Informatyka, Bezpieczeństwo systemów informatycznych
Wydział Elektryczny
Politechnika Poznańska

Podstawy teleinformatyki

LabEye

Wojciechowski Aleksander 126878
Sobieszczyk Mateusz 126874
Prusimski Michał Krzysztof 121379

Spis treści

1. Wstęp

Celem projektu LabEye było stworzenie aplikacji zdolnej do monitorowania pracowni laboratoryjnej. Aplikacja składa się z dwóch modułów: aplikacji prowadzącego zajęcia, przeznaczonej do wyświetlania zebranych informacji, oraz aplikacji agenta działającego w tle podczas pracy w trakcie zajęć.

Moduł agenta kompatybilny jest z wersjami systemu Windows 7 i wyższymi. Został on wyposażony w mechanizmy umożliwiające mu pozyskiwanie listy obecnie uruchomionych procesów na danym stanowisku oraz wysyłaniu powiadomień na podstawie listy niedozwolonych programów i stron internetowych.

Aplikacja prowadzącego przeznaczona jest na systemy Windows 7 i wyższe jednak planowane jest wsparcie dla Ubuntu. Umożliwia ona podgląd informacji zdobytych przez agenta oraz modyfikowanie list niedozwolonych stron internetowych i procesów.

Temat został wybrany z powodu chęci pomocy prowadzącym zajęcia laboratoryjne w weryfikacji studentów próbujących uzyskać ocenę pozytywną z przedmiotu za pomocą materiałów, które zostały uznane za niedozwolone. Dodatkowo część projektu pokrywa się z aplikacją, która obecnie jest rozwijana w ramach pracy inżynierskiej.

2. Podział prac

Projekt został zrealizowany dzięki współpracy trzech osób. Wszyscy brali czynny udział przy powstawaniu całości aplikacji, jednakże w celu usprawnienia prac i działania dla każdej osoby przydzielone zostało wykonanie pewnej części zadań.

Autor	Podzadanie
Mateusz Sobieszczyk	Obsługa połączenia między bazą danych, generowanie powiadomień oraz ich przesyłanie, rozpoznawanie użytkowników w aplikacji agenta, przesyłanie obrazu z aplikacji agenta do prowadzącego, projekt bazy danych
Aleksander Wojciechowski	Projekt i obsługa bazy danych, weryfikacja uruchomionych procesów i stron internetowych, wyświetlanie pozyskanych informacji i powiadomień w aplikacji prowadzącego, obsługa czarnych list
Michał Prusimski	Pozyskiwanie informacji o uruchomionych w systemie procesach oraz odwiedzanych przez użytkownika stronach internetowych, weryfikacja z czarną listą aplikacji i stron internetowych, generowanie powiadomień

Tabela 1: Podział prac

3. Funkcjonalności

Zostały rozdzielone pomiędzy funkcjonalne i niefunkcjonalne. W zrealizowanym projekcie są to następujące wymogi:

Funkcjonalne:

- System składa się z dwóch aplikacji: “Agent”, “Prowadzący”.
- Aplikacja agenta możliwa jest do wyłączenia jedynie w ramach zamykania systemu operacyjnego. Student w przeciwieństwie do prowadzącego nie ma możliwości zamknięcia programu.
- Istnieją dwie podstawowe role: Prowadzący oraz Student.
- Agent jest domyślnie zminimalizowany do paska powiadomień.
- Prowadzący domyślnie przypisuje nazwę hosta do stanowiska, zmiana użytkownika pracującego przy danym komputerze jest możliwa wyłącznie przez prowadzącego.
- Prowadzący poza danymi zbieranymi na bieżąco ma możliwość zarządzania czarnymi listami aplikacji oraz stron internetowych
- Aplikacja prowadzącego umożliwia na usunięcie rekordu z bazy danych dotyczącego konkretnego studenta
- Prowadzący ma możliwość przeglądania logów (zdjęć pulpitu)

Niefunkcjonalne:

- Aplikacje agentów dane umieszczają w bazie danych.
- Aplikacja prowadzącego pobiera dane z bazy.
- Informacje przesyłane są cyklicznie do bazy danych, a następnie za pomocą żądania, bądź cyklicznie, są przesyłane do aplikacji administratora. Wyjątkiem są logi w postaci zdjęć pulpitów. Są one wysyłane bezpośrednio do administratora.
- Przechowuje dane w bazie danych która powinna znajdować się w sieci lokalnej.
- Wszelkie aspekty prawne regulować prowadzący.

Następujące dane będą gromadzone:

- Uruchomione aplikacje i procesy
- Wygenerowane powiadomienia oraz ich logi

4. Funkcjonalności

Użyte narzędzia oraz środowiska:

- Visual Studio Enterprise 2017 – potężne środowisko programistyczne, które w znacznym stopniu ułatwia prace związane z tworzeniem oprogramowania. Zawiera IDE Microsoft Visual C# oraz designer przeznaczony do tworzenia aplikacji z graficznym interfejsem użytkownika WPF czy WinForms
    - Język programowania: C# – obiektowość, garbage collector, wsparcie dla ogromnej ilości tworzonych bibliotek
    - XAML – głównie używany do opisu interfejsu użytkownika, ułatwia na przełamanie bariery między surowym kodem, a projektowaniem interfejsu.
    - WinForms – wykorzystywane do projektu aplikacji prowadzącego. Wybrane głównie ze względu na możliwość wydania aplikacji dla systemów Ubuntu
- Serwer bazy danych MongoDB – wybrane ze względu na brak ściśle zdefiniowanej struktury, dokumenty w stylu JSON, zapytania do zagnieżdżonych pól dokumentów, łatwa replikacja.
- Github – jeden z najpopularniejszych serwisów stworzonym z myślą o przechowywaniu repozytoriów projektów, wspomaganiu pracy zespołowych czy przeglądaniu wprowadzonych zmian przez poszczególnych użytkowników. Wybrany głównie ze względu na swoją prostotę oraz popularność.

Biblioteki jakie wykorzystano w projekcie to:

- VncSharp - zdalny dostęp do komputera użytkownika.

5. Architektura rozwiązania

LabEye wykorzystuje wielu agentów zbierających informacje z poszczególnych stanowisk i przekazuje je do bazy danych MongoDB lub bezpośrednio do aplikacji prowadzącego jeśli są to logi.

5.1. Opis ogólny

Przed rozpoczęciem pracy agentów prowadzący zobligowany jest do odpowiedniego skonfigurowania stanowisk. Powinien wskazać ścieżkę, która zezwoli na połączenie z przygotowaną bazą oraz nadać nazwę hosta dla danego stanowiska.

Po wykonaniu wstępnej konfiguracji na stacjach roboczych studentów, aplikacje są gotowe do rozpoczęcia zbierania informacji w systemie. Student zostanie poproszony o podanie identyfikatora, za pomocą którego będzie mógł być zweryfikowany. Jest to wymagane do kontynuowania pracy na danym stanowisku.

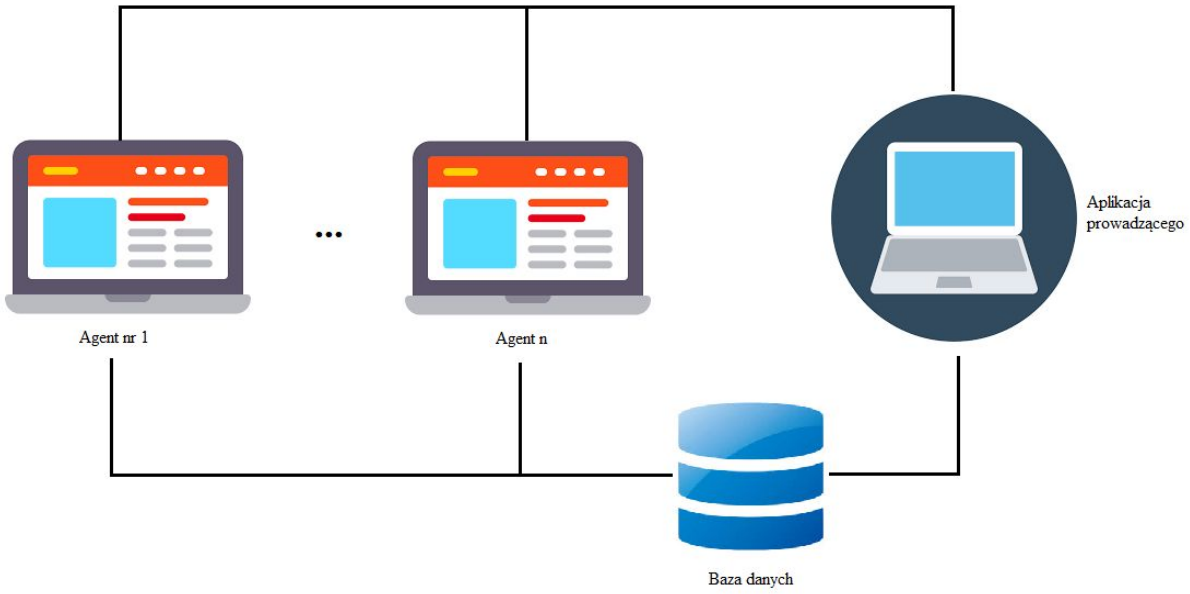
Kolejnym etapem jest skonfigurowanie aplikacji przeznaczonej dla prowadzącego. Jeżeli po jej uruchomieniu nie zostanie zlokalizowany plik konfiguracyjny, użytkownik zostanie poproszony o podanie niezbędnych danych, które umożliwią dalsze działanie aplikacji. W tym celu wymagane są:

- **Ścieżka MongoDb** – za jej pomocą następuje konfiguracja oraz późniejsze ustanawianie połączenia z bazą danych. W momencie gdy jest ona błędna aplikacja wyświetli komunikat o błędzie, a użytkownik zmuszony będzie do poprawienia parametru
- **Nazwa bazy danych** – określa nazwę bazy danych, z którą chcemy uzyskać połączenie. W przypadku gdy dana baza nie istnieje zostanie wyświetlony komunikat o błędzie, a użytkownik zmuszony będzie do poprawienia parametru

Następnie prowadzący, za pomocą skonfigurowanej już aplikacji, cyklicznie bądź na żądanie pobiera uzyskane informacje z bazy danych. Od tego momentu prowadzący posiada pełen dostęp do bazy danych, zawierającej informacje o nazwie hosta, imieniu i nazwisku, adresie IP oraz powiadomień wygenerowanych podczas pracy studenta na danym stanowisku. Oprócz tych funkcjonalności możliwa jest też aktualizacja czarnych list.

Aplikacja prowadzącego oferuje również możliwość przejęcia ekranu użytkownika. W tym celu ustanawiane jest, za pośrednictwem informacji znajdujących się w bazie danych, bezpośredniego połączenia między wybraną stacją roboczą, na której działa agent, a interfejsem prowadzącego.

Opisana architektura przedstawiona jest na *Rysunku 1*.



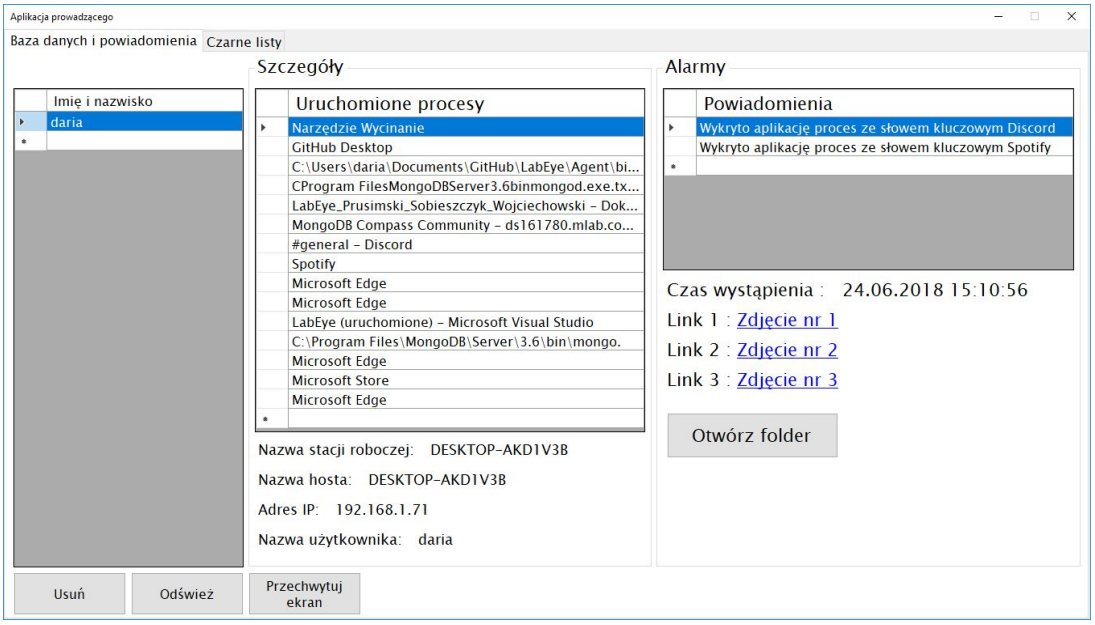
Rysunek 1: Schemat architektury systemu

5.2. Interfejs aplikacji prowadzącego

Interfejs aplikacji wykonany został za pomocą biblioteki WinForms. Całość oparta jest o gridy zawierające informacje pozyskiwane z bazy danych oraz przyciski służące do wykonywania akcji takich jak usuwanie istniejących wpisów czy pozyskiwanie nowych

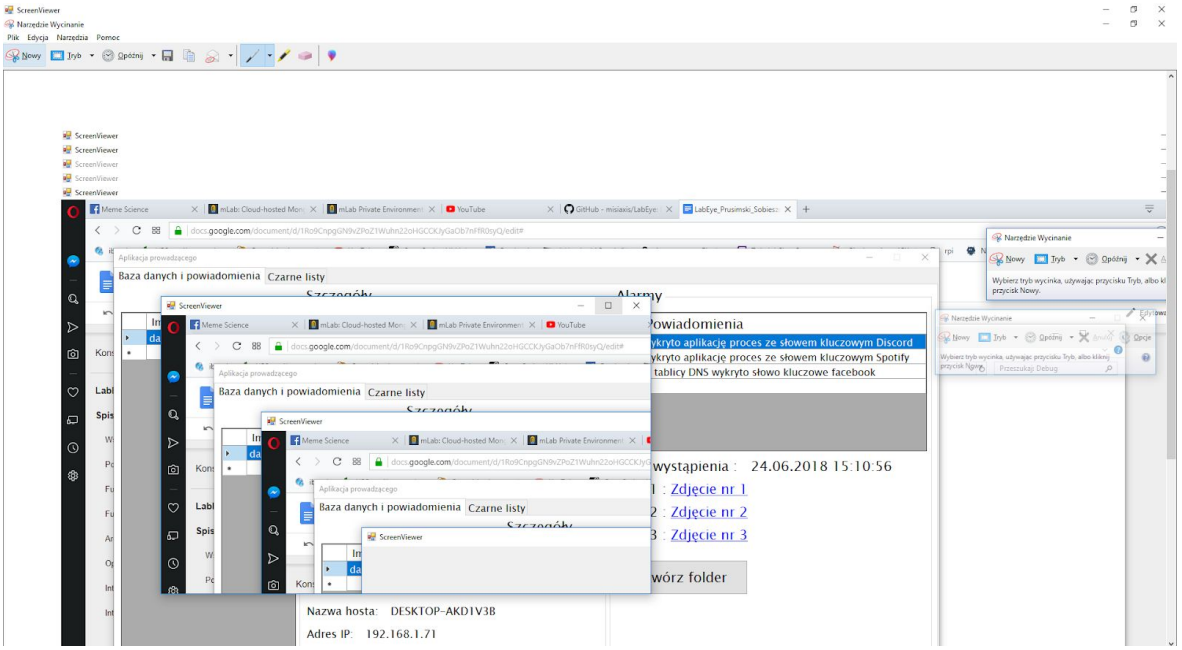
5.2.1. Wyświetlanie zawartości bazy danych i powiadomień

Na *Rysunku 2* przedstawiony został interfejs aplikacji prowadzącego odpowiedzialny za zarządzanie informacjami otrzymywanymi z bazy danych. W pierwszej tabeli znajdują się imienia i nazwiska studentów, którzy obecnie pracują na swoich stanowiskach wykorzystujących agenta LabEye. Prowadzący ma możliwość usunięcia rekordu zawierającego wpis o danym studencie. W takim przypadku z bazy danych zostaną usunięte informacje o wszystkich obecnie uruchomionych procesach, wszystkie szczegóły dotyczące danej stacji roboczej oraz wszystkie alarmy związane z danym użytkownikiem. Usunięte zostaną również wszystkie zdjęcia znajdujące się w bazie danych. Te które zostały pobrane zostaną zachowane.



Rysunek 2: interfejs aplikacji administratora, zakładka bazy danych i powiadomień

W części *Szczegóły* znajduje się lista wszystkich obecnie uruchomionych procesów na danej stacji roboczej, na której pracuje konkretny student. Dodatkowo wyświetlana jest nazwa stacji, nazwa hosta, adres IP oraz nazwa zalogowanego użytkownika. Wartości tych nie można edytować.



Rysunek 3: Przechwycony ekran

Za pomocą przycisku "*Przechwytuj ekran*" prowadzący ma możliwość przechwycenia obrazu z komputera, na którym znajduje się agent LabEye. Domyślna rozdzielczość jest identyczna co na komputerze, z którego obraz jest pozyskiwany. Po kliknięciu przycisku zostanie wyświetlone nowe okno, a w nim już przechwycony ekran przedstawione jest to na *Rysunku 3*.

Ze względów na wydajność działania oraz wymagane zasoby ekran odświeżany jest z częstotliwością jednej sekundy. Zwiększanie wartości powodowało znaczne zwiększenie wymagań oraz zapotrzebowania zasobów przez aplikację. Prowadzący może uruchomić wiele ekranów na raz, jednakże związane to jest ze zwiększeniem zużycia zasobów.

Użytkownik Agenta LabEye nie zostanie poinformowany o byciu podglądanym przez prowadzącego. Takie podejście pozwala na monitorowanie pracy studenta w czasie "prawie rzeczywistym" oraz formułowanie uwag, wskazówek czy ostrzeżeń w kierunku osoby, która obecnie jest monitorowana.

Alarmy zawierają wszystkie powiadomienia, które zostały wygenerowane na używanej przez studenta stacji. Wyróżnia się dwa typy powiadomień. Związane z odwiedzeniem niedozwolonej strony internetowej oraz z wystąpieniem użycia programu uznanego za niezaakceptowany przez prowadzącego. Każde powiadomienie posiada czas jego wystąpienia oraz trzy zdjęcia służące jako dowód złamania zasad.

Zdjęcia są przechowywane w bazie danych i automatycznie pobierane na komputer użytkownika. Zapisywane są one w ścieżce folderów, która znajduje się w głównym folderze aplikacji, a jej schemat to :

<Nazwa użytkownika>\<Treść alarmu>\<dd_mm_rr_hh_msms_<numer porządkowy>>.jpg

<Nazwa użytkownika>	Określa nazwę użytkownika danej stacji roboczej w formie : Imię i Nazwisko
<Treść alarmu>	Rozróżniane są dwa rodzaje alarmów : "Wykryto aplikację proces ze słowem kluczowym *<Nazwa aplikacji>*", gdzie *<Nazwa aplikacji>* jest nazwą aplikacji, która została uruchomiona w trakcie działania Agenta na stacji roboczej i została przez niego zdefiniowana jako niedozwolona oraz "*W tablicy DNS wykryto słowo kluczowe <zakazana witryna>*", gdzie *<zakazana witryna>* jest stroną internetową, która podczas pracy Agenta została wykryta jako jedna ze stron znajdujących się na czarnej liście
dd	Określa dzień, w którym nastąpiło wystąpienie danego alarmu
mm	Określa miesiąc, w którym nastąpiło wystąpienie danego alarmu
rrrr	Określa rok , w którym nastąpiło wystąpienie danego alarmu
hh	Przedstawia godzinę, w formacie dwudziestoczterogodzinnym, wystąpienia alarmu
msms	Określa informacje o minucie danej godziny, w której wystąpił alarm
<numer porządkowy>	Wprowadzony w celu powiązania ze sobą dowodów, w postaci zdjęć, które definiują wystąpienie tego samego alarmu. Może przyjmować tylko wartości "*1*", "*2*", "*3*", oznaczające kolejno zdjęcie wykonane jako pierwsze, jako drugie i jako trzecie

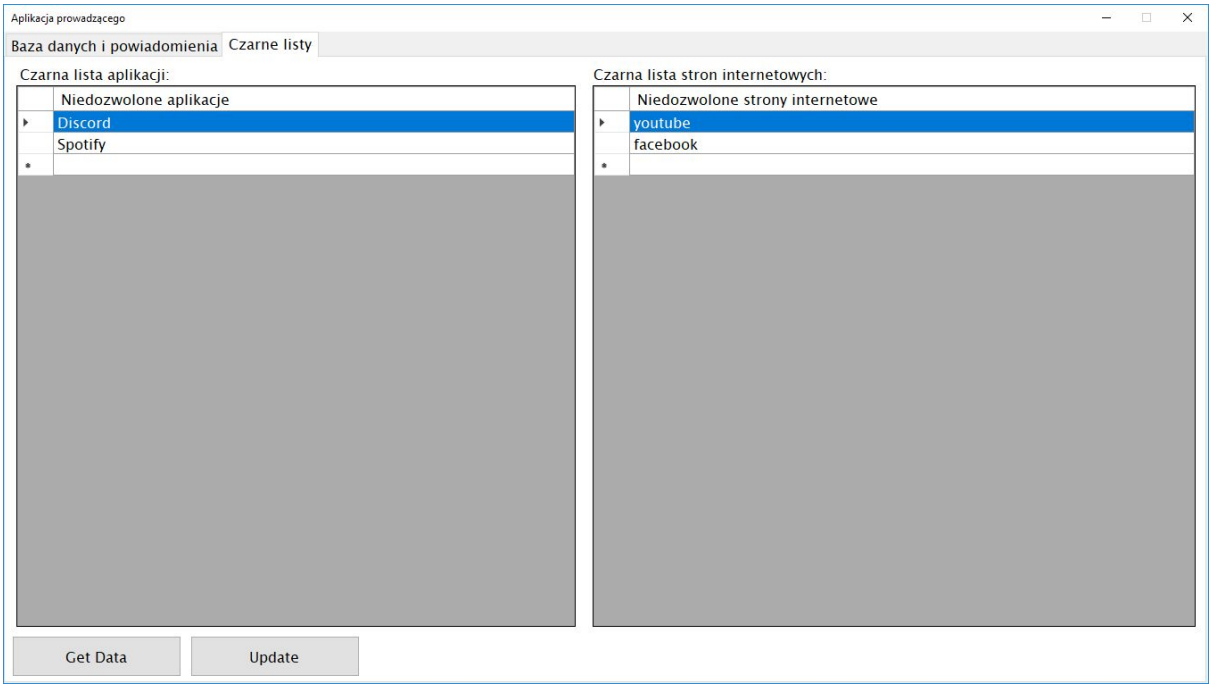
Tabela 2: Objaśnienie roli poszczególnych pól w schemacie ścieżki folderów służącej do przechowywania zdjęć

W *Tabeli 2* zostały wyjaśnione poszczególne pola związane ze schematem ścieżki folderów wykorzystywanej do przechowywania dowodów w formie zdjęć. Każdy użytkownik posiada swój osobny folder, w którym przechowywane są informacje dotyczące alarmów wygenerowanych przez danego klienta Agenta LabEye.

5.2.2. Czarne listy

W tym widoku prowadzący ma możliwość edytowania czarnych list aplikacji i stron internetowych. W tym celu wystarczy kliknąć na obecny rekord by go edytować lub kliknąć klawisz delete aby go usunąć.

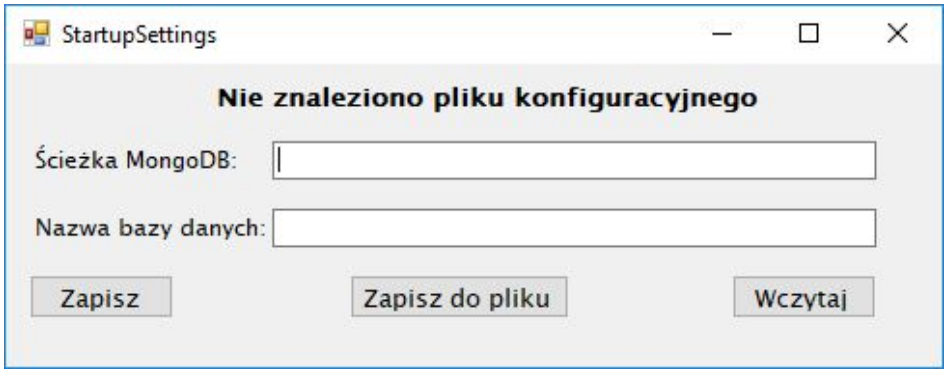
Dodawanie realizowane jest poprzez kliknięcie na wolne pole i rozpoczęcie pisania. Wszystkie zmiany zatwierdzane są enterem, a zaktualizowane listy zostają przesłane do bazy poprzez kliknięcie *Update*.



Rysunek 4: interfejs aplikacji administratora, zakładka czarnych list

5.2.3. Okno konfiguracji

W momencie gdy nie zostanie zlokalizowany plik konfiguracyjny zostanie wyświetlone okno konfiguracji aplikacji prowadzącego (przedstawione na *Rysunku 5*), w którym użytkownik będzie zobowiązany do podania informacji opisanych w *Tabeli 3*.



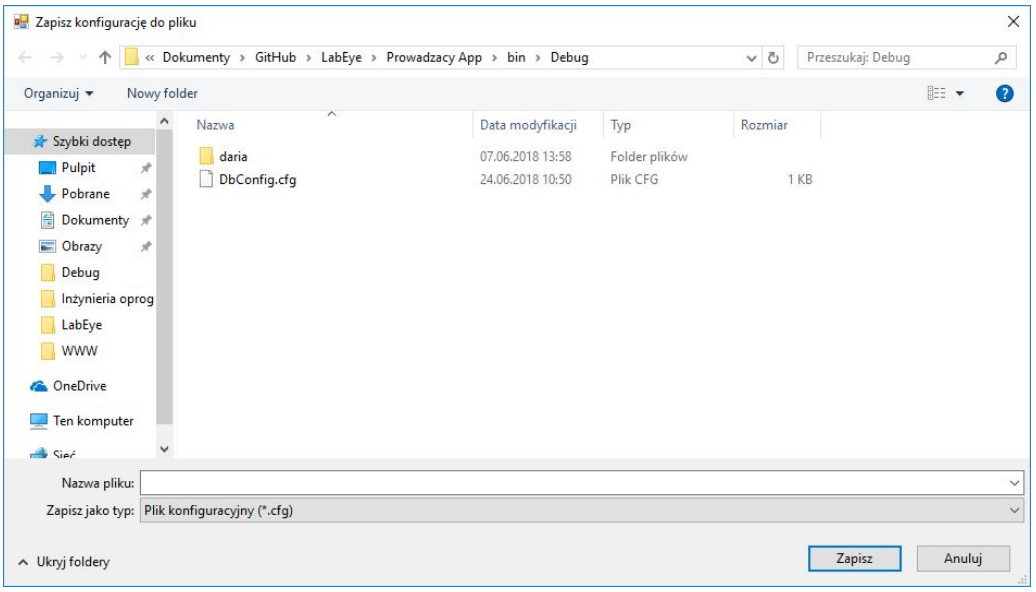
Rysunek 5: Okno konfiguracji aplikacji prowadzącego

Ścieżka MongoDB	Wiersz definiujący parametry wykorzystywane do nawiązania połączenia się z serwerem bazy danych MongoDB. Przykładowy wiersz połączenia : *"mongodb://admin:abc123@ds11111.mlab.com:30300/lab_eye_mongo"*
Nazwa bazy danych	Określa nazwę bazy danych, z którą użytkownik pragnie się połączyć i pozyskiwać, oraz do której chce wysyłać informacje

Tabela 3: Objaśnienie parametrów konfiguracji

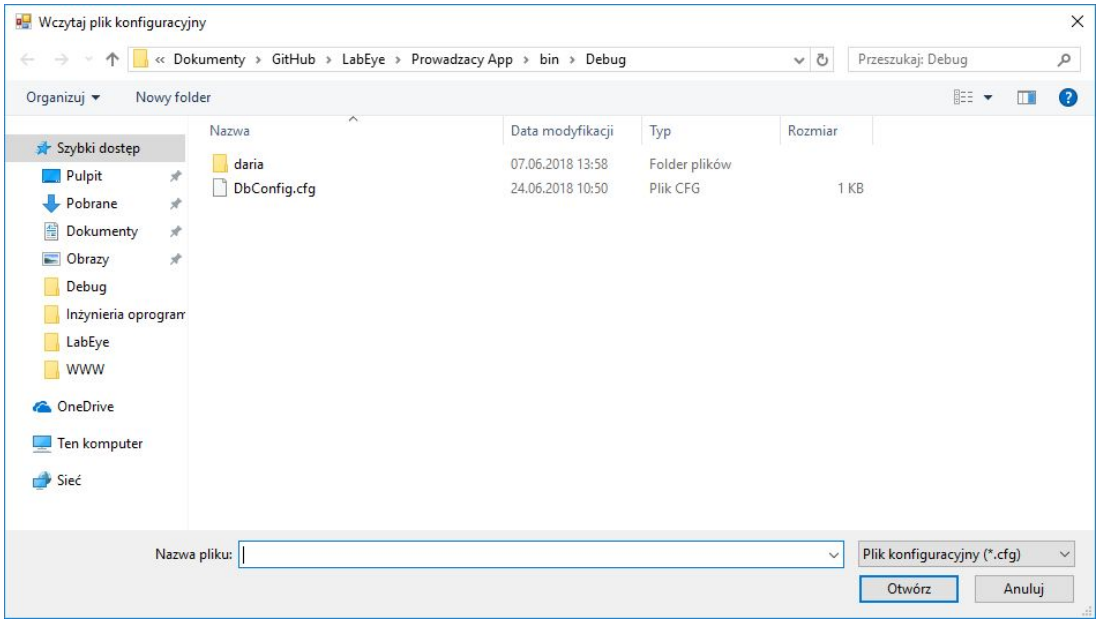
W przypadku pominięcia konfiguracji poprzez nie podanie poprawnych wartości opisanych w *Tabeli 3* aplikacja zgłosi błąd, a użytkownik zmuszony będzie do dokonania poprawek.

Konfiguracja może zostać zapisana do pliku i wykorzystywana jako szablon. Przycisk "*Zapisz*" służy do wprowadzenia parametrów do pliku, o ile istnieje, oraz bezpośrednio do aplikacji.



Rysunek 6: Okno wykorzystywana do wskazania lokalizacji oraz nazwy zapisu pliku konfiguracyjnego

"*Zapisz do pliku*" spowoduje utworzenie nowego pliku konfiguracyjnego lub nadpisanie już istniejącego, co pozwoli na późniejsze wykorzystanie go jako szablon. W tym celu otworzone zostanie okno pozwalające na wskazanie lokalizacji zapisu szablonu konfiguracyjnego domyślnym typem jest "**.cfg*". Przykładowy wygląd takiego okna przedstawiony został na *Rysunku 6*.



Rysunek 7: Okno wykorzystywane do wczytywania pliku konfiguracyjnego

Po kliknięciu "*Wczytaj*" zostanie otworzone okno (*Rysunek 7*), w którym użytkownik będzie mógł wskazać plik konfiguracyjny, z którego zostaną pobrane parametry konfiguracyjne. Domyślnym typem pliku konfiguracyjnego jest "**.cfg*".

5.3. Interfejs Agenta

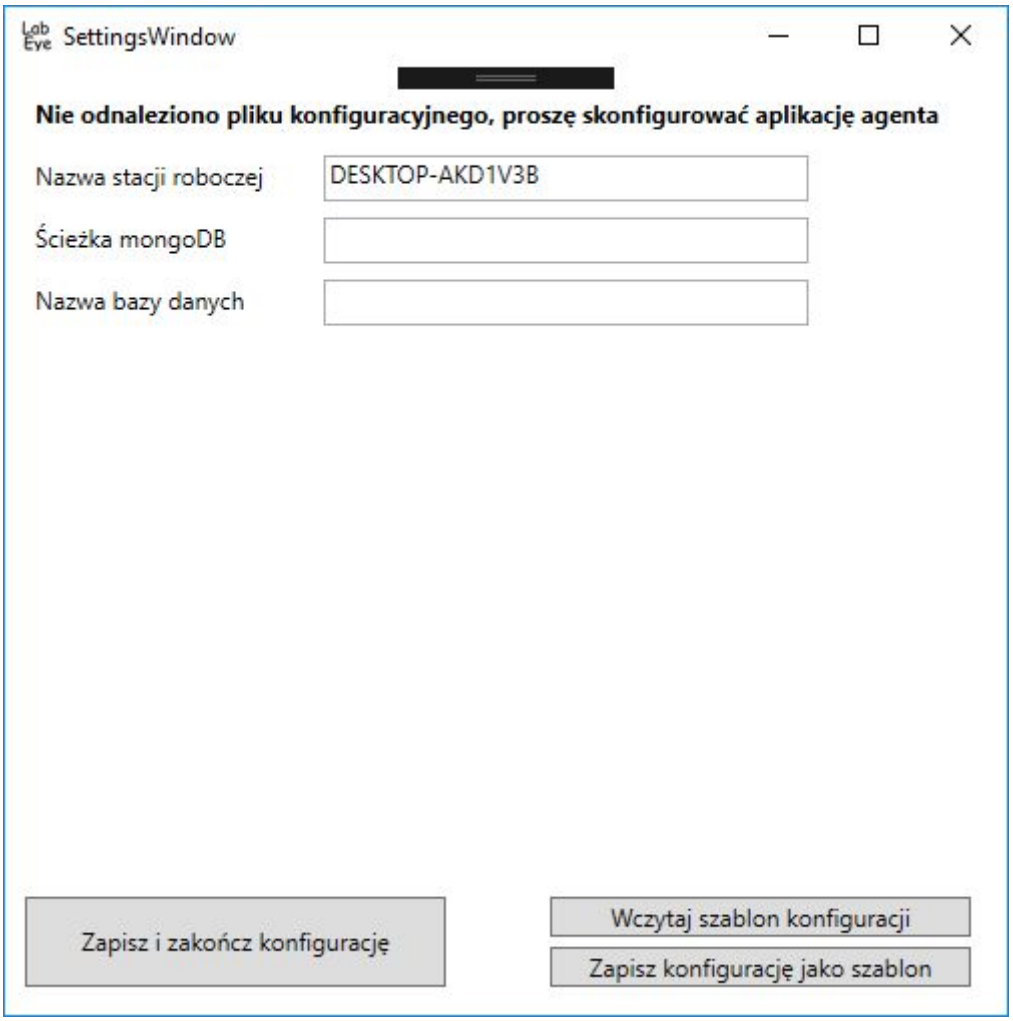
Na *rysunku 8* przedstawione jest okno, które wyświetli się zaraz po uruchomieniu systemu operacyjnego. Okno przysłoni cały ekran i jedyną możliwością kontynuacji będzie podanie identyfikatora studenta.

Rysunek 8: okno wpisywania identyfikatora studenta

W momencie gdy student poda swój identyfikator aplikacja minimalizuje się do traya i dostępna jest jedynie ikona z logiem *LabEye*. Kliknięcie prawym przyciskiem myszy na ikonę skutkuje otwarciem menu kontekstowego zawierającego następujące opcje:

- *Aplikacja zablokowana / odblokowana*
  - pole o zmiennej wartości.Służy do wyświetlania stanu aplikacji agenta LabEye

- *Zakończ*
    - pole pozwalające prowadzącemu, po podaniu odpowiedniego hasła, na zakończenie pracy agenta
- *Odblokuj*
    - pole służące do odblokowywania aplikacji. Po wybraniu tej opcji wyświetli się okienko (*rysunek 9),* w którym wymagane będzie podanie hasła odblokowującego aplikację. Od tego momentu można swobodnie korzystać ze wszystkich pozostałych opcji.
- *Ustawienia*
    - wybranie *ustawień* spowoduje wyświetlenie konfiguratora aplikacji agenta. Możliwe opcje do edycji to nazwa stacji roboczej, ścieżka do bazy danych MongoDB oraz hasło służące odblokowywaniu aplikacji.



Rysunek 9: Okno konfiguracji aplikacji agenta

Na *Rysunku 6* ukazane jest okno konfiguracji, które uruchomi się w przypadku gdy *Agent* nie znajdzie pliku konfiguracyjnego. W takim wypadku po podaniu parametrów konfiguracyjnych, opisanych w *Tabeli 4* zostanie wygenerowany odpowiedni plik. Konsekwencją kliknięcia przycisku "*Zapisz i zakończ konfigurację*" będzie zapisanie w pamięci aplikacji agenta informacji konfiguracyjnych wykorzystywanych w celu połączenia z serwerem bazy danych oraz walidacji stacji roboczej. "*Wczytaj szablon konfiguracji*" otworzy okno identyczne do tego przedstawionego na *Rysunku 7*. Sytuacja jest taka sama w

przypadku "*Zapisz konfigurację jako szablon*". Należy pamiętać, że pliki konfiguracyjne aplikacji Agenta oraz aplikacji Prowadzącego nie są takie same. Wykorzystanie niepoprawnego szablonu może prowadzić do błędów przy pozyskiwaniu parametrów konfiguracyjnych z pliku.

Ścieżka MongoDB	Wiersz definiujący parametry wykorzystywane do nawiązania połączenia się z serwerem bazy danych MongoDB. Przykładowy wiersz połączenia : *"mongodb://admin:abc123@ds11111.mlab.com:30300/lab_eye_mongo"*
Nazwa bazy danych	Określa nazwę bazy danych, z którą użytkownik pragnie się połączyć i pozyskiwać, oraz do której chce wysyłać informacje
Nazwa stacji roboczej	Nazwa dowolna, służy w celu personalizacji nazewnictwa stacji roboczych, pozwalając na łatwiejsze skojarzenie poszczególnych miejsc pracy z ich faktycznym położeniem w świecie rzeczywistym
Hasło administratora	Hasło ustalane przez prowadzącego zajęcia. Wykorzystywane do odblokowywania funkcjonalności aplikacji Agenta przeznaczonych tylko i wyłącznie dla administratora systemu. Opcje te zostały szczegółowo opisane we wcześniejszej części rozdziału

Tabela 4: Parametry konfiguracyjne aplikacji Agenta

5.4. Baza danych

LabEye wykorzystuje bazę danych MongoDB. Jest ona bazą danych wysoce skalowalną i elastyczną. MongoDB przechowuje dane w dokumentach podobnych budową do JSON. Modele dokumentów odwzorowują obiekty w kodzie aplikacji co znacznie ułatwia pracę z danymi. Jest to baza rozproszona, co w głównej mierze oznacza jej wysoką dostępność. MongoDB jest darmowa i typu open-source, opublikowana na licencji GNU Affero General Public License. Dodatkowo posiada bardzo przejrzystą i czytelną dokumentację, opartą o proste przykłady jak i propozycje rozwiązań trudniejszych problemów. W celu implementacji wykorzystania MongoDB w LabEye użyty został sterownik .NET w wersji 2.6.1.

Zaawansowana obsługa bazy danych może odbywać się z poziomu konsoli systemu za pomocą odpowiednich funkcji, bądź przy użyciu predefiniowanych programów służących do mniej wymagających akcji takich jak modyfikacja, dodawanie czy usuwanie dokumentów.

5.4.1. Zawartość bazy danych

Baza danych aplikacji LabEye składa się z pięciu kolekcji, z czego dwie z nich są utworzone przez bibliotekę *GridFS* odpowiedzialną za wysyłanie i pobieranie zdjęć z bazy danych. Na *Rysunku 10* został przedstawiony wygląd zawartości za pomocą programu *MongoDB Compass Community*

Collection Name ▲	Documents	Avg. Document Size	Total Document Size
AlertsHistory	4	496.0 B	1.9 KB
BlackList	1	112.0 B	112.0 B
Workstations	1	2.0 KB	2.0 KB
fs.chunks	30	243.2 KB	7.1 MB
fs.files	18	240.0 B	4.2 KB

Rysunek 10: Zawartość bazy danych LabEye

- **Collection Name**
  - Określa nazwę danej kolekcji w bazie danych LabEye. Jak widać na *Rysunku 10* zawarte jest pięć kolekcji:
    - **Workstations**
      - Główna kolekcja bazy danych LabEye zawierająca niezbędne informacje dotyczące stacji roboczych tj. imię i nazwisko użytkownika czy nazwa hosta, na których pracuje aplikacja Agenta
    - **BlackList**
      - Jest to kolekcja wykorzystywana przez aplikację agenta LabEye do pozyskiwania informacji o stronach internetowych i aplikacjach, które zostały uznane przez prowadzącego za niestosowne czy niedozwolone w trakcie pracy na laboratorium i w efekcie wpisane na jedną z czarnych list za pomocą aplikacji prowadzącego
    - **AlertsHistory**
      - W tej kolekcji zawarta jest historia wszystkich alarmów, które wystąpiły podczas pracy wszystkich zainstalowanych aplikacji agentów LabEye na dostępnych stacjach roboczych

        - **fs.files**
            - Każdy plik, który zostanie wysłany do bazy danych przy pomocy biblioteki *GridFS* posiada dokument w tej kolekcji. Zawiera informacje o przechowywanym pliku oraz jest ściśle połączony z **fs.chunks**
        - **fs.chunks**
            - W momencie gdy plik jest wysyłany za pomocą *GridFS* jest on rozbijany na mniejsze kawałki, a te są kolejno przesyłane do bazy danych. Kiedy jednak plik jest pobierany, poszczególne kawałki są składane na nowo, dzięki czemu oryginalny plik jest na nowo dostępny. **fs.chunks** przechowuje te kawałki
- **Documents**
    - Pozwala w bardzo szybki sposób określić ilość dokumentów znajdujących się w danej kolekcji
- **Avg. Document Size**
    - Pole przechowujące informację określającą średni rozmiar pojedynczego dokumentu znajdującego się w kolekcji
- **Total Document Size**
    - Pole przechowujące informację określającą całkowity rozmiar wszystkich dokumentów znajdujących się w kolekcji

5.4.2. Zawartość kolekcji Workstation

Kolekcja *Workstation* odpowiada za główne informacje dotyczące stacji roboczych i jak i pracujących na przy nich studentach. Na niej opiera się działanie całości projektu LabEye. Na *Rysunku 11* przedstawiony został przykładowy dokument jaki może znajdować się w kolekcji *Workstation*.

Dokument składa się z wielu pól, w celu rozjaśnienia ich znaczeń poniżej zawarte są ich krótkie i zwięzłe opisy:

- **_id**
    - Zawiera informacje dotyczące ID danego dokumentu. Każdy z nich posiada unikatowy identyfikator, który przydzielany jest automatycznie w momencie utworzenia wpisu w bazie danych aplikacji LabEye.
- **WorkstationName**
    - W tym polu znajduje się informacja odnośnie nazwy stacji roboczej, która została nadana podczas konfiguracji aplikacji agenta LabEye. Jest ona wartością unikatową, ponieważ za jej pomocą wykonywana jest większość operacji aplikacji LabEye. Służy do rozróżniania poszczególnych stacji roboczych znajdujących się w laboratorium.
- **StudentFirstAndLastName**
    - Pole przechowujące informacje na temat imienia i nazwiska studenta pracującego przy konkretnej stacji roboczej. Dane te są podawane przy starcie aplikacji agenta LabEye i mogą być zmienione w trakcie działania programu tylko i wyłącznie przez prowadzącego zajęcia

- **HostName**
    - Wartość **HostName** określa nazwę hosta jaka została nadana podczas konfiguracji systemu operacyjnego. Ta wartość pobierana jest automatycznie poprzez aplikację agenta LabEye.

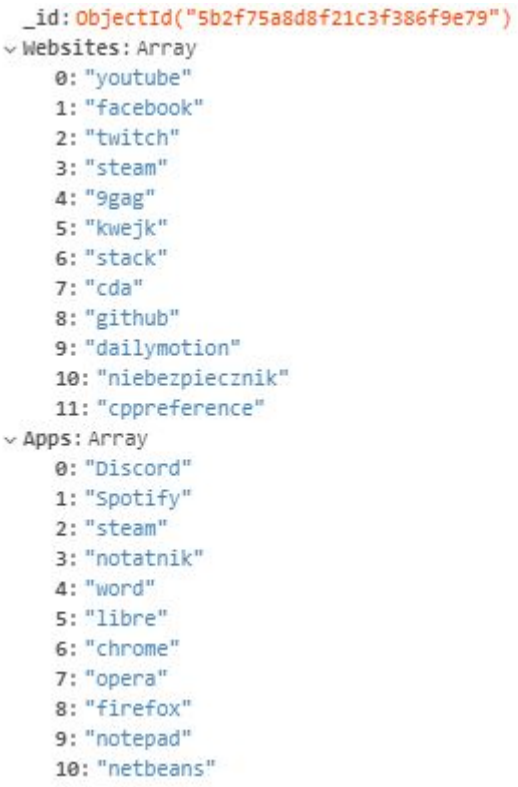

```
_id: ObjectId("5b2f985cd8f21c3a2023609e")
WorkstationName: "DESKTOP-AKD1V3B"
StudentFirstAndLastName: "daria"
HostName: "DESKTOP-AKD1V3B"
IPAdress: "192.168.1.71"
UserName: "daria"
Alerts: Array
  0: Object
      AddDate: "24.06.2018 18:57:46"
      StudentFirstAndLastName: "daria"
      AlertName: "Wykryto aplikację proces ze słowem kluczowym Discord"
      Link1: ObjectId("5b2fcd87d8f21c160406d44f")
      Link2: ObjectId("5b2fcd88d8f21c160406d451")
      Link3: ObjectId("5b2fcd88d8f21c160406d453")
  1: Object
      AddDate: "24.06.2018 18:57:46"
      StudentFirstAndLastName: "daria"
      AlertName: "Wykryto aplikację proces ze słowem kluczowym Spotify"
      Link1: ObjectId("5b2fcd89d8f21c160406d455")
      Link2: ObjectId("5b2fcd89d8f21c160406d457")
      Link3: ObjectId("5b2fcd89d8f21c160406d459")
  2: Object
      AddDate: "24.06.2018 19:00:02"
      StudentFirstAndLastName: "daria"
      AlertName: "W tablicy DNS wykryto słowo kluczowe facebook"
      Link1: ObjectId("5b2fce11d8f21c160406d45b")
      Link2: ObjectId("5b2fce11d8f21c160406d45d")
      Link3: ObjectId("5b2fce12d8f21c160406d45f")
Apps: Array
    0: "Narzędzie Wycinanie"
    1: "GitHub Desktop"
    2: "StartupSettings"
    3: "C:\Users\daria\Documents\GitHub\LabEye\Agent\bin\Debug\Workstation.cfg..."
    4: "CProgram FilesMongoDBServer3.6binmongod.exe.txt — Notatnik"
    5: "LabEye_Prusimski_Sobieszczyk_Wojciechowski - Dokumenty Google – Opera"
    6: "MongoDB Compass Community - ds161780.mlab.com:61780/lab_eye_mongo.Aler..."
    7: "#general - Discord"
    8: "Spotify"
    9: "Microsoft Edge"
    10: "Microsoft Edge"
    11: "LabEye (uruchomione) - Microsoft Visual Studio "
    12: "C:\Program Files\MongoDB\Server\3.6\bin\mongo.exe"
    13: "Microsoft Edge"
    14: "Microsoft Store"
    15: "Microsoft Edge"
```


Rysunek 11: Przykładowy wygląd dokumentu z kolekcji Workstation w bazie danych aplikacji LabEye

- **IPAdress**
    - Pole to przechowuje adres IPv4 stacji roboczej, który jest automatycznie pozyskiwany przez aplikację agenta LabEye. Adres IP może następnie zostać wykorzystany do ustanowienia połączenia w celu przechwytywania ekranu, z określonej stacji roboczej, za pomocą aplikacji administratora LabEye. Pola tego nie można edytować
- **UserName**
    - **UserName** jest kolejnym polem pozyskiwanym automatycznie. Przechowuje ono informacje o nazwie pod jaką do systemu zalogował się użytkownik. W przypadku nazwy domenowej *@student.put.poznan.pl* wykorzystywana jest separacja, a za separator służy znak @, zaś do pola wpisywana jest wartość przed znakiem separacji
- **Alerts**
    - Jest to lista dowolnej długości zawierająca dokumenty opisujące alarmy, które wystąpiły podczas pracy aplikacji agenta LabEye na stacji roboczej, przy której znajduje się student
        - **AddDate**
            - Przechowuje informacje o dacie i czasie wystąpienia danego alarmu. Format daty to *dd.mm.rrrr*, a format godziny to *hh:mm:ss*
        - **StudentFirstAndLastName**
            - Opisane wyżej, wykorzystywane w celach porządkowych w kolekcji **AlertsHistory**
        - **AlertName**
            - Pole zawierające szczegółowy opis dotyczący poszczególnego alarmu. Wyróżniane są dwa rodzaje alarmów: “Wykryto aplikację proces ze słowem kluczowym <*Nazwa aplikacji*>”, gdzie <*Nazwa aplikacji*> jest nazwą aplikacji pozyskiwaną z pola tytułowego okna aplikacji, która została uruchomiona w trakcie działania Agenta na stacji roboczej i została przez niego zdefiniowana jako niedozwolona oraz “*W tablicy DNS wykryto słowo kluczowe <zakazana witryna>*”, gdzie <*zakazana witryna*> jest stroną internetową, pozyskaną z tablicy DNS, która podczas pracy Agenta została wykryta jako jedna ze stron znajdujących się na czarnej liście. Za pomocą tego pola klasyfikowane są zdjęcia pobierane przez aplikacje prowadzącego LabEye
        - **Link1, Link2 oraz Link3**
            - Pola te przechowują informacje o plikach zdjęć utworzonych w momencie wystąpienia alarmu na określonej stacji roboczej. Pole to jest typu *ObjectId* i wskazuje na plik znajdujący się w kolekcji **fs.files**, za pomocą którego mogą zostać pozyskane pliki z bazy danych LabEye
- **Apps**
    - Jest to pole przechowujące listę odpowiednio przefiltrowanych procesów, które obecnie są uruchomione na danej stacji roboczej. Aplikacja agenta LabEye pozyskuje nazwę procesu z tytułu jej okna

5.4.3. Zawartość kolekcji BlackList

Kolekcja *BlackList* jest bardzo prosta w budowe. *Rysunek 12* prezentuje jej koncepcje oraz przykładowy dokument.



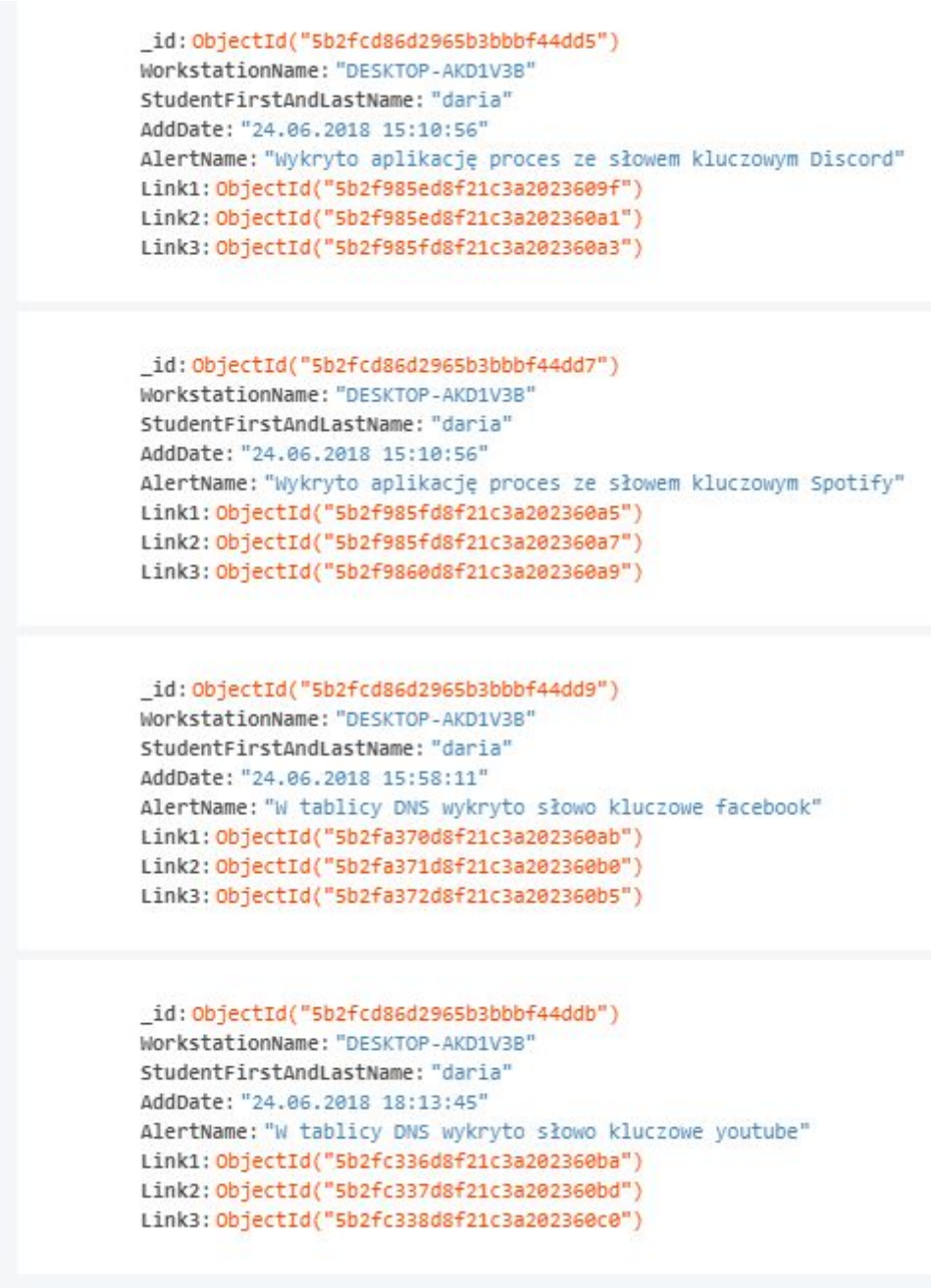
Rysunek 12: Zawartość kolekcji BlackList

W kolekcji BlackList znajduje się tylko jeden dokument przechowujący dwie listy służące do weryfikacji wystąpień alertów na stacjach roboczych:

- **Apps**
  - Jest to lista aplikacji niedozwolonych przez prowadzącego w czasie pracy laboratorium. Nazwy podawane są bez znaków specjalnych, w przypadku podania takiego znaku aplikacja prowadzącego LabEye zwróci błąd
- **Websites**
  - Pole to przechowuje listę stron zakazanych przez prowadzącego zajęcia. Nie jest konieczne podawanie pełnej nazwy strony internetowej, wystarczy jej główny człon

5.4.4. Zawartość kolekcji AlertsHistory

Kolekcja **AlertsHistory** przechowuje informacje o wszystkich alarmach dotyczących wszystkich użytkowników, którzy korzystali ze stacji roboczych w czasie działania aplikacji agenta LabEye. *Rysunek 13* prezentuje przykładową zawartość tej kolekcji:



Rysunek 13: Przykładowa zawartość kolekcji AlertsHistory

Jak widać schemat pojedynczego dokumentu jest identyczny jak schemat pojedynczego elementu znajdującego się w liście **Alerts** w kolekcji **Workstations**. Jest to zabieg celowy, którego zadaniem jest ułatwienie przeglądania kolekcji **AlertsHistory.**

5.4.5. Zawartość kolekcji fs.files

Jest to kolekcja odpowiedzialna za przechowywanie informacji o plikach zdjęć wysyłanych do bazy danych w momencie wystąpienia alarmu. Za operacje wysyłania i pobierania odpowiedzialny jest *GridFS*. *Rysunek 14* przedstawia przykładową zawartość tej kolekcji.

```
_id: ObjectId("5b2f985ed8f21c3a2023609f")
length: 224871
chunkSize: 261120
uploadDate: 2018-06-24 15:10:54.650
md5: "593eff1d1321798ffa2fb7358d64c26e"
filename: "Image1"

_id: ObjectId("5b2f985ed8f21c3a202360a1")
length: 225774
chunkSize: 261120
uploadDate: 2018-06-24 15:10:55.015
md5: "19f70e9b738e1a9dcc3ce22d218c74ba"
filename: "Image2"

_id: ObjectId("5b2f985fd8f21c3a202360a3")
length: 225620
chunkSize: 261120
uploadDate: 2018-06-24 15:10:55.350
md5: "aee3c5970ba1cb9568d2ae619ebb6876"
filename: "Image3"

_id: ObjectId("5b2f985fd8f21c3a202360a5")
length: 225751
chunkSize: 261120
uploadDate: 2018-06-24 15:10:55.669
md5: "e02085a5ff46f979643cbf4e2951a844"
filename: "Image1"

_id: ObjectId("5b2f985fd8f21c3a202360a7")
length: 226017
chunkSize: 261120
uploadDate: 2018-06-24 15:10:56.016
md5: "64f0927039085c904214f532773c85a6"
filename: "Image2"

_id: ObjectId("5b2f9860d8f21c3a202360a9")
length: 229445
chunkSize: 261120
uploadDate: 2018-06-24 15:10:56.360
md5: "85c0cfb95364463ec84537be3061d697"
filename: "Image3"
```


Rysunek 14: Przykładowa zawartość kolekcji fs.files

- **_id**
  - Zawiera informacje dotyczące ID danego dokumentu. Każdy z nich posiada unikatowy identyfikator, który przydzielany jest automatycznie w momencie utworzenia wpisu w bazie danych aplikacji LabEye. Pozwala na przypisanie poszczególnych kawałków konkretnym plikom

- **length**
  - Przechowuje informacje o rozmiarze pliku, który znajduje się w kolekcji **fs.chunks.** Podawane w bajtach
- **chunkSize**
  - W tym polu zawarta jest informacja o maksymalnym rozmiarze kawałka wysłanego pliku, z wyjątkiem ostatniej części, która jest tak duża jak to tylko możliwe. Oryginalnie wartość ta wynosi 255 kilobajtów
- **uploadDate**
  - Pole to przechowuje informacje o dacie pierwszego pojawienia się tego pliku w bazie danych LabEye. Jest ono typu *Date*
- **md5**
  - Jest to skrót kompletnego pliku zwrócony przez komendę filemd5. Wartość ta przyjmuje typ *String*
- **filename**
  - pole opcjonalne pozwalające na nadanie przyjaznej człowiekowi nazwy przechowywanego pliku

5.4.6. Zawartość kolekcji fs.chunks

Kolekcja ta jest odpowiedzialna za przechowywanie plików podzielonych na kawałki zadanej wielkości. Wysyłanie i pobieranie odbywa się za pomocą *GridFS*. Jest ona ściśle powiązana z kolekcją **fs.files**. *Rysunek 15* przedstawia przykładową zawartość tej kolekcji.

- **_id**
  - Jest to unikatowy identyfikator poszczególnego kawałka utworzonego z pliku, który tej formie przechowywany jest w bazie danych LabEye. Dzięki niemu możliwa jest identyfikacja kawałków w celu ich późniejszego pobrania i złożenia w pełny plik
- **files_id**
  - Pole identyfikatora rodzica znajdującego się w kolekcji **fs.files**. Jest ono identyczne dla wszystkich kawałków opisujących jeden podzielony plik. Służy jako odwołanie
- **n**
  - Pole określające numer sekwencyjny danego kawałka. Duże pliki dzielone są na mniejsze kawałki, każdy z nich posiada swój numer porządkowy wykorzystywany do łączenia kawałków w jeden plik podczas ich pobierania za pomocą aplikacji prowadzącego LabEye. Numeracja rozpoczynana jest od liczby zero, a każdy kolejny kawałek posiada wartość o jeden wyższą.
- **data**
  - Pole przechowujące ładunek zawierający dane uzyskane z podzielenia pliku na mniejsze kawałki. Dane te przechowywane są w binarnym typie BSON, który jest formatem serializacji służącym do przechowywania dokumentów i zdalnego wywoływania funkcji w MongoDB. Powstał poprzez połączenie słów *binary* oraz *JSON*

Rysunek 15: Przykładowa zawartość kolekcji fs.chunks

6. Charakterystyczne i ciekawe fragmenty kodu

LabEye jest stosunkowo rozbudowanym projektem, w związku z tym zawiera umiarkowanie dużo linijek kodu. Z tego powodu warto wspomnieć i wyjaśnić kilka charakterystycznych operacji, które odpowiadają za realizację poszczególnych zadań i funkcjonalności.

6.1. Zdjęcia

Operacją wartą wspomnienia i wyjaśnienia jest wysyłanie i pobieranie plików zdjęć z serwera bazy danych. Operacja opiera się na wykorzystaniu *GridFS* oraz kilku prostych metod konwertujących. *Rysunek 16* przedstawia główną metodę odpowiedzialną za wysyłanie zdjęć.

```
/// <summary>
/// Used to send image to gridFS of database
/// </summary>
/// <param name="img">image source</param>
/// <param name="name">name of the file</param>
/// <returns>Id pointing to a new img</returns>
Odwołania: 0 | 0 zmian | 0 autorów, 0 zmian
public ObjectId SendImage(Image img, string name)
{
    var bytes = ImageToBytes(img);
    var bucket = new GridFSBucket(db);
    ObjectId id = bucket.UploadFromBytes(name, bytes);
    return id;
}
```

Rysunek 16 : Metoda wykorzystywana do konwersji oraz wysłania pliku do bazy danych

Jak widać jest to metoda dwuargumentowa, która jako parametry przyjmuje obiekty typu:

- *Image* img
    - Przechowujący wcześniej już utworzony plik zdjęcia dowodu wykonanego przez aplikacje agenta LabEye
- *string* name
    - Przechowujący nazwę wcześniej utworzonego pliku zdjęcia dowodu wykonanego przez aplikacje agenta LabEye

Następnie przy pomocy metody *ImageToBytes*, przedstawionej na *Rysunku 17*, wykonywana jest konwersja obiektu *Image* do jego reprezentacji w bajtach.

Kolejnym krokiem jest utworzenie nowego obiektu *GridFSBucket*. Jest to klasa dostarczana przez bibliotekę *GridFS* oraz realizująca operacje przesyłania zdjęcia w formie bajtów do bazy danych. Jako parametr przyjmuje odniesienie do wcześniej pozyskanej bazy, która ma przechowywać dane

Linijka następna odpowiada już za samą realizację wysłania zdjęcia do bazy danych LabEye. Wykorzystywana jest do tego metoda, która przyjmuje dwa parametry:

- nazwę pliku przesyłanego do bazy danych
- sam plik reprezentowany w bajtach

W momencie gdy operacja zakończy się sukcesem zwracany jest unikatowy identyfikator typu *ObjectId* do wcześniej utworzonej zmiennej.

Ostatnim krokiem jest zwrócenie przez metodę unikatowego identyfikatora pliku przechowywanego w bazie danych w celu jego późniejszego wykorzystania.

```
/// <summary>
/// Used to convert Image type to bytes
/// </summary>
/// <param name="imageIn">variable which contains image</param>
/// <returns>array of bytes</returns>
1 odwołanie | 0 zmian | 0 autorów, 0 zmian
public static byte[] ImageToBytes(Image imageIn)
{
    ImageConverter imageConverter = new ImageConverter();
    byte[] bytes = (byte[])imageConverter.ConvertTo(imageIn, typeof(byte[]));
    return bytes;
}
```

Rysunek 17: Metoda wykorzystywana do konwersji zdjęcia typu Image do tablicy bajtów

Metoda *ImageToBytes* jako parametr przyjmuje obiekt typu *Image* przechowujący plik zrzutu ekranu wykonanego przy wystąpieniu alarmu, podczas pracy aplikacji agenta LabEye

Na samym początku tworzony jest nowy obiekt typu *ImageConverter*. Jak sama nazwa wskazuje wykorzystywany jest w celu przekonwertowania jednego typu obiektu na drugi.

Następnie przy pomocy metody *ConvertTo* przyjmującej jako parametry obiekt zawierający dane ze zdjęciem oraz docelowy typ jaki chcemy uzyskać po konwersji. W tym przypadku jest to typ *byte[]*.

Metoda zwraca tablicę bajtów do wcześniej utworzonej zmiennej. Z kolei ta natomiast jest już ostatecznie zwracana przez metodę *ImageToBytes*.

```
/// <summary>
/// Used to download image from Alerts collection
/// </summary>
/// <param name="alert">Single alert field</param>
/// <param name="imgId">Id of downloading image</param>
/// <param name="number">ordinal number, must be unique</param>
/// <returns></returns>
Odwołania: 3 | 0 zmian | 0 autorów, 0 zmian
public string DownloadImage(Alerts alert, ObjectId imgId, int number)
{
    var bucket = new GridFSBucket(db);
    var bytes = bucket.DownloadAsBytes(imgId);
    var image = BytesToImage(bytes);

    string path = alert.StudentFirstAndLastName+"\\"+alert.AlertName;
    //If Directory does not exists then...
    if(!(Directory.Exists(path))) Directory.CreateDirectory(path);
    string cleanString = Regex.Replace(alert.AddDate, @"[^0-9]", "_");
    string fileName = cleanString + "_"+number+".jpg";
    path = Path.Combine(path, fileName);
    Bitmap b = new Bitmap(image);
    b.Save(path);
    return path;
}
```

Rysunek 18: Metoda odpowiedzialna za pobieranie pliku zrzutu ekranu z bazy danych LabEye

Przedstawiona na *Rysunku 18* metoda *DownloadImage* jest metodą trójargumentową, która w celu pozyskania pliku z bazy danych wymaga trzech parametrów:

- *alert*
  - obiekt typu *Alert* przechowujący informacje alarmie, który wygenerował plik zrzutu ekranu
- *imgId*
  - obiekt typu *ObjectId* przechowujący unikatowy identyfikator pliku znajdującego się w bazie danych
- *number*
  - obiekt typu *int* określający numer porządkowy wykorzystywany w celu sortowania oraz rozróżniania kolejności wykonanych przez aplikacje agenta LabEye zrzutów ekranów.

Podobnie jak w przypadku operacji wysyłania, wykorzystywana jest klasa *GridFSBucket*. Dane pobierane są w formie bajtów za pomocą metody *DownloadAsBytes*. Przyjmuje ona unikatowy identyfikator zdjęcia jako parametr, a następnie zwraca plik do wcześniej utworzonej zmiennej *bytes*.

Kolejnym krokiem jest wykorzystanie metody *BytesToImage* do uzyskania obiektu typu *Image* z wcześniej pobranych bajtów. *Rysunek 19* przedstawia ciało metody *BytesToImage*.

Zmienna *path* przechowuje ścieżkę folderów w formacie *<Imię i nazwisko studenta>\<Nazwa alarmu>,* gdzie *<Nazwa alarmu>* może przyjmować dwa schematy:

- "*Wykryto aplikację proces ze słowem kluczowym <Nazwa aplikacji>*", gdzie *<Nazwa aplikacji>* jest nazwą aplikacji, która została uruchomiona w trakcie działania Agenta na stacji roboczej i została przez niego zdefiniowana jako niedozwolona
- "*W tablicy DNS wykryto słowo kluczowe <zakazana witryna>*", gdzie *<zakazana witryna>* jest stroną internetową, która podczas pracy Agenta została wykryta jako jedna ze stron znajdujących się na czarnej liście

W przypadku gdy dany folder docelowy nie istnieje zostanie utworzony przy użyciu wcześniej przygotowanej ścieżki.

Etap następny przedstawia proces tworzenia nazwy pliku zdjęcia. W tym celu należy utworzyć czysty ciąg znaków, ponieważ niektóre z nich są nieakceptowalne. Wykorzystywane jest do tego wyrażenie regularne, które pozwala na używanie tylko i wyłącznie cyfr od 0 do 9, a każdy inny znak zamieniany jest na "_". Po uzyskaniu już czystego ciągu znaków reprezentującego datę i godzinę powstania zdjęcia dodawany jest numer porządkowy, który przybiera wartości od 1 do 3 w zależności od kolejności pozyskiwanego zdjęcia. Ostatecznym krokiem tego etapu jest dodanie rozszerzenia pliku w formacie **.jpg*.

Następnie tworzona jest *Bitmapa* z pliku zawierającego zrzut ekranu pobrany z bazy danych i już przekonwertowany do typu *Image*.

Ostatecznie z wykorzystaniem utworzonej wcześniej ścieżki i nazwy pliku zapisywana jest bitmapa gotowa do wyświetlenia jako zdjęcie w formacie **.jpg*, a sama ścieżka jest zwracana przez metodę.

```
/// <summary>
/// Used to convert bytes to Image type
/// </summary>
/// <param name="bytes">bytes containing Image</param>
/// <returns>instance of Image type</returns>
1 odwołanie | 0 zmian | 0 autorów, 0 zmian
public static Image BytesToImage(byte[] bytes)
{
    Image x = (Bitmap)((new ImageConverter()).ConvertFrom(bytes));
    return x;
}
```

Rysunek 19: Ciało metody BytesToImage

Jak widać na *Rysunku 19* metoda *BytesToImage* jest bardzo prosta w swej budowie, mieszcząc się w jedynie dwóch linijkach. Jako argument przyjmuje tablicę bajtów, która to następnie wykorzystywana jest przez metodę *ConvertFrom* zawartej

w klasie *ImageConverter*, do wygenerowania, po odpowiednim rzutowaniu, *Bitmapy* przypisanej do obiektu typu *Image*. Metoda zwraca ten obiekt.

6.2. Filtrowanie odwiedzanych stron internetowych

Mechanizm monitorowania aktywności w sieci opiera się na przeglądaniu tablicy resolvera DNS i przeszukiwaniu go pod kątem słów kluczowych. W ramach systemu, nie blokujemy żadnych stron ani połączeń, prowadzony jest jedynie obserwacja

Zaletami takiego rozwiązania jest przede wszystkim łatwość implementacji oraz bardzo duża wydajność obliczeniowa procesu który zwyczajnie nie jest skomplikowany. Taka metoda monitoringu jest również odporna na programy maskujące aktywność, jak również na połączenia po przez VPN. Resolver DNS jest tak elementarną częścią systemu operacyjnego iż nie ma możliwości właściwie ominięcia takiego monitoringu.

Wadą takiego rozwiązania jest konieczność bardzo dokładnego wybierania słów kluczowych oraz brak możliwości rozróżnienia faktycznego odwiedzania strony internetowej od aplikacji korzystających z usług jakiejś strony. Przykładowo, Facebook oraz Google są bardzo popularnymi domenami internetowymi oraz dostawcami różnorodnych funkcjonalności dla innych stron internetowych. Praktycznie nie możliwe jest określenie czy użytkownik wszedł na stronę facebook.com czy znajduje się na stronie która w swoim źródle odwołuje się do jakiegoś zasobu na serwerze *facebook'a*.

```
/// <summary>
/// Getting actual dns table.
/// </summary>
/// <returns>String contains whole DNS table.</returns>
1 odwołanie | 0 zmian | 0 autorów, 0 zmian
private string Getdnstable()
{
    Process process = new Process();
    ProcessStartInfo startInfo = new ProcessStartInfo();
    startInfo.WindowStyle = ProcessWindowStyle.Hidden;
    startInfo.FileName = "cmd.exe";
    startInfo.Arguments = "/C ipconfig /displaydns";
    startInfo.RedirectStandardOutput = true;
    startInfo.UseShellExecute = false;
    startInfo.CreateNoWindow = true;
    process.StartInfo = startInfo;
    process.Start();
    var msg = process.StandardOutput.ReadToEnd();
    process.WaitForExit();
    return msg;
}
```

Rysunek 20: Fragment kodu odpowiedzialny za pozyskiwanie danych z tablicy resolvera DNS

Jak widać na *Rysunku 20* całość opiera się na wykorzystaniu konsoli systemowej, której to polecenia wykonywane są w tle, w ukryciu przed użytkownikiem stacji roboczej. Największym minusem tego rozwiązania jest fakt iż podmiana programu odpowiedzialnego za konsolę spowoduje kompletne wyłączenie tej funkcjonalności agenta LabEye. Pozyskana tablica zapisywana jest w zmiennej, która to następnie jest zwracana przez metodę *Getdnstable*.

6.3. Filtrowanie uruchomionych aplikacji i aktywnych procesów

Analogicznie do monitorowania stron internetowych, moduł monitorowania aplikacji działa na zasadzie przeglądania listy procesów oraz procesów pod kątem słów kluczowych. Jako listę poddawaną takiemu przeszukiwaniu składają się nazwy otwartych okien aplikacji oraz nazwy procesów.

Podobnie jak monitorowanie stron internetowych ten moduł działa bardzo szybko, ze względu na możliwość szybkiego dostępu do danych. Ponieważ średnio podczas pracy rekordów w takiej liście jest około 300, przeszukiwanie nie zajmuje dużo czasu.

```
/// <summary>
/// Checking running processes by its window title and name.
/// </summary>
/// <returns>List of running processes</returns>
Odwołania: 2 | mmmati1996, 39 dni temu | 1 autor, 1 zmiana
public List<string> GetRunningApplications()
{
    List<string> processList = new List<string>();
    try
    {
        Process[] processes = Process.GetProcesses();
        foreach (Process p in processes)
        {
            if (!String.IsNullOrEmpty(p.MainWindowTitle))
                processList.Add(p.MainWindowTitle);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show("Wystąpił błąd z pobieraniem procesów. Treść błędu: /n /n" + ex);
    }
    return processList;
}
```

Rysunek 21: Kod odpowiedzialny za pozyskiwanie list obecnie uruchomionych aplikacji działających procesów

Po przyjrzeniu się *Rysunkowi 21* można zauważyć wadę tego rozwiązania. Jest nią fakt, że proces musi posiadać tytuł okna, aby mógł zostać wykryty i dodany do listy. Pozwala to jednak na wstępne przefiltrowanie listy procesów, a następnie odrzucenie tych z nich, które są kluczowe do działania systemu, a informacja o ich istnieniu nie przyniosłaby żadnych korzyści. Założenie zakładało, że student może korzystać z gotowych źródeł w formie tekstowej, bądź aplikacji ułatwiających pracę,

które mogłby być nieznane prowadzącemu zajęcia i to właśnie one miałyby się znajdować na tej liście.

6.4. Przechwytywanie obrazu, wykonywanie zrzutów ekranu

Mechanizm wykonywania zrzutów ekranu jest wykorzystywany przy przechwytywaniu ekranu oraz przesyłaniu logów w postaci zdjęć do aplikacji prowadzącego. *Rysunek 22* przedstawia fragment kodu odpowiedzialny za wykonywanie zrzutów ekranu stacji roboczej z zainstalowaną aplikacją agenta LabEye.

```
/// <summary>
/// Used to make screen shot
/// </summary>
/// <returns>screen shot as a Bitmap object</returns>
1 odwołanie | mmmati1996, 97 dni temu | 1 autor, 1 zmiana
private Bitmap GetDesktopScreenshot()
{
    double screenLeft = SystemParameters.VirtualScreenLeft;
    double screenTop = SystemParameters.VirtualScreenTop;
    double screenWidth = SystemParameters.VirtualScreenWidth;
    double screenHeight = SystemParameters.VirtualScreenHeight;
    Bitmap screenshot = new Bitmap((int)screenWidth, (int)screenHeight, PixelFormat.Format32bppArgb);
    Graphics graph = Graphics.FromImage(screenshot);
    graph.CopyFromScreen((int)screenLeft,(int)screenTop,0,0,screenshot.Size);
    return screenshot;
}
```

Rysunek 22: Wykonywanie zrzutów ekranu

Pozyskiwane są parametry położenia obrazu, a następnie z pomocą metody *CopyFromScreen* uzyskiwany jest zrzut ekranu przechowywany w zmiennej *screenshot*, która jest typu *Bitmap*.

Uzyskane w ten sposób zdjęcie wysyłane jest z pomocą protokołu TCP do aplikacji prowadzącego. Tam z kolei jest wyświetlane w osobnym oknie. Niestety nie jest to rozwiązanie najoptymalniejsze, jednak w względzie jakościowym jedno z lepszych.

7. Instrukcja obsługi aplikacji

Szczegółowe objaśnienie możliwych operacji oraz ich obsługa zostały objaśnione w *rozdziale 5. Architektura rozwiązania*. LabEye jest aplikacją ściśle powiązaną z bazą danych MongoDB. Obecnie jedna instancja znajduje się na hostingu *mlab.com*. Niestety ogranicza on rozmiar do 500mb. Może on zostać w łatwy sposób przekroczony poprzez mechanizm przechowywania zrzutów ekranu, w związku z tym baza musi być czyszczona z pobranych już, jako logi, zdjęć. W celu połączenia się należy podać następujący *connection string*:

mongodb://<dbuser>:<dbpassword>@ds161780.mlab.com:61780/lab_eye_mongo

Gdzie za *<dbuser>* należy wpisać *testUser*, a za *<dbpassword>* *v3ry#h@rd#p@ssVv0rd*.

8. Perspektywy rozwoju

LabEye był projektowany z myślą o spełnieniu wszystkich swoich funkcjonalności w jak najlepszy sposób. Każdy element przygotowywany był w bardzo staranny sposób, jednakże jak wiadomo nic nie jest idealne, a to znaczy, że wszystko może zostać usprawnione.

Zgodnie z tą ideą rozdział ten krótko przedstawi kilka rzeczy, które zdaniem autorów aplikacji LabEye mogłyby zostać poprawione w przyszłości:

- Usprawnić metodę pozyskiwania lub filtrowania stron internetowych, uniezależnić ją od programów zewnętrznych
- Poprawić sposób filtrowania listy procesów
- Dodać funkcjonalność umożliwiającą wykrycie podłączenia urządzeń zewnętrznych do stacji roboczej
- Wykorzystać bibliotekę VncSharp pozwalającą na pełne przejmowanie kontroli nad ekranem (klawiatura, myszka)
- Zoptymalizować sposób przechowywania zrzutów ekranu w bazie danych np. przechowywać linki zamiast pełnych danych
- Zaimplementować bardziej zoptymalizowany i rozbudowany interfejs graficzny aplikacji prowadzącego