	Metody: **GET** - zwraca wszystkie dostępne listy zakazane, używa następujących nagłówków: ● **Authorization** - określa token autoryzacyjny, oraz następujących parametrów: ● **user** - określa użytkownika, który ma zamiar pobrać zakazane listy, ● **only_active** - określa, czy należy pobrać jedynie aktywne listy, **PATCH** - aktualizacja pojedynczych wartości w liście zakazanej, używa następujących nagłówków: ● **Authorization** - określa token autoryzacyjny, oraz parametrów: ● **id** - identyfikator aktualizowanej listy, ● **active** - wartość boolowska oznaczająca, czy lista ma być aktywowana czy dezaktywowana, **DELETE** - usuwanie wybranej listy procesów zakazanych, używa następujących nagłówków: ● **Authorization** - określa token autoryzacyjny, oraz następujących parametrów: ● **id** - identyfikator usuwanej listy, **POST** - dodanie nowej listy, używa następujących nagłówków: ● **Authorization** - określa token autoryzacyjny, oraz następującego ciała żądania:

		● **name** - nazwa dodawanej listy, ● **processes** - zbiór procesów zakazanych, ● **group** - nazwa grupy, do której lista jest przypisana, **PUT** - aktualizowanie całej zawartości wybranej listy zakazanych procesów, używa następujących nagłówków: ● **Authorization** - określa token autoryzacyjny, oraz następujących parametrów: ● **id** - identyfikator aktualizowanej listy, oraz następującego ciała żądania: ● **name** - nazwa aktualizowanej listy, ● **processes** - zbiór procesów zakazanych, ● **group** - nazwa grupy, do której lista jest przypisana,
ScreenshotlistApi	**/api/screenshotlist**	Metody: **GET** - pobieranie całej listy zrzutów ekranów z konkretnego przedziału określonego w następujących nagłówkach: ● **Authorization** - określa token autoryzacyjny, oraz parametrach: ● **time_from** - określenie czasu od którego ma być pobierana lista zrzutów ekranów, ● **time_to** - określenie czasu do którego ma być pobierana lista zrzutów ekranów, ● **group** - nazwa grupy, w której zrzuty ekranów są wykonywane, ● **newest** - parametr wskazujący na chęć otrzymania tylko najnowszych informacji o zrzutach ekranów dla każdej aplikacji klienckiej studenta z osobna,
ScreenshotApi	**/api/screenshot**	Metody: **GET** - pobieranie konkretnego zrzutu

		ekranu dla następujących nagłówków: ● **Authorization** - określa token autoryzacyjny, oraz następujących parametrów: ● **filename** - nazwa konkretnego zrzutu ekranu,
ProcessApi	**/api/process**	Metody: **GET** - pobieranie wszystkich procesów w przedziale czasowym oraz dla konkretnej grupy dla następujących nagłówków: ● **Authorization** - określa token autoryzacyjny, oraz następujących parametrów: ● **time_from** - określenie czasu od którego ma być pobierana lista procesów, ● **time_to** - określenie czasu do którego ma być pobierana lista procesów, ● **group** - nazwa grupy, w której procesy są wykonywane,

Tabela 6.1. Wykorzystane mikroserwisy.

Poniżej zostaną przedstawione diagramy UML przypadków użycia dla aplikacji klienckiej prowadzącego. Zostały podzielone ze względu na użytkownika zalogowanego oraz użytkownika niezalogowanego.



Rys. 6.2. Diagram UML przypadków użycia użytkownika zalogowanego.

Na powyższym rysunku w systemie aplikacji klienckiej prowadzącego, użytkownik zalogowany ma możliwość wylogowania, wybrania używanego filtru, tj. listy procesów zakazanych, podejrzenia procesów oraz zrzutów ekranu wybranej przez siebie osoby, zarządzania filtrami wraz z edycją, dodaniem nowego oraz usunięciem, powrotu do ekranu grupy z widoku zarządzania filtrami, a także wybrania grupy do nasłuchiwania. Wszystkie powyższe funkcjonalności zostały zaimplementowane zgodnie z oczekiwaniami. Wykorzystanie powyższych funkcjonalności zostało przedstawione w rozdziale Instrukcja użytkowania aplikacji.

Rys. 6.3. Diagram UML przypadków użycia użytkownika niezalogowanego.

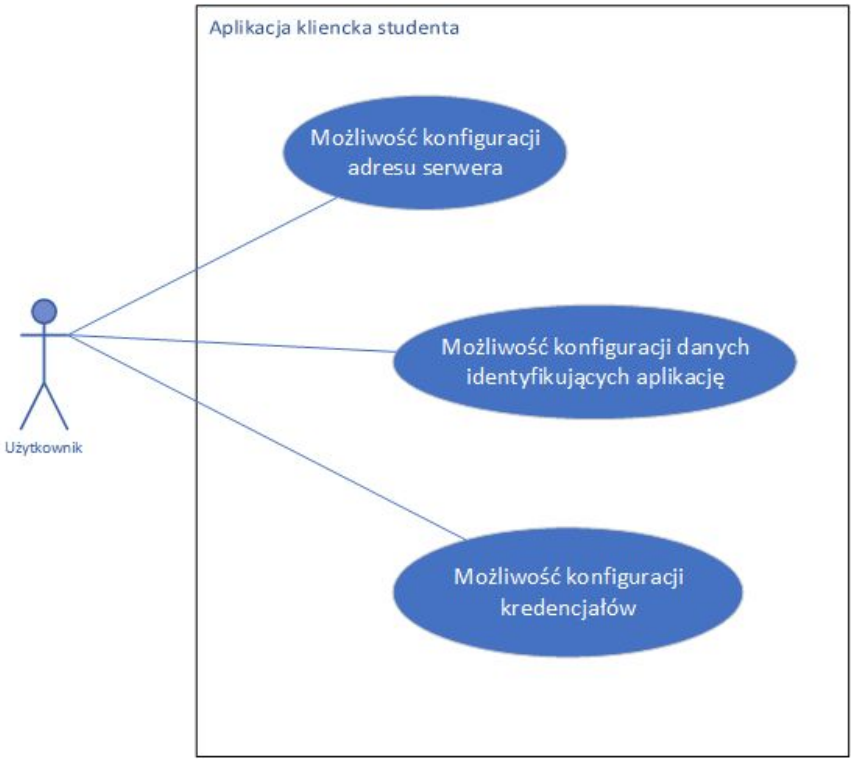
Na powyższym rysunku w systemie aplikacji klienckiej prowadzącego, użytkownik niezalogowany ma jedynie możliwość zalogowania się do systemu.

- Architektura aplikacji klienckiej studenta

Aplikacja kliencka studenta napisana jest w języku Python 3.x (z założeniem kompatybilności z wersją od 3.5.2 wzwyż), z wykorzystaniem zewnętrznych modułów psutil, mss oraz Requests. W równych odstępach czasu odpytuje ona system operacyjny o listę aktywnych w danym momencie procesów oraz wykonuje zrzuty ekranu (obie te operacje są jednak wykonywane niezależnie), zapisując informację o czasie tej czynności (w postaci unixowego znacznika czasu), a następnie wysyła wszystkie zebrane informacje do serwera poprzez udostępniany przez niego interfejs REST API.
W przypadku gdy nie powiedzie się wysłanie danych (np. z powodu tymczasowej utraty połączenia z internetem), zachowywane są one w pamięci i przy kolejnej próbie wysyłane zbiorczo wraz z nowymi danymi.

Ze względu na wyżej wspomnianą informację o czasie oraz budowę interfejsu REST API, nawet w takiej sytuacji możliwe jest łatwe rozróżnienia, w jakim momencie czasu dana podkolekcja procesów została zebrana lub dany zrzut ekranu wykonany.
Zrzuty ekranu są tymczasowo zapisywane w pamięci dyskowej, do czasu wysłania z powodzeniem danego zrzutu do serwera.

Poniżej zostanie przedstawiony diagram UML przypadków użycia dla użytkownika aplikacji.



Rys.6.4. Diagram UML przypadków użycia dla użytkownika aplikacji studenta.

Użytkownik w systemie aplikacji klienckiej studenta ma możliwość konfiguracji adresu serwera docelowego, możliwość konfiguracji danych indentyfikujących aplikację oraz możliwość konfiguracji kredencjałów.

- Architektura aplikacji serwerowej

Aplikacja serwerowa udostępnia serwisy REST napisane w języku Python3.6 z wykorzystaniem minimalistycznego frameworka do tworzenia aplikacji webowych - cherrypy oraz z modułu ułatwiającego pracę z systemem bazy danych MongoDB - Pymongo. Zostało utworzonych 7 serwisów, które są wykorzystywane zarówno przez aplikację prowadzącego jak i przez aplikacje klienckie (“nazgule”). Serwisy te są wystawione na różnych endpointach oraz udostępniają metody HTTP, które umożliwiają ograniczoną interakcję z bazą danych. Serwer został implementowany z myślą o systemie operacyjnym Ubuntu Server w wersji 18.04. Poniżej został umieszczony opis serwisów wraz z dostępnymi metodami oraz parametrami:

Serwis użytkowników - został on stworzony z myślą o umożliwieniu osobie zarządzającej systemem posiadania kontroli nad kontami użytkowników. Umożliwia on dodawanie, usuwanie oraz modyfikowanie kont użytkowników oraz wylistowanie wszystkich kont.

Metoda **GET** - pozwala pobrać listę użytkowników wraz z wszystkimi ich danymi (login, hasło oraz typ konta). Wymagany status: **superadmin**.

Metoda **POST** - pozwala na utworzenie nowego konta użytkownika. Wymaga podania w ciele metody słownika w formacie JSON, który zawiera pola *login*, *password* oraz *account_type*, które odpowiadają przypisaniu do użytkownika kolejno loginu, hasła oraz typu konta. Wymagany status: **superadmin**.

Metoda **PUT** - pozwala na zmodyfikowanie informacji o użytkowniku - zmodyfikowania hasła lub typu konta. Wymaga podania loginu jako parametru (*login* w przypadku parametru typu query) oraz podania w ciele metody słownika w formacie JSON, który zawiera pola *password* oraz *account_type*, które oznaczają kolejno nowe hasło użytkownika oraz typ konta. Wymagany status: **superadmin**.

Metoda **DELETE** - pozwala na usunięcie konta użytkownika. Po wykonaniu tej operacji z bazy danych zostaną usunięte wszystkie informacje o użytkowniku z kolekcji users (zniknie informacja o loginie i powiązanym z nich haśle i typie konta). Wymaga podania loginu jako parametru (*login* w przypadku parametru typu query). Wymagany status: **superadmin**.

Serwis listy nazguli - pozwala użytkownikowi na wypisanie listy nazguli (aplikacji klienckich), które nadają (wysyłają informacje) od określonego momentu.

Metoda **GET** - zwraca listę nazguli, które nadają. Wymaga podania czasu, od którego ma następować sprawdzanie jako parametru (*time_from* w przypadku parametru typu query). Opcjonalnymi parametrami, które mogą dodatkowo filtrować listę są: czas, do którego ma następować sprawdzanie oraz grupa, do której ma się ograniczać sprawdzanie (odpowiednio *time_to* oraz *group* w przypadku parametrów typu query). Wymagany status: **superadmin** lub **user**.

Serwis autoryzujący - serwis pomocniczy ułatwiający weryfikowanie poprawności danych logowania.
Metoda **GET** - zwraca kod 200 w przypadku poprawnego zalogowania oraz 401 w przeciwnym wypadku. Wymagany status: **superadmin** lub **user**.

Serwis obsługujący listy niedozwolonych procesów - pozwala użytkownikom (zarówno z aplikacji prowadzącego jak i nazgulom) na korzystanie z bazy zawierającej listy niedozwolonych procesów.

Metoda **GET** - pozwala pobrać listę list niedozwolonych procesów. Opcjonalne parametry, które pozwalają dodatkowo filtrować listy to wskaźnik czy mają być pobrane tylko aktywne listy (domyślnie ustawiony na True), nazwa grupy, której mają dotyczyć listy oraz nazwa konkretnego użytkownika który zdefiniował daną listę (odpowiednio *only_active, group* oraz *user* w przypadku parametrów typu query). Wymagany status: **superadmin, nazgul** lub **user**.

Metoda **POST** - pozwala na utworzenie nowej list niedozwolonych procesów. Wymaga podania w ciele metody słownika w formacie JSON, który zawiera pola *name, processes* oraz *group*, które odpowiadają przypisaniu do listy odpowiednio nazwy listy, listy procesów oraz grupy, której lista ma dotyczyć. Domyślnie utworzona lista ma status nieaktywny. Każda lista ma losowo generowane unikatowe id, które ma ułatwiać później odwoływanie się do niej. Wymagany status: **superadmin** lub **user**.

Metoda **PATCH** - pozwala użytkownikom na aktywację lub dezaktywację wybranej listy procesów niedozwolonych. Wymaga podania identyfikatora listy oraz wskaźnika czy lista ma być aktywowana czy wręcz przeciwnie jako parametrów (*id* oraz *active* w przypadku parametrów typu query). Wymagany status: **superadmin** lub **user**.

Metoda **PUT** - pozwala użytkownikom na modyfikację informacji, które

są przechowywane w bazie o danej liście. Umożliwia ona zmianę procesów, nazwy oraz grupy, której dotyczy dana lista. Wymaga podania identyfikatora listy jako parametr (*id* w przypadku parametrów typu query). Wymaga podania w ciele metody słownika w formacie JSON, który zawiera pola *name, processes* oraz *group*, które odpowiadają przypisaniu do listy odpowiednio nazwy listy, listy procesów oraz grupy, której lista ma dotyczyć.

Metoda **DELETE** - pozwala użytkownikom na usunięcie wybranej listy niedozwolonych procesów. Wymaga podania identyfikatora listy jako parametr (*id* w przypadku parametrów typu query). Wymagany status: **superadmin** lub **user**.

Serwis obsługujący zrzuty ekranu - serwis pozwalający użytkownikom na obsługiwanie zrzutów ekranu, które są przechowywane na serwerze.

Metoda **GET** - pozwala użytkownikom na pobranie konkretnego zrzutu ekranu przekonwertowanego do formy base64. Wymaga podania nazwy pliku jako parametr (*filename* w przypadku parametrów typu query). Wymagany status: **superadmin** lub **user**.

Metoda **POST** - pozwala użytkownikom na wrzucenie zrzutu ekranu na serwer. Plikowi generuje się nazwa zawierająca grupę, login nazgula raz czas wrzucenia pliku. Wymaga podania jako parametrów pliku screenshot (nazwa parametru o identycznej nazwie) oraz daty utworzenia pliku i grupy, której dotyczy dany zrzut (odpowiednio *create_time* oraz *group* w przypadku parametrów typu query). Wymagany status: **nazgul**.

Serwis listy zrzutów ekranu - serwis pozwala na pobranie listy informacji o zrzutach ekranu, które są przechowywane w bazie danych na serwerze.

Metoda **GET** - pozwala użytkownikom na pobranie listy z informacjami o zrzutach ekranu. Każdy element takiej listy zawiera nazwę pliku, nazwę nazgula, nazwę grupy oraz czas utworzenia pliku. Używając opcjonalnych parametrów można dodatkowo filtrować listę. Można sprecyzować czas, w którym pliki zostały utworzone (odpowiednio *time_from* i *time_to* w przypadku parametrów typu query), określić, którego nazgula i której grupy mają dotyczyć zrzuty ekranu (odpowiednio *nazgul* i *group*) oraz użyć parametru określającego, że mają zostać wysyłane tylko najnowsze informacje dla danego nazgula (parametr noszący nazwę *newest). W*ymagany status: **superadmin** lub **user**.

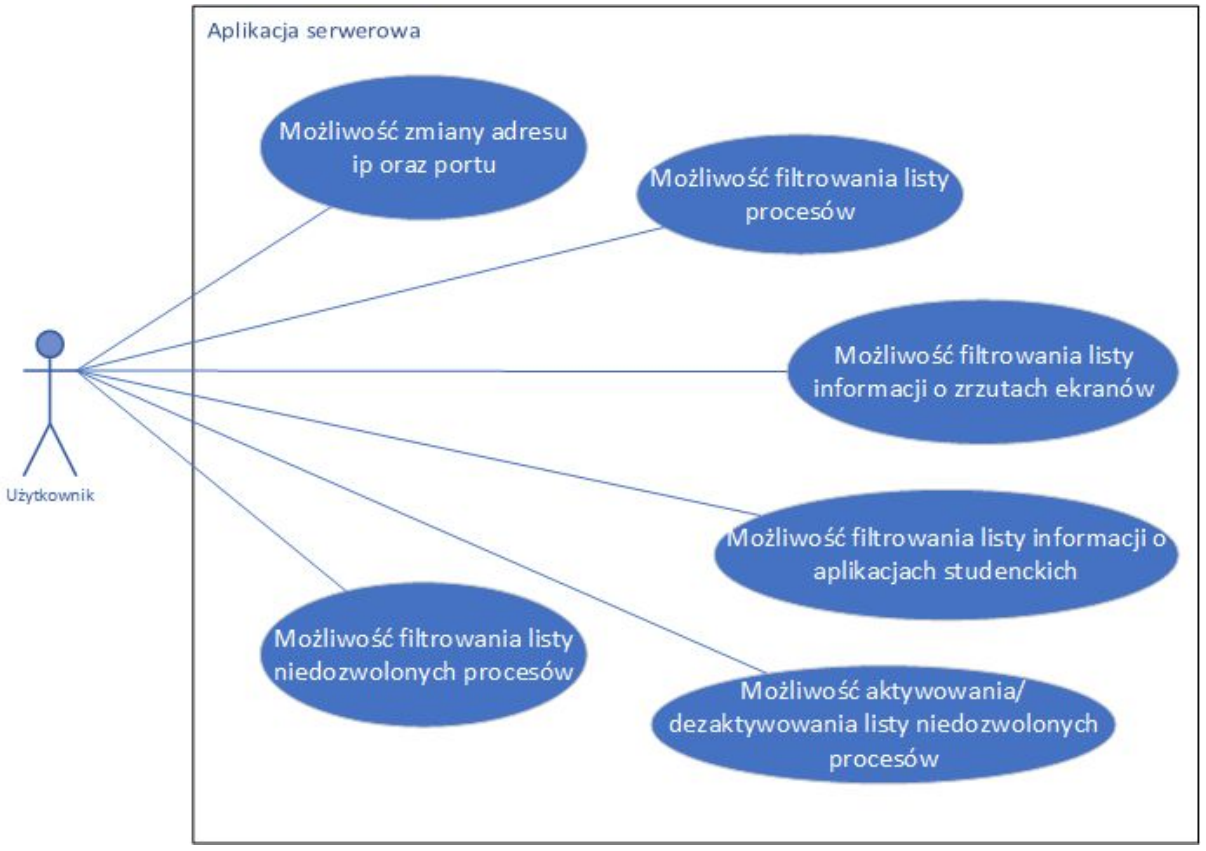
Serwis obsługujący procesy - pozwala użytkownikom na zarządzanie listami procesów, które są przechowywane w bazie danych na

serwerze.

Metoda **GET** - pozwala użytkownikom na pobranie list procesów. Przechowywane informacje zawierają takie dane jak czas wysłania listy procesów, nazwa nazgula, który wysłał procesy, listę procesów, które były uruchomione w momencie wysyłania, grupa, do której podłączony był nazgul oraz informacja o tym czy dana lista procesów zawierała jakiś alarmowy. Można dodatkowo filtrować listę procesów, korzystając z opcjonalnych parametrów. Można sprecyzować czas, w którym procesy zostały wysłane (odpowiednio *time_from* i *time_to* w przypadku parametrów typu query), nazgula, od którego zostały wysłane listy procesów oraz grupę, która ma zostać monitorowana (odpowiednio *nazgul* i *group*). *W*ymagany status: **superadmin** lub **user**.

Metoda **POST** - pozwala użytkownikom na wysyłanie listy procesów do bazy danych na serwerze. Wymaga podania w ciele metody słownika w formacie JSON, który zawiera pola zawierające czas utworzenia listy procesów, listę procesów, które były uruchomione w momencie wysyłki, grupę, której dotyczy dana lista oraz informacja o tym czy dana lista zawiera jakiś niedozwolony proces (nazwy pół to odpowiednio *create_time, processes, group* oraz *alarm*). Dodatkowo do listy dopisywana jest nazwa nazgula, który wysłał dane zapytanie. Wymagany status: **nazgul**.

Poniżej zostanie przedstawiony diagram UML przypadków użycia dla użytkownika aplikacji.



Rys 6.5. Diagram UML przypadków użycia dla użytkownika aplikacji serwerowej.

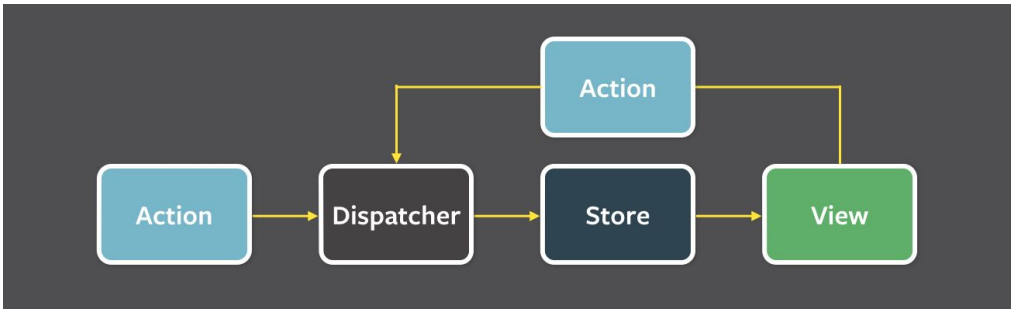
Użytkownik w systemie aplikacji serwerowej ma możliwość zmiany adresu ip oraz portu, na którym działać ma serwer, możliwość filtrowania listy procesów za pomocą parametrów określających przedział czasowy, konkretną grupę oraz nazwę nazgula, możliwość filtrowania listy informacji o zrzutach ekranu za pomocą parametrów określających przedział czasowy, konkretną grupę, nazwę nazgula oraz określenie czy mają być pobierane tylko najnowsze informacje dla każdego nazgula osobno, możliwość filtrowania listy informacji o nazgulach za pomocą parametrów określających przedział czasowy nadawania oraz konkretną grupę, możliwość filtrowania listy zawierającej informacje o niedozwolonych procesach oraz możliwość aktywowania lub dezaktywowania danej listy procesów niedozwolonych.

7. Opis implementacji

W tym rozdziale zostaną przedstawione ciekawsze rozwiązania zastosowane w systemie monitorowania urządzeniami w sali laboratoryjnej.

- Aplikacja kliencka prowadzącego została stworzona przy użyciu frameworka Vue dla języka JavaScript. Głównym aspektem tworzenia

aplikacji webowych w tym frameworku jest definiowanie widoków będących komponentami struktury aplikacji. Każdy komponent może mieć swój wewnętrzny stan, lecz ze względu na mnogość tychże stanów w bardziej rozwiniętych aplikacjach zalecane jest użycie biblioteki implementującej architekturę Flux, która została opracowana przez Facebook. Architektura ta opiera się na tym, że stan całej aplikacji mieści się w sklepie (z ang. store), którego to właściwości zmieniane są przy pomocy akcji, które to wywołują zmianę tego stanu poprzez dispatch.



Rys. 7.1 Architektura Flux.

Zespół odpowiedzialny za framework Vue stworzył dedykowane rozwiązanie implementujące tę architekturę pod nazwą Vuex. Jest to więc najbardziej zalecana biblioteka, ale można też skorzystać z innych rozwiązań takich jak Redux czy Mobx.

We wszelkich miejscach w aplikacji, gdzie potrzebne jest odświeżanie jakiejś wartości należy połączyć ją z odpowiednim stanem w sklepie, a zmiana tej wartości poprzez dispatch wywoła wyrenderowanie nowej wartości. W przypadku asynchronicznego pobierania pewnych danych z serwera obsługiwane są przypadki, gdzie coś się nie powiedzie i wyświetlany jest błąd w postaci pojawiającego się okna informacyjnego. Do tego przy pewnych akcjach wymagane jest czekanie na jej zakończenie, a więc została zaimplementowana animacja kręcącego się okręgu w celu pokazania, że aplikacja pracuje nad zadaną przez użytkownika akcją.

- Aplikacja kliencka studenta została stworzona przy użyciu języka Python 3.x. (z założeniem kompatybilności z wersją od 3.5.2 wzwyż) i zewnętrznych modułów psutil (w celu implementacji funkcjonalności pobierania listy aktywnych procesów w systemie), mss (w celu implementacji funkcjonalności wykonywania zrzutów ekranu) oraz Requests (w celu wykonywania zapytań HTTP do interfejsu REST API udostępnionego przez serwer). Wykorzystany został paradygmat programowania zorientowanego obiektowo; w trakcie implementacji dążono także do zgodności z PEP8 oraz ogólnymi dobrymi praktykami programistycznymi.

- Aplikacja serwerowa została podzielona na 7 podserwisów, które

udostępniają funkcje, które dotyczą konkretnych funkcjonalności. Dla ułatwienia konfiguracji, została ona wyodrębniona do osobnego pliku, którego edycja może skutkować zmianą adresu IP oraz portu, na którym ma zostać uruchomiona usługa. Same serwisy są podzielone na 4 tematyczne pliki, które dotyczą użytkowników, zrzutów ekranu, procesów oraz list niedozwolonych procesów. Każdy z tych plików zawiera co najmniej jedną klasę, która jest udekorowana dekoratorem cherrypy.expose. Każda klasa zawiera metody o nazwach zgodnych z nazwami metod HTTP (np. GET, POST czy DELETE). Gdy metoda zakłada przyjmowanie w ciele metody HTTP tekstu w formacie JSON jest ona zdobiona dekoratorem cherrypy.tools.json_in(), co wymusza na użytkowniku metody utrzymanie ciała metody w takim formacie. W przypadku gdy metoda ma zwracać test w formacie JSON, zdobiona jest ona dekoratorem cherrypy.tools.json_out(). Wszystkie te klasy korzystają z pliku zawierającego informacje o bazie danych (o nazwie bazy oraz nazwach kolekcji) oraz menadżer kontekstu ułatwiający korzystanie z tychże kolekcji. Wszystkie klasy są dołączane do drzewa cherrypy, które wystawia je na odpowiednich endpointach. Całość aplikacji napisana jest w języku Python w wersji 3.6 wraz z wykorzystaniem bibliotek cherrypy (web framework) oraz pymongo (biblioteka ułatwiająca korzystanie z MongoDB w Pythonie). Baza danych jest obsługiwana przez system MongoDB, w którym została utworzona baza “sauron” oraz kolekcje takie jak “users” czy “processes”. Wysyłanie obecnych na serwerze zrzutów ekranu odbywa się za pomocą przekonwertowania ich na base64, a dodawanie nowych zrzutów ekranu na serwer wykorzystuje dodanie pliku do form-data w metodzie POST serwisu obsługującego zrzuty.

8. Napotkane problemy

W poniższym rozdziale zostaną przedstawione ciekawsze napotkane przez zespół programistyczny problemy oraz sposoby ich rozwiązania.

- Wybór grupy do podsłuchiwania procesów - w fazie koncepcyjnej projektu grupa miała być wybierana poprzez wybór dostępnych z listy. Założenie dotyczyło, że programy nasłuchujące procesy będą przypisane do danych stanowisk. Z punktu widzenia wygody korzystania z aplikacji prowadzącego jak i uruchamiania aplikacji podsłuchującej procesy byłoby to wygodne, ponieważ prowadzący miałby listę dostępnych grup, a studenci mieliby tylko wpisywać swoje imię i nazwisko. Takie rozwiązanie wymagałoby jednak manualnego zarządzania przez administratora, aby przypisywał dane jednostki do grup. Innym wyjściem, które okazało się korzystniejsze ze względu na łatwość konfiguracji i większą elastyczność jest wymuszenie konfiguracji nadawania przez studenta i wybór grupy przez prowadzącego poprzez jej wpisanie.

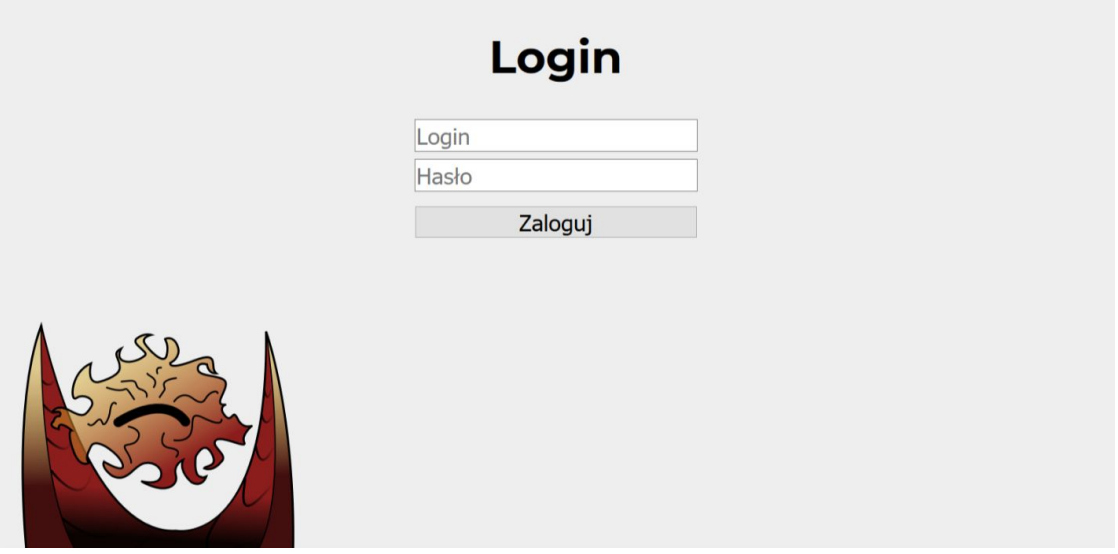
- Blacklista - w początkowych założeniach systemu, miała istnieć lista dozwolonych procesów, jednak podczas prac implementacyjnych został wykryty zbyt niski potencjał tej metody. Whitelista byłaby niemożliwa lub bardzo trudna do poprawnego zdefiniowania, gdyż prowadzący musiałby ręcznie wpisywać każdy możliwy przypadek dozwolonego procesu. Ostatecznie whitelista została zastąpiona blacklistą, umożliwiającą znacznie szybsze wprowadzanie zmian.

- Problem optymalizacji czasu przesyłania danych - problem może być rozwiązany przez filtrowanie danych po stronie serwera, jednak jest to funkcjonalność przewidziana jako przyszły rozwój

9. Instrukcja użytkowania aplikacji

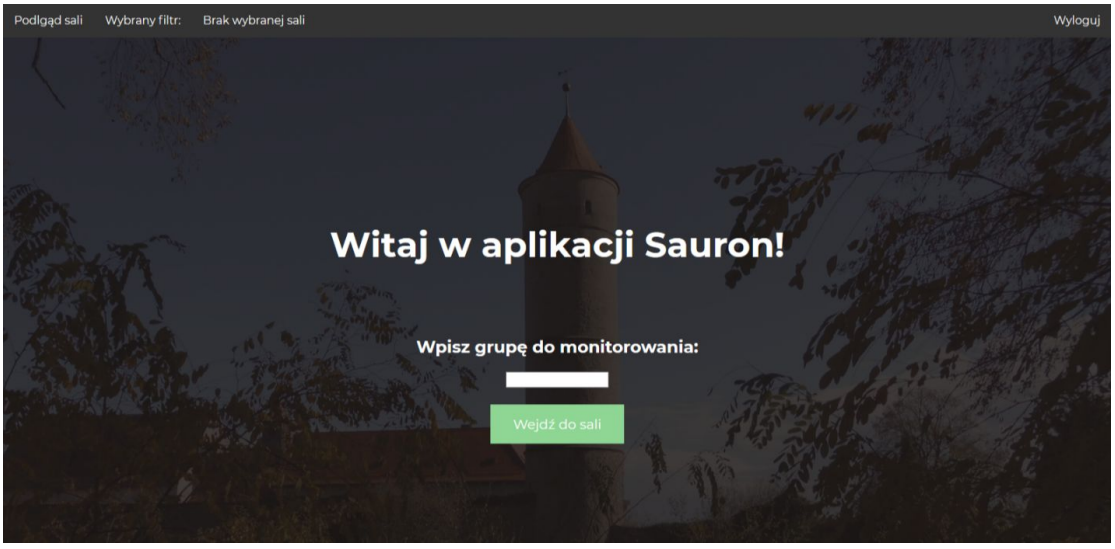
W poniższym rozdziale zostanie zaprezentowana instrukcja obsługi systemu monitorowania urządzeń w sali laboratoryjnej.

Jako pierwsza zostanie przedstawiona aplikacja kliencka prowadzącego. Prowadzący posiada jedno z kilku kont utworzonych specjalnie na potrzeby pobierania danych. Przechowywane są w bazie danych po stronie serwera.



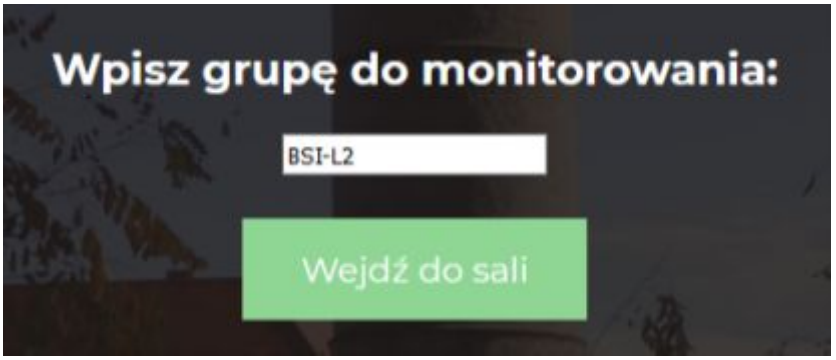
Rys. 9.1. Widok ekranu logowania.

Po uruchomieniu aplikacji, użytkownik prowadzący przekierowywany jest na stronę logowania. Jest tam proszony o podanie nazwy użytkownika oraz hasła do konta. Następnie dane wysyłane są do aplikacji serwerowej, gdzie weryfikowana jest poprawność wprowadzonych danych. Jeśli weryfikacja się powiodła, użytkownik przekierowywany jest do strony głównej aplikacji.



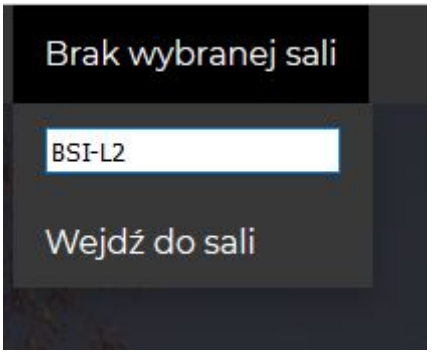
Rys. 9.2. Widok ekranu strony głównej

Kolejnym krokiem, po przejściu weryfikacji logowania, jest wybranie grupy do monitorowania co ma uprościć ekran pokazany na rysunku 9.3. Jest on widoczny tylko po zalogowaniu do czasu wyboru grupy. Korzystając z tego ekranu należy wpisać grupę w widoczne pole, a następnie nacisnąć przycisk ‘Wejdź do sali’.



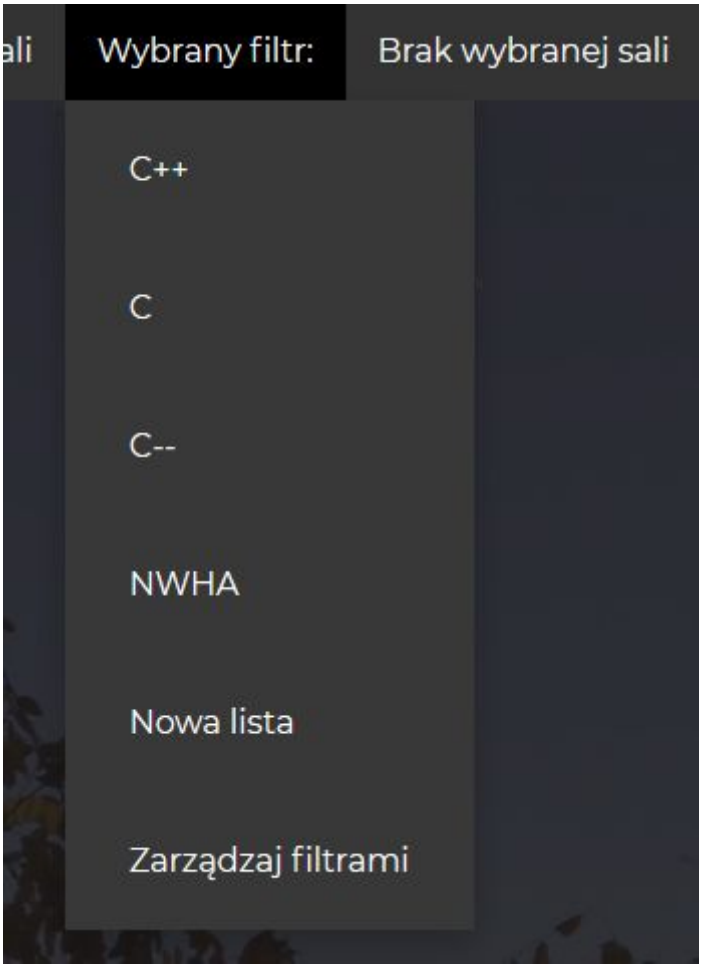
Rys. 9.3. Widok wybrania sali sposobem pierwszym.

Drugim sposobem jest rozwinięcie paska w górnym menu ‘Brak wybranej sali’ oraz wpisanie w wyznaczone miejsce nazwy grupy. Następnie należy nacisnąć przycisk ‘Wejdź do sali’ mieszczący się pod miejscem do wpisania nazwy grupy. Drugi sposób został przedstawiony na rysunku 9.4.



Rys. 9.4. Widok wybrania sali sposobem drugim.

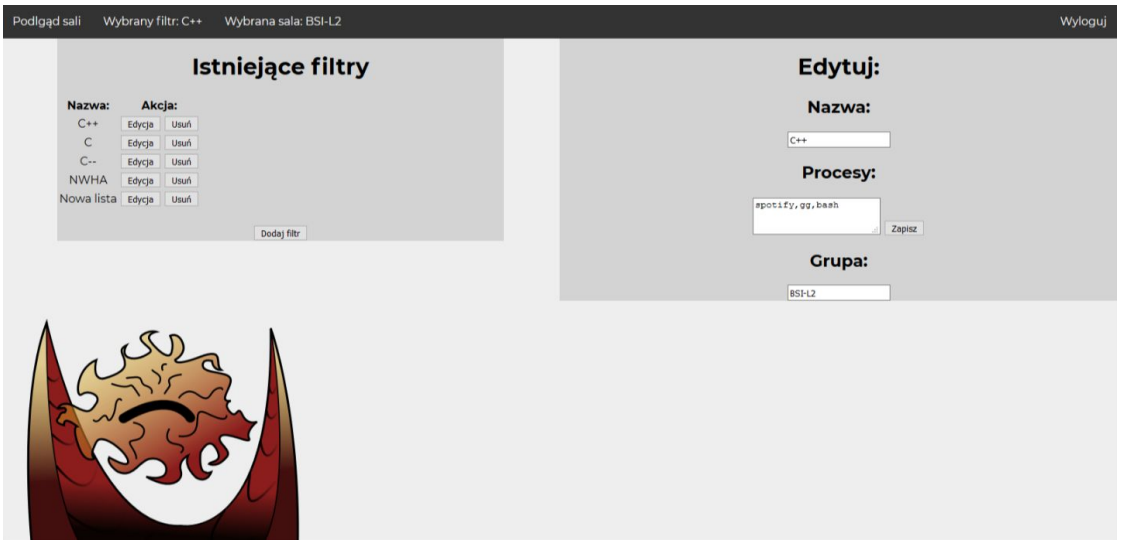
Aby móc wybrać listę zakazanych procesów, należy rozwinąć pasek w górnym menu o nazwie ‘Wybrany filtr’. Wyświetlone zostaną aktualnie dostępne listy na serwerze. Wybranie jednej z nich skutkuje ustawieniem listy dla wszystkich użytkowników aplikacji studenckich. Sposób wykonania tej czynności został pokazany na rysunku 9.5. W przypadku braku wybrania listy, żadne ograniczenia nie zostaną zastosowane.



Rys. 9.5. Widok wybrania listy procesów zakazanych.

Po wybraniu opcji ‘Zarządzaj filtrami’, użytkownik zostanie przekierowany do strony z możliwością zarządzania listami procesów

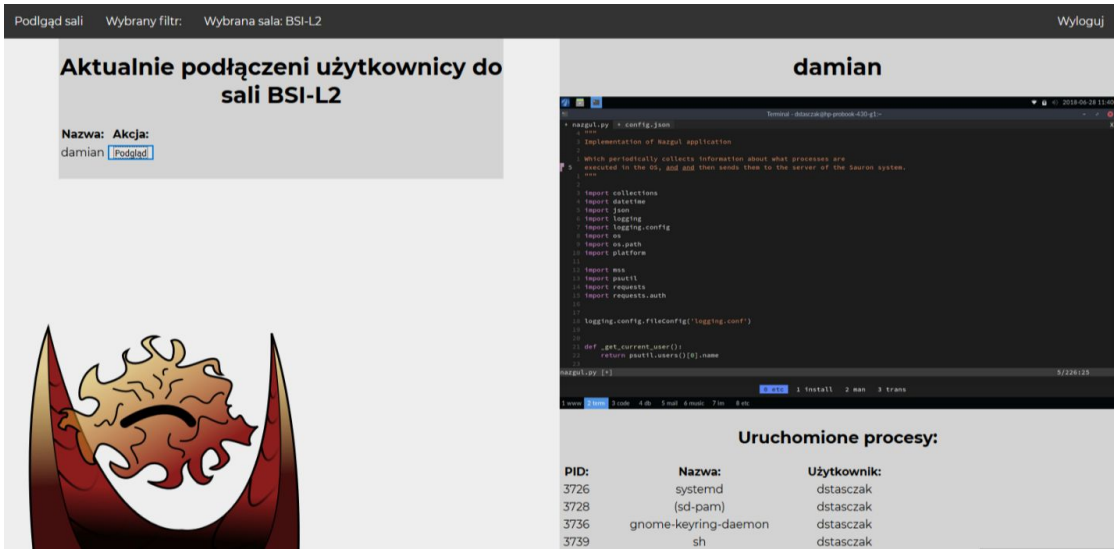
zakazanych. Widok tej operacji został przedstawiony na rysunku 9.6.



Rys. 9.6. Widok ekranu strony zarządzania filtrami.

Po przekierowaniu do strony z zarządzaniem filtrami, użytkownik ma kilka możliwości do wykonania. Opcja ‘Dodaj filtr’ umożliwia dodanie nowej listy procesów zakazanych wraz z jej nazwą oraz grupą, do której ma zostać przypisana. Domyślną wartością dla nazwy listy jest ‘Nowa lista’. Procesy, które mają zostać zakazane, należy wpisywać oddzielając kolejne znakiem ‘,’ (przecinka). Po uzupełnieniu formularza należy nacisnąć przycisk ‘Zapisz’ w celu zapisania nowej listy. Opcja ‘Edycja’ oznacza pobranie aktualnych wartości dla nazwy, procesów oraz grupy z bazy danych oraz stwarza możliwość edycji tychże wartości. Opcja ‘Usuń’ oznacza trwałe usunięcie konkretnej listy. Aby powrócić do strony głównej, należy nacisnąć przycisk ‘Podgląd sali’ mieszczący się w górnym menu

.



Rys. 9.7. Widok ekranu głównego z podglądem grupy.

Ostatnim, a zarazem najważniejszym ekranem, jest widok ekranu głównego z podglądem grupy. Pozwala użytkownikowi na podgląd procesów oraz zrzutów ekranów użytkowników aktualnie podłączonych do systemu jako aplikacje studentów. Aby zobaczyć podgląd procesów, należy nacisnąć przycisk ‘Podgląd’ mieszczący się przy każdym z użytkowników. W przypadku wystąpienia oszustwa po stronie aplikacji klienckiej studenta, jego nazwa zostanie podświetlona kolorem czerwonym.

Jako druga przedstawiona zostanie aplikacja kliencka studenta. Jest to skrypt, którego jedynym interfejsem z użytkownikiem są pliki konfiguracyjne: *logging.conf*, przeznaczony do edycji procesu logowania, a więc i przeznaczony de facto bardziej dla programisty/testera, aniżeli dla właściwego użytkownika; oraz *config.json*, umożliwiający konfigurację takich właściwości jak adres serwera, kredencjały czy częstotliwość komunikacji z serwerem.

```
{
    "server": {
        "hostname": "http://example.com/sauron",
        "process_endpoint": "/api/process/",
        "screenshots_endpoint": "/api/screenshot/",
        "blacklist_endpoint": "/api/whitelist/"
    },
    "name": "john.doe@student.put.poznan.pl",
    "password": "1234",
    "group": "BSI-L2",
    "time period": {
        "server communication": 10
    }
}
```

Rys. 9.8. Przykładowy plik konfiguracyjny aplikacji klienckiej studenta *config.json.*

W przypadku nie podania nazwy, aplikacja wykorzystuje nazwę aktualnie zalogowanego w systemie użytkownika.

Po uruchomieniu aplikacji i załadowaniu plików konfiguracyjnych, skrypt przechodzi do właściwej funkcjonalności, jaką jest periodyczne rejestrowanie i wysyłanie listy aktywnych procesów oraz wykonanych zrzutów ekranu. Użytkownik nie musi się przy tym martwić o takie problemy techniczne jak błąd połączenia z siecią, gdyż działanie aplikacji przewiduje takie sytuacje.

Przewidziane są także próby oszukania systemu. Jeśli użytkownik podałby nieprawidłowe dane w pliku konfiguracyjnym, to miałoby to odzwierciedlenie w aplikacji prowadzącego, w najgorszym wypadku poprzez nie ukazanie użytkownika na liście obecnych i nasłuchiwanych. Jest to wystarczające do podjęcia odpowiednich środków przez prowadzącego, dlatego w interesie użytkownika jest, aby aplikacja kliencka na jego komputerze z powodzeniem połączyła się z serwerem i przesyłała wszystkie niezbędne dane. Czyni to też zbędną potrzebę, aby aplikacja pracowała w tle i uruchamiała się przy starcie systemu.

Perspektywą na przyszły rozwój mogłoby być opracowanie graficznego interfejsu, który w wygodniejszy i bardziej odporny na błędny sposób pozwalałby na określenie konfiguracji aplikacji; a także zabezpieczenie kodu skryptu przed edycją przez użytkownika, np. poprzez kompilację do pliku binarnego.

Jako ostatnia zostanie przedstawiona aplikacja serwerowa. Aby uruchomić aplikację serwerową na serwerze należy przejść do katalogu, w którym znajdują się pliki serwerowe. Następnie należy uruchomić plik api.py za pomocą komendy sudo python3 api.py. Zostanie wtedy uruchomiona pełna aplikacja ze wszystkimi dostępnymi serwisami. Aby zmodyfikować adres IP oraz port, na którym zostaną uruchomione serwisy HTTP, należy

zmodyfikować plik config.py, w którym znajduje się konfiguracja frameworka cherrypy. Adres IP jest określony w słowniku CHERRYPY_CONFIG_DEFAULT w polach ‘server.socket_host’ oraz ‘tools.auth_basic.realm’. Aby zmienić adres IP należy zmodyfikować oba te pola i podać identyczny adres IP. W analogiczny sposób odbywa się zmiana portu, na którym uruchamia się serwer. Należy zmodyfikować pole ‘server.socket_port’ i zmienić obecną wartość na docelową.

10. Podsumowanie projektu oraz perspektywy dalszego rozwoju

Projekt *Sauron* wraz z całą fazą projektowania oraz implementacji przebiegł zgodnie z założonym harmonogramem. System do monitorowania urządzeń w sali laboratoryjnej może mieć swoje zastosowanie między innymi na terenie szkół oraz uczelni, wykorzystujących elektroniczne metody sprawdzania wiedzy swoich uczniów. Każdy z członków zespołu wywiązał się ze swoich obowiązków, o czym świadczy pełna implementacja w systemie kontroli wersji.

Jako rozwój projektu, można wskazać kilka kierunków. Aplikacja studenta mogłaby zostać zaopatrzona w interfejs graficzny oraz być bardziej odporna na błędny sposób konfiguracji. Rozwojem aplikacji prowadzącego mogłoby być rozbudowanie funkcjonalności związanych z optymalizacją przesyłania danych oraz bardziej przyjazny dla oka interfejs graficzny. Dodatkową funkcją mogłoby być dodanie funkcjonalności obsługi użytkowników prowadzących w samej aplikacji. W przypadku aplikacji serwerowej dobrym pomysłem byłoby dodanie funkcjonalności przesyłania plików audio lub wideo.

11. Bibliografia

Wykorzystane obrazy:

https://www.publicdomainpictures.net/en/view-image.php?image=251922&picture=medieval-tower

Dokumentacje:

Vuejs - https://vuejs.org/
Vuex - https://vuex.vuejs.org/
Python - https://www.python.org/doc/
Cherrypy - https://cherrypy.org
Psutil - https://psutil.readthedocs.io/en/latest/
MongoDB - https://university.mongodb.com/
MongoDB - https://api.mongodb.com/python/current/