

COLLEGIUM WITELONA
Uczelnia Państwowa

Wydział Nauk Technicznych i Ekonomicznych
Kierunek Informatika
Specjalność Programowanie Aplikacji Mobilnych i Internetowych

Sebastian Górska

**Nethelt – wieloplatformowy system monitorowania urządzeń sieciowych z modułem wykrywania anomalii
opartym na AI**

**Praca dyplomowa inżynierska
napisana pod kierunkiem
dr inż. Zbigniew Fryźlewicz**

Legnica 2026 rok

Spis treści

1 Wstęp	3
1.1 Wprowadzenie do problematyki	3
1.2 Cel pracy	4
1.3 Zakres pracy	4
1.4 Założenia projektowe	5
2 Warstwa API	7
2.1 Struktura bazy danych	7
3 Model SI do wykrywania anomalii	9
3.1 Założenia	9
3.2 Użyte narzędzia	9
3.3 Trening modelu SI	10
3.3.1 Źródło danych	10
3.3.2 Przygotowanie i agregacja danych	10
3.3.3 Trening modelu	11
3.3.4 Parametry modelu	11

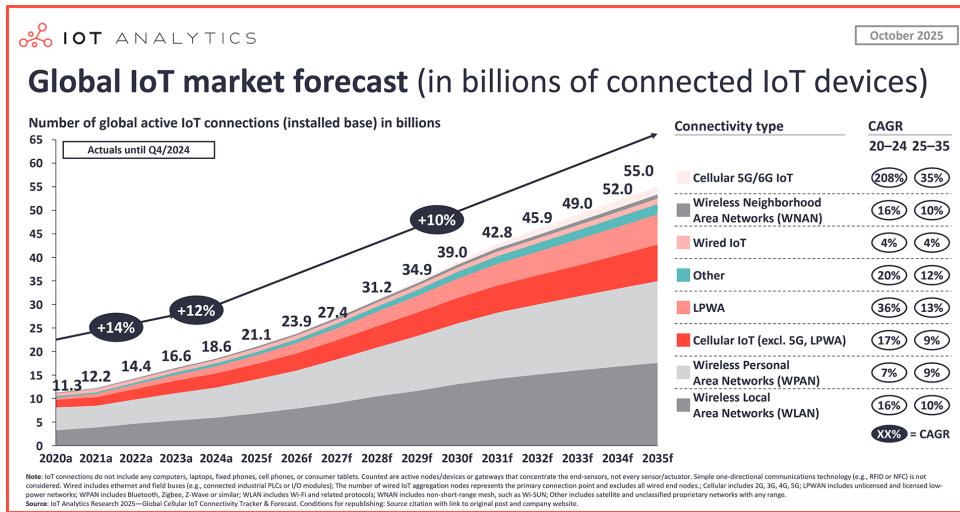
1. Wstęp

1.1 Wprowadzenie do problematyki

Aktualnie nie sposób wyobrazić sobie firmy, przedsiębiorstwa, a nawet gospodarstwa domowego, w którym nie korzysta się z internetu. Dostęp do zewnętrznych serwisów i usług stał się naszą codziennością. Korzystamy z internetu oraz urządzeń, łączących się do niego każdego dnia. W domach internet służy nam często do rozrywki i ułatwiania codziennych czynności, do wyszukiwania informacji i rozwiązywania problemów. Dzięki niemu jesteśmy w stałym kontakcie z innymi, mamy dostęp do płatności elektronicznych, a zakupy możemy zrobić nie wychodząc z domu. W pracy korzystamy z niego w celu wyszukiwania informacji i przesyłania ich dalej, wysyłania maili, pobierania danych z Państwowych spółek, czy chociażby robienia wideokonferencji z klientami z całego świata.

Poza dostępem do zewnętrznych usług coraz większą popularność zyskują urządzenia Smart oraz IoT (Internet of Things). Wielu z nas nawet nie ma pojęcia na temat tego jak wiele takich urządzeń znajduje się w naszym otoczeniu. Według IoT Analytics pod koniec 2025 r. liczba podłączonych urządzeń IoT na świecie miała osiągnąć 21,1 miliarda To wzrost o ok. 14% w stosunku do roku poprzedniego **TODO: zweryfikować dane przed oddaniem pracy**. Według prognozy liczba ta będzie tylko rosła i w 2030 roku może sięgnąć 39 miliardów[1]. Tendencję tę można zaobserwować na rysunku 1.1 przedstawiający liczbę aktywnych urządzeń typu IoT wraz z prognozą na przyszłe lata.

Pojawienie się tak dużej ilości urządzeń, z których specyfikacji wynika, że wręcz wymagają stałego połączenia z siecią, tworzy nowe problemy i pytania. Kluczowym pytaniem, które powinniśmy sobie zadać, jest to, czy przy utracie łączności z siecią takiego urządzenia pojawią się poważne komplikacje. Chociaż w przypadku np. lodówki bądź ekspresu do kawy działającym w sieci, brak połączenia nie wyrządzi sporych szkód, tak jednak w przypadku maszyn na hali produkcyjnej lub sygnalizacji świetlnej w centrum miasta, szkody mogą być spore i mogą zagrażać nie tylko sytuacji majątkowej, ale też bezpieczeństwu ludzi. W związku z powyższym należy monitorować stan takich urządzeń, a w przypadku ich awarii jak najszybciej dążyć do naprawy usterki.



Rysunek 1.1: Liczba urządzeń typu IoT w latach 2020–2035

Źródło: <https://iot-analytics.com/number-connected-iot-devices/>

1.2 Cel pracy

Celem niniejszej pracy jest stworzenie systemu wielomodułowego do monitorowania i zarządzania urządzeniami sieciowymi w środowisku lokalnym (LAN). System ma umożliwiać rejestrację i logowanie użytkowników, konfigurację połączeń sieciowych oraz monitorowanie stanu urządzeń.

Dodatkowym celem jest opracowanie modułu detekcji anomalii z wykorzystaniem algorytmu *isolation forest*, który pozwoli na wczesne wykrywanie nieprawidłowości w funkcjonowaniu sieci. System powinien wspierać dwa typy użytkowników – administratora oraz użytkownika zwykłego – z odpowiednimi uprawnieniami do zarządzania systemem i dostępem do danych.

Projekt obejmuje także wdrożenie aplikacji webowej w środowisku chmurowym, zapewniając dostępność dla użytkowników końcowych niezależnie od lokalizacji. Aplikacja desktopowa będzie dostępna do pobrania bezpośrednio na stronie internetowej.

1.3 Zakres pracy

Zakres pracy obejmuje:

- implementację aplikacji webowej w technologii Java 25 z użyciem frameworka Spring Boot oraz interfejsu w TypeScript i Angular 21,
- stworzenie desktopowego klienta w Javie 25, działającego w trybie serwisu, umożliwiającego komunikację z serwerem,
- opracowanie modułu AI w Pythonie, wykorzystującego algorytm *isolation forest* do detekcji typowych anomalii w sieci,
- integrację modułów z centralną bazą danych PostgreSQL 18.2, współdzieloną między aplikacjami,

-
- wdrożenie aplikacji w środowisku chmurowym (Google Cloud Console) w celu zapewnienia dostępności dla użytkowników.

W zakresie pracy nie przewiduje się:

- stworzenia mobilnej aplikacji,
- obsługi wszystkich możliwych typów anomalii w sieci – moduł AI ograniczony jest do najczęściej występujących przypadków,
- rozbudowanego GUI w aplikacji desktopowej - wersja komputerowa będzie dostępna w trybie konsolowym. Wersja z interfejsem użytkownika zostanie zaimplementowana, jeśli pozostałe moduły będą gotowe przed ostatecznym terminem.

Zakres dokumentacji obejmuje przygotowanie instrukcji użytkownika, dokumentacji technicznej, diagramów UML oraz podstawowych statystyk działania systemu.

1.4 Założenia projektowe

Projekt ma charakter wielomodułowy i obejmuje trzy główne komponenty:

- aplikację webową,
- aplikację desktopową,
- moduł detekcji anomalii oparty na algorytmie AI typu *isolation forest*.

System zostanie zaprojektowany w taki sposób, aby umożliwić użytkownikowi rejestrację, logowanie, konfigurację połączenia ze swoją siecią LAN, dodawanie lokalnych urządzeń sieciowych, monitorowanie ich stanu oraz zmianę ustawień powiadomień. Aplikacja desktopowa będzie pełniła rolę klienta komunikującego się z serwerem, co jest niezbędne do prawidłowego funkcjonowania systemu w środowisku wielomodułowym. Dodatkowym wyzwaniem projektu będzie integracja wytrenowanego modelu AI, który będzie wspierał wczesne wykrywanie anomalii w sieci.

System przewiduje co najmniej dwa typy użytkowników: administratora oraz użytkownika zwykłego. Zalogowanie do systemu będzie obowiązkowe, a proces logowania zostanie udostępniony zarówno poprzez protokół OAuth2 (Google), jak i w formie lokalnej autoryzacji. Administrator będzie posiadał rozszerzone uprawnienia, w tym możliwość zarządzania użytkownikami oraz przeglądania podstawowych statystyk dotyczących korzystania z systemu.

Aplikacja webowa będzie realizowana w oparciu o nowoczesną wersję języka Java (wersja 25) oraz framework Spring Boot (w wersji 4.x.x), który ułatwia implementację i utrzymanie systemu. Interfejs użytkownika zostanie zbudowany przy użyciu TypeScript oraz frameworka Angular (wersja 21). Moduł AI zostanie opracowany w języku Python z wykorzystaniem odpowiednich bibliotek do uczenia maszynowego, co przyspieszy proces modelowania i analizy

danych. Aplikacja desktopowa zostanie stworzona w tej samej wersji Javy, z której korzysta program internetowy, co pozwoli na jej uruchamianie na różnych systemach operacyjnych.

Dane pomiędzy modułami systemu będą współdzielone poprzez centralną bazę danych, której silnik wykorzysta PostgreSQL w wersji 18.2.

Projekt przewiduje wdrożenie aplikacji w środowisku chmurowym w celu zapewnienia dostępności dla użytkowników końcowych z dowolnej lokalizacji. Wstępnie planowane jest wykorzystanie platformy Google Cloud Console do tego celu.

2. Warstwa API

2.1 Struktura bazy danych

Na rysunku 2.1 przedstawiono schemat relacyjnej bazy danych aplikacji NetHelt. Diagram obrazuje strukturę tabel oraz relacje pomiędzy encjami systemu.

Centralnym elementem modelu jest tabela `users`, przechowująca dane użytkowników systemu wraz z przypisaną rolą (`roles`). Każdy użytkownik może posiadać wiele urządzeń sieciowych zapisanych w tabeli `devices`.

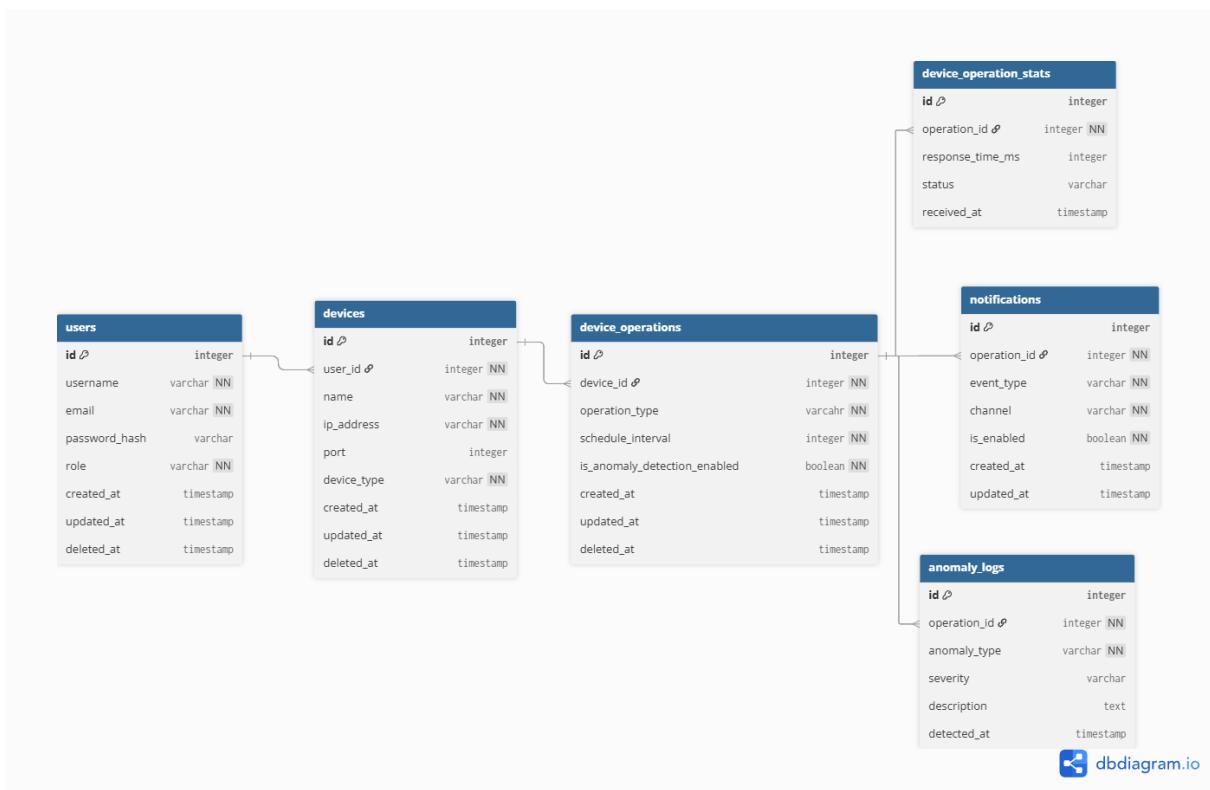
Z każdym urządzeniem powiązane są operacje sieciowe (`device_operations`), określające typ wykonywanej operacji oraz czas między kolejnymi wywołaniami operacji. Dodatkowo w tej tabeli można przełączać wykrywanie anomalii.

Wyniki wykonywanych operacji zapisywane są w `device_operation_stats`, gdzie gromadzone są informacje dotyczące odpowiedzi z urządzenia. Na podstawie zgromadzonych statystyk mogą być tworzone okna czasowe, wysyłane do analizy do modułu AI. W przypadku wykrycia anomalii, wpis taki generowany jest w tabeli `anomaly_logs`.

Konfiguracja powiadomień dotyczących zdarzeń monitorujących przechowywana jest w tabeli `notifications`. Powiadomienia są powiązane z operacjami monitorującymi i definiują kanał komunikacji, typ zdarzenia wywołującego alert oraz status danych powiadomień.

Niektóre tabele posiadają kolumnę `deleted_at`, która pozwala na wykonywanie miękkich usunięć z systemu, co ułatwi utrzymanie integralności danych.

Kolumny takie jak: `role`, `device_type`, `channel` itd. są ograniczone do predefiniowanych wartości, które w kodzie źródłowym programu występują jako typ enumeryczny. Pozwala to uniknąć błędnego wprowadzenia danych poprzez wyświetlenie odpowiedniego komunikatu błędu.



Rysunek 2.1: Diagram bazy danych aplikacji NetHelt
 Źródło: Opracowanie własne w witrynie internetowej <https://dbdiagram.io/>

3. Model SI do wykrywania anomalii

3.1 Założenia

Jednym z kluczowych elementów systemu jest model sztucznej inteligencji przeznaczony do wykrywania anomalii w parametrach jakości połączenia sieciowego na wczesnym etapie ich występowania.

Na obecnym etapie implementacji model umożliwia analizę wyłącznie parametrów związanych z pomiarami czasu odpowiedzi (ping). Dane wejściowe mają postać zagregowanych statystyk obliczanych w stałych oknach czasowych, co pozwala na ograniczenie wpływu pojedynczych, losowych odchyleń oraz umożliwia podejmowanie decyzji w krótkim czasie po zakończeniu danego interwału.

Model wykorzystuje algorytm Isolation Forest, do którego przekazywane są przetworzone dane wejściowe. Przyjęto podejście uczenia nienadzorowanego, zakładające brak oznaczonych przykładów anomalii w zbiorze treningowym. Algorytm uczy się rozkładu obserwacji uznawanych za typowe, a istotne odchylenia od tego rozkładu klasyfikuje jako anomalie.

Ze względu na różnice w specyfikacji i przeznaczeniu w różnego typu urządzeniach sieciowych, model będzie musiał powstać w co najmniej 3 wariantach. Na początkowym etapie projektu zakłada się wytrenowanie modelu na 3 zestawach danych: zebranych z urządzeń podłączonych do sieci poprzez protokół WiFi, z urządzeń typu IoT oraz z urządzeń typu klient LAN (wliczając w to: komputery stacjonarne, serwery, oraz infrastrukturę sieciową - router, switch, AP).

Założono możliwość dalszego rozszerzenia modelu o dodatkowe parametry jakości połączenia oraz ponowne trenowanie modelu wraz ze zmianą charakterystyki środowiska sieciowego.

3.2 Użyte narzędzia

Model został zaimplementowany w języku Python (wersja 3.12) z wykorzystaniem odpowiednich bibliotek wspierających uczenie maszynowe i analizę danych. Wśród tych bibliotek znalazły się:

-
- **Pandas** – do wczytywania, przetwarzania i agregacji danych pomiarowych,
 - **scikit-learn** – do implementacji algorytmu Isolation Forest oraz standaryzacji cech,
 - **Matplotlib** oraz **Seaborn** – do wizualizacji wyników i analizy rozkładu danych

Środowisko projektu zostało zarządzane z wykorzystaniem narzędzia Poetry. Dodatkowo wykorzystana została biblioteka `requests` pozwalająca na komunikację z API aplikacji.

3.3 Trening modelu SI

3.3.1 Źródło danych

Dane wykorzystane w procesie trenowania modelu zostały podzielone na dwa zbiory: zbiór treningowy oraz zbiór testowy. Część danych pochodzi z rzeczywistego środowiska sieciowego, w którym cyklicznie wykonywano pomiary czasu odpowiedzi (ping), a następnie zapisywano je do plików tekstowych.

Pomiary zostały wykonane dla różnych typów urządzeń: komputera stacjonarnego (połączenie Ethernet), laptopa (WiFi), serwera, macierzy dyskowej, routera, przełącznika sieciowego, punktu dostępowego oraz urządzenia typu IoT. Pozwoliło to uzyskać zróżnicowany zbiór danych odzwierciedlający rzeczywiste warunki pracy sieci.

Dodatkowo wygenerowano dane syntetyczne, symulujące skrajne przypadki, takie jak wysoka utrata pakietów lub niestabilny czas odpowiedzi. Dane te umożliwiły weryfikację zachowania modelu w kontrolowanych warunkach.

3.3.2 Przygotowanie i agregacja danych

Surowe dane zostały poddane wstępнемu przetworzeniu obejmującemu usunięcie nieprawidłowych rekordów, zbędnych kolumn oraz ujednolicenie formatu danych.

Następnie pomiary zostały zagregowane do stałych, jednominutowych okien czasowych. Dla każdego okna obliczono wybrane statystyki opisujące jakość połączenia. Kluczowe cechy wejściowe modelu stanowiły:

- **packet_loss_1m** – procent utraconych pakietów w danym oknie czasowym,
- **ping_std_1m** – odchylenie standardowe czasu odpowiedzi w danym oknie.

Rozważano również wykorzystanie cech `ping_avg_1m` oraz `ping_diff_1m`, jednak w dalszych eksperymentach skoncentrowano się na parametrach najlepiej opisujących niestabilność połączenia.

Fragment implementacji agregacji przedstawiono na listingu 3.1.

```

1   for window, g in grouped:
2       successful = g[g["success"] == 1]
3
4       if len(successful) >= 3:
5           ping_std = successful["ping_ms"].std()
6       else:
7           ping_std = float("nan")
8
9       packet_loss = 1 - len(successful) / len(g)
10
11      windows.append({
12          "window_start": window,
13          "device_group": g["device_group"].iloc[0],
14          "ping_std_1m": round(ping_std, 3),
15          "packet_loss_1m": round(packet_loss, 3)
16      })

```

Listing 3.1: Agregacja danych do okien czasowych

3.3.3 Trening modelu

Przygotowany zbiór danych treningowych został wykorzystany do uczenia modelu opartego na algorytmie `Isolation Forest`. Przed rozpoczęciem procesu uczenia dane wejściowe zostały poddane standaryzacji z wykorzystaniem klasy `StandardScaler`.

Standaryzacja polega na przekształceniu każdej cechy do rozkładu o średniej równej 0 oraz odchyleniu standardowym równym 1, co zapobiega dominacji cech o większym zakresie wartości.

Proces przygotowania danych i trenowania modelu przedstawiono na listingu 3.2.

```

1 df = pd.read_csv(csv_path)
2 X = df[FEATURES]
3
4 feature_scaler = StandardScaler()
5 X_scaled = feature_scaler.fit_transform(X)
6
7 model.fit(X_scaled)

```

Listing 3.2: Skalowanie danych i trening modelu

3.3.4 Parametry modelu

Model `Isolation Forest` został skonfigurowany z wykorzystaniem następujących parametrów:

- `n_estimators` – liczba drzew izolacyjnych budujących model,

-
- **contamination** – oczekiwany udział anomalii w zbiorze danych,
 - **random_state** – parametr zapewniający powtarzalność wyników.

Parametr `contamination` wpływa bezpośrednio na próg decyzyjny modelu i określa, jaki odsetek obserwacji zostanie zaklasyfikowany jako anomalny. Parametr `n_estimators` odpowiada za stabilność działania modelu – większa liczba drzew zwiększa powtarzalność wyników kosztem wydłużenia czasu treningu.

Konfigurację modelu przedstawiono na listingu 3.3.

```
1 RANDOM_STATE = 42
2
3 def create_model(contamination: float, n_estimators: int) ->
4     IsolationForest:
5     return IsolationForest(
6         n_estimators=n_estimators,
7         contamination=contamination,
8         random_state=RANDOM_STATE,
9     )
```

Listing 3.3: Inicjalizacja modelu Isolation Forest

Po zakończeniu procesu uczenia wytrenowany model wraz ze skalarem został zapisany do pliku w formacie `joblib`, co umożliwia jego późniejsze wykorzystanie w aplikacji bez konieczności ponownego trenowania.

TODO: Dodać testowanie modelu (wykresy itd)

Spis rysunków

1.1	Liczba urządzeń typu IoT w latach 2020–2035 Źródło: https://iot-analytics.com/number-connected-iot-devices/	4
2.1	Diagram bazy danych aplikacji NetHelt Źródło: Opracowanie własne w witrynie internetowej https://dbdiagram.io/	8

Spis tabel

Spis listingów

3.1	Agregacja danych do okien czasowych	11
3.2	Skalowanie danych i trening modelu	11
3.3	Inicjalizacja modelu Isolation Forest	12

Bibliography

- [1] S. Sinha, *State of IoT 2025: Number of connected IoT devices growing 14% to 21.1 billion globally*, 2025, <https://iot-analytics.com/number-connected-iot-devices/> [dostęp: 07.12.2025].