Uniwersytet Bielsko-Bialski

PRACA DYPLOMOWA

WYDZIAŁ
BUDOWY MASZYN I INFORMATYKI

KIERUNEK: Informatyka

SPECJALNOŚĆ: TECHNIKI TWORZENIA OPROGRAMOWANIA

Maciej Tonderski

nr albumu: 62572

Praca magisterska

Uproszczenie procesu wdrażania środowisk HomeLab poprzez projekt i implementację zintegrowanego systemu zarządzania.

Kategoria pracy: projektowa

Promotor: dr inż. RUSLAN SHEVCHUK

Bielsko-Biała, 2025

Streszczenie

W niniejszej pracy magisterskiej przedstawiono projekt i implementację systemu HomeLab, który umożliwia użytkownikom łatwe zarządzanie infrastrukturą IT w środowisku domowym. Celem projektu było stworzenie rozwiązania, które pozwala na automatyczne wdrażanie i kontrolowanie maszyn wirtualnych, kontenerów oraz zasobów sieciowych, przy jednoczesnym zapewnieniu wysokiego poziomu bezpieczeństwa i intuicyjności obsługi.

Praca rozpoczyna się od omówienia koncepcji HomeLab oraz analizy istniejących rozwiązań do zarządzania infrastrukturą IT, takich jak Proxmox, Unraid, Docker, Kubernetes i inne narzędzia do automatyzacji oraz monitorowania. Wskazano ich zalety i ograniczenia, co pozwoliło na określenie luki technologicznej, którą uzupełnia proponowany system.

Kolejne rozdziały opisują szczegóły techniczne dotyczące architektury systemu, sposobu implementacji oraz wykorzystanych technologii. Główne elementy składające się na rozwiązanie to:

- Backend stworzony w technologii GoLang zarządzający uzytkownikami, usługami oraz monitorujący system.
- Baza danych SQLite przechowująca informację o zarejestrowanych uzytkownikach, przeprowadzonych ostatnich pomiarach oraz monitorująca
- Interfejs użytkownika stworzony przy użyciu TypeScript oraz szablonu TailAdmin,
- Mechanizmy uwierzytelniania i autoryzacji z wykorzystaniem JWT, zapewniające bezpieczeństwo operacji,

W pracy przeprowadzono również testy wydajnościowe, które potwierdziły, że system jest w stanie obsłużyć duże obciążenie i działa stabilnie przy wysokiej liczbie jednoczesnych użytkowników. Ponadto przedstawiono teoretyczne aspekty testowania bezpieczeństwa, wskazując najlepsze praktyki oraz narzędzia stosowane do analizy podatności systemu.

Opracowany system HomeLab stanowi kompleksowe i skalowalne rozwiązanie, które może być wykorzystywane przez administratorów IT, pasjonatów nowych technologii oraz osoby chcące zwiększyć swoją kontrolę nad infrastrukturą IT w środowisku domowym. Praca ta dostarcza także szczegółowej analizy istniejących narzędzi oraz wdrożonych mechanizmów, stanowiąc podstawę do dalszego rozwoju podobnych systemów.

Spis treści

1. Wprowadzenie

Współczesna technologia informatyczna umożliwia pasjonatom IT, administratorom systemów oraz programistom budowanie i zarządzanie własnymi środowiskami testowymi oraz produkcyjnymi w domowych warunkach. Koncepcja HomeLab, czyli prywatnego środowiska IT, zyskuje na popularności dzięki coraz szerszemu dostępowi do wydajnego sprzętu, technologii wirtualizacji oraz narzędzi do automatyzacji zarządzania infrastrukturą. W dobie rosnącej cyfryzacji oraz postępującej popularyzacji rozwiązań chmurowych, coraz więcej osób decyduje się na stworzenie własnego, niezależnego środowiska IT, które może służyć zarówno celom edukacyjnym, jak i profesjonalnym.

Dzięki HomeLab użytkownicy mogą zdobywać praktyczne doświadczenie w administrowaniu serwerami, testować nowe technologie, wdrażać rozwiązania chmurowe oraz konfigurować usługi sieciowe. Środowisko to może również pełnić funkcję osobistego centrum danych, umożliwiającego przechowywanie plików, hostowanie aplikacji czy zarządzanie systemami Internetu Rzeczy (IoT). Wiele osób wykorzystuje HomeLab jako platformę do nauki DevOps, testowania nowych systemów operacyjnych oraz rozwijania umiejętności związanych z cyberbezpieczeństwem.

Jednak dla wielu użytkowników proces konfiguracji i utrzymania takiego środowiska może być skomplikowany i czasochłonny. Wymaga on znajomości technologii wirtualizacji, zarządzania siecią, administracji systemami oraz narzędzi do automatyzacji. Dodatkowym wyzwaniem jest zapewnienie bezpieczeństwa systemu oraz zdalnego dostępu do jego zasobów, co wymaga stosowania odpowiednich mechanizmów uwierzytelniania i autoryzacji.

1.1 Cel pracy

Celem niniejszej pracy magisterskiej jest zaprojektowanie i implementacja systemu HomeLab, który uprości proces budowy, konfiguracji oraz zarządzania własną infrastrukturą IT. System ten ma zapewnić użytkownikom intuicyjne narzędzia do zarządzania serwerami, maszynami wirtualnymi, kontenerami oraz siecią, a także umożliwić zdalny, bezpieczny dostęp do zasobów. Kluczowym założeniem projektu jest maksymalna automatyzacja procesów, co pozwoli na minimalizację konieczności manualnej konfiguracji i zwiększy wygodę użytkowania.

Praca ta ma na celu stworzenie kompleksowego rozwiązania, które pozwoli użytkownikom na:

- Proste i szybkie wdrażanie usług i aplikacji w środowisku HomeLab,
- Centralne zarządzanie infrastrukturą z poziomu intuicyjnego interfejsu użytkownika,
- Automatyzację procesów administracyjnych, takich jak aktualizacje systemów, backupy czy monitorowanie wydajności,
- Zapewnienie mechanizmów bezpieczeństwa chroniących zasoby przed nieautoryzowanym dostępem,
- Integrację z popularnymi technologiami wykorzystywanymi w branży IT, takimi jak Docker, Kubernetes, Proxmox czy Ansible.

Opracowany system ma na celu obniżenie progu wejścia dla użytkowników, którzy chcieliby rozpocząć przygodę z HomeLab, ale nie posiadają zaawansowanej wiedzy technicznej. Jednocześnie zapewni on elastyczność i rozszerzalność, dzięki czemu bardziej doświadczeni użytkownicy będą mogli dostosować go do swoich potrzeb.

1.2 Zakres pracy

W ramach niniejszej pracy zostaną omówione istotne aspekty techniczne związane z budową HomeLab, w tym wybór odpowiednich technologii, metod zarządzania infrastrukturą oraz zapewnienia jej bezpieczeństwa. Ponadto przedstawiona zostanie analiza istniejących rozwiązań oraz uzasadnienie wyboru implementowanych funkcjonalności. Efektem końcowym pracy będzie gotowy system, który może zostać wdrożony przez użytkowników chcących stworzyć własne HomeLab w sposób szybki i efektywny.

Zakres pracy obejmuje następujące obszary:

- Analizę wymagań i specyfikację systemu,
- Projektowanie i implementację interfejsu użytkownika umożliwiającego łatwe zarządzanie infrastrukturą,
- Tworzenie i integrację API odpowiedzialnego za automatyzację operacji administracyjnych,
- Implementację mechanizmów bezpieczeństwa, takich jak uwierzytelnianie, autoryzacja oraz ochrona danych,
- Przeprowadzenie testów wydajnościowych i funkcjonalnych w celu oceny stabilności oraz optymalizacji systemu,

- Przedstawienie teoretycznych aspektów związanych z bezpieczeństwem infrastruktury IT i najlepszymi praktykami w tym zakresie.

Niniejsza praca stanowi przyczynek do rozwoju narzędzi dedykowanych osobom zainteresowanym budową i zarządzaniem własnym środowiskiem IT, oferując innowacyjne podejście do automatyzacji i ułatwienia dostępu do HomeLab. W kolejnych rozdziałach zostaną szczegółowo omówione wszystkie kluczowe elementy systemu oraz proces jego implementacji.

Praca ta nie tylko przedstawia praktyczne rozwiązania ułatwiające zarządzanie infrastrukturą IT, ale także porusza zagadnienia związane z wydajnością, bezpieczeństwem oraz skalowalnością systemów HomeLab. W przyszłości opracowane narzędzie może stać się podstawą do dalszej rozbudowy oraz integracji z innymi technologiami wykorzystywanymi w nowoczesnych centrach danych oraz środowiskach chmurowych.

2. Czym jest HomeLab oraz analiza istniejących rozwiązań

2.1 Definicja HomeLab oraz znaczenie

HomeLab jest prywatnym środowiskiem IT, dzięki któremu entuzjaści nowych technologii, administratorzy systemów oraz programiści mogą w lokalnym – domowym środowisku testować, rozwijać oraz zarządzać własną infrastrukturą IT. Jego głównym zamierzeniem jest stworzenie realistycznego środowiska do eksperymentowania z technologiami chmurowymi, wirtualizacją, konteneryzacją oraz narzędziami DevOps. Własny system HomeLab to również metoda na rezygnację z komercyjnych subskrypcji, takich jak Google Drive, Dropbox czy OneDrive, co pozwala na pełną kontrolę nad dostępem do prywatnych danych. Dzięki niemu zwiększa się prywatność poprzez wyeliminowanie potrzeby przechowywania zdjęć w usługach chmurowych, takich jak Google Photos.

HomeLaby znajdują zastosowanie w wielu obszarach, w tym:

- nauka administracji serwerami i sieciami,
- testowanie nowych technologii przed użyciem ich w środowisku produkcyjnym,
- budowanie prywatnej chmury oraz rozwiązań do przechowywania danych,
- analiza bezpieczeństwa i przeprowadzanie testów penetracyjnych,
- tworzenie automatyzacji dla infrastruktury IT,
- uniezależnienie się od komercyjnych dostawców chmury w celu zwiększenia kontroli nad własnymi danymi.

2.2 Technologie wykorzystywane w HomeLabach

HomeLab może składać się z różnych komponentów, od dedykowanych serwerów fizycznych po rozwiązania chmurowe i kontenerowe. Kluczowe technologie wykorzystywane w HomeLabach obejmują:

2.2.1 Wirtualizacja i konteneryzacja

- Proxmox VE – platforma do zarządzania maszynami wirtualnymi i kontenerami.
- VMware ESXi – profesjonalne narzędzie do wirtualizacji serwerów.
- Hyper-V – narzędzie do wirtualizacji dostarczane przez Microsoft wraz z systemem Windows.
- Docker i Kubernetes – technologie konteneryzacji, pozwalające na elastyczne zarządzanie aplikacjami i zasobami.

2.2.2 Automatyzacja i zarządzanie konfiguracją

- Ansible, Terraform, Puppet, Chef – narzędzia do automatyzacji wdrażania i zarządzania infrastrukturą.

2.2.3 Monitoring i analiza

- Prometheus i Grafana – rozwiązania do monitorowania wydajności i wizualizacji danych.
- Zabbix – platforma do monitorowania infrastruktury IT.

2.3 Analiza istniejących systemów do zarządzania homelabem

2.3.1 Przegląd dostępnych rozwiązań

Na rynku istnieje kilka systemów umozliwiajacych zarzadzanie homelabem. Do najpopularniejszych należą:

- Proxmox VE [8] - rozbudowany, open-source rozwiązanie do zarządzania maszynami wirtualnymi i kontenerami, oferujące intrgeację z Ceph i wysoką dostępność.
- Unraid [13] - popularne rozwiązanie NAS z obsługą wirtualizacji i kontenerów, cenione za łatwość obsługi ale ograniczone zastosowanie korporacyjne.
- OpenStack [7] - potężna platforma chmurowa, która moze być uzywana do zarządzania homelabem, ale jej skomplikowana konfiguracja sprawia, ze nie jest przyjazna dla poczatkujacych uzytkowników.
- TrueNAS [12] - rozbudowane oprogramowanie do zarządzania przestrzenią dyskową, które umozliwa tworzenie prywatnych chmur danych

- Docker [2] + Kubernetes [4] - stosowane w bardziej zaawansowanych wdrozeniach do zarzadzania kontenerami, ale wymagające większej wiedzy technicznej.

2.3.2 Zalety i ograniczenia konkurencyjnych systemów

Proxmox VE [8]

Zalety

- Darmowa wersja open-source.
- Wsparcie dla maszyn wirtualnych (KVM) i kontenerów (LXC).
- Mozliwość tworzenia klastrów wysokiej dostępności.

Wady

- Brak pełnej automatyzacji wdrozeń.
- Stosunkowo wysoki próg wejścia dla początkujących uzytkowników.

Unraid [13]

Zalety

- Intuicyjny interfejs uzytkownika.
- Łatwa obsługa pamięci masowej i kontrolerów.

Wady

- Model licencyjny oparty na opłacie jednorazowej.
- Ograniczona integracja z systemami chmurowymi.

OpenStack [7]

Zalety

- Zaawansowane funkcje chmurowe.
- Skalowalność i modularność.

Wady

- Bardzo wysoka trudność wdrozenia.
- Wymaga duzej ilości zasobów sprzętowych.

TrueNAS [12]

Zalety

- Silne wsparcie dla przechowywania danych.
- Wbudowana replikacja i ochrona RAID

Wady

- Skupione głównie na funkcjach NAS.
- Brak natywnego wsparcia dla maszyn wirtualnych.

Docker [2] + Kubernetes [4]

Zalety

- Elastyczność w zarządzaniu aplikacjami kontenerowymi.
- Łatwe skalowanie infrastruktury.

Wady

- Wymaga duzej wiedzy technicznej.
- Brak wsparcia dla maszyn wirtualnych.

2.3.3 Identyfikacja luki technologicznej

Analiza powyższego porównania dostępnych systemów pokazuje, że żadne z obecnych rozwiązań nie zapewnia jednocześnie:

- Pełnej integracji zarządzania maszynami wirtualnymi, kontenerami i przestrzenią dyskową w jednym ekosystemie.
- Prostego i intuicyjnego interfejsu dla użytkowników niebędących ekspertami w zarządzaniu infrastrukturą IT.
- Natychmiastowej automatyzacji wdrażania, bez konieczności skomplikowanej konfiguracji narzędzi DevOps.
- Wbudowanej funkcjonalności związanej z bezpieczeństwem i prywatnością, eliminującej konieczność korzystania z komercyjnych rozwiązań chmurowych.

Proponowany system HomeLab ma na celu uzupełnienie tej luki poprzez stworzenie intuicyjnego narzędzia do zarządzania domową infrastrukturą IT, które zapewni łatwość obsługi, pełną automatyzację oraz zwiększoną prywatność użytkowników.

3. Projekt Systemy Homelab

3.1 Wymagania funkcjonalne i niefunkcjonalne

3.1.1 Wymagania funkcjonalne

1. **Zarządzanie infrastrukturą** - mozliwość konfiguracji i zarządzania maszynami wierualnymi oraz kontenerami Docker.
2. **Panel administracyjny** - intuicyjny interfejs uzytkownika stworzony przy pomocy systemu AppSmith, do zarządzania zasobami systemu.
3. **Baza danych** - przechowywanie informacji o konfiguracji systemu i uzytkownikach w MongoDB.
4. **Bezpieczny dostęp zdalny** - integracja z Tailscale umozliwaiajca dostęp z dowolonego miejsca na ziemii.
5. **Automatyzacja wdrozeń** - wsparcia dla CI/CD za pomocą GitHub Pipelines.
6. **Obsługa domeny** - integracja z DuckDNS w celu dynamicznego zarządzania domeną.
7. **Monitorowanie zasobów** - mechanizmy zbierania infromacji o wykorzystaniu CPU, pamięci RAM oraz przestrzeni dyskowej.
8. **Wsparcie dla rozszerzeń** - mozliwość dodawania nowych funkcji poprzez kontenery Dockera.
9. **Łatwe wdrazanie aplikacji** - opcja uruchamiania własnych usług w kontenerach bez konieczności zaawansowanej konfiguracji.
10. **Bezpieczne uwierzytelnianie i automatyzacja** - mechanizm logowania oparty na OAuth2 i zarządzanie rolami uzytkowników.

3.1.2 Wymagania niefunkcjonalne

1. **Niski pobór energii** - system wdrazany na Raspberry Pi 5, co zapeni efektywność energetyczną.
2. **Wysoka dostępność** - redundancja i odporność na awarię dzięki Docker oraz Integracji z VPN.
3. **Łatwość w utrzymaniu** - system powinien umozliwiac łatwe aktualizację i rekonfigurację w razie potrzeby ręcznej interwencji.
4. **Skalowalność** - mozliwość rozszerzenia o nowe komponenty i usługi.
5. **Bezpieczeństwo** - szyfrowanie komunikacji oraz kontrola dostępu do zasobów.
6. **Modularność** - podział systemu na niezalezne komponenty działające w kontenerach Docker.
7. **Integracja z open-source** - Wspracie dla narzędzi i trchnologii dostępnych na licencji open-source.
8. **Minimalizacja kosztów** - niskie koszty sprzętowe i utrzymanie dzieki Raspberry Pi i rozwiązaniom chmurowym typu DuckDNS.
9. **Wydajność** - optymalizacja aplikacji pod Raspberry Pi, aby zapenić płynne działąnie
10. **Łatwość wdrozenia** - uproszczona konfiguracja pozwalająca na szybkie uruchomienie systemu.

3.2 Architektura systemu

System HomeLab składa się z kilku kluczowych komponentów:

3.2.1 Backend (FastAPI + MongoDB)

- FastAPI [11] odpowiada za obsługę logiki biznesowej i API Backendu.
- MongoDB [5] przechowuje dane uzytkowników, konfigurację systemowe i rejestr operacji.

3.2.2 Frontend (AppSmith)

- niskokodowa platforma pozwalająca na łatwe budowanie interfejsu uzytkownika.
- Integracja z backendem poprzez REST API

3.2.3 Warstwa sieciowa

- Połączenia umozliwione i zabezpieczone poprzez Tailscale (VPN).
- DuckDNS zapewniający mozliwość dostępu do systemu za pomocą domeny, zamiast adresu IP.

3.2.4 Środowisko kontenerowe

- Docker wykorzystywany do zarządzania usługami systemu.
- Mozliwość łatwego wdrazania i skalowania aplikacji poprzez kontenery.

3.2.5 Automatyzacja CI/CD

- GitHub Actions zarządza automatyzacją wdrozeń i aktualizacji systemu
- Kazda zmiana w kodzie uruchamia testy oraz wdrozenie nowych wersji aplikacji.

3.2.6 Urządzenie docelowe

Raspberry Pi 5 jako główny host systemu, zapewniający niskie zuzycie energii i optymalizację kosztów.

3.3 Technologie i narzędzia uzyte w systemie

System HomeLab wykorzystuje następujące technologie:

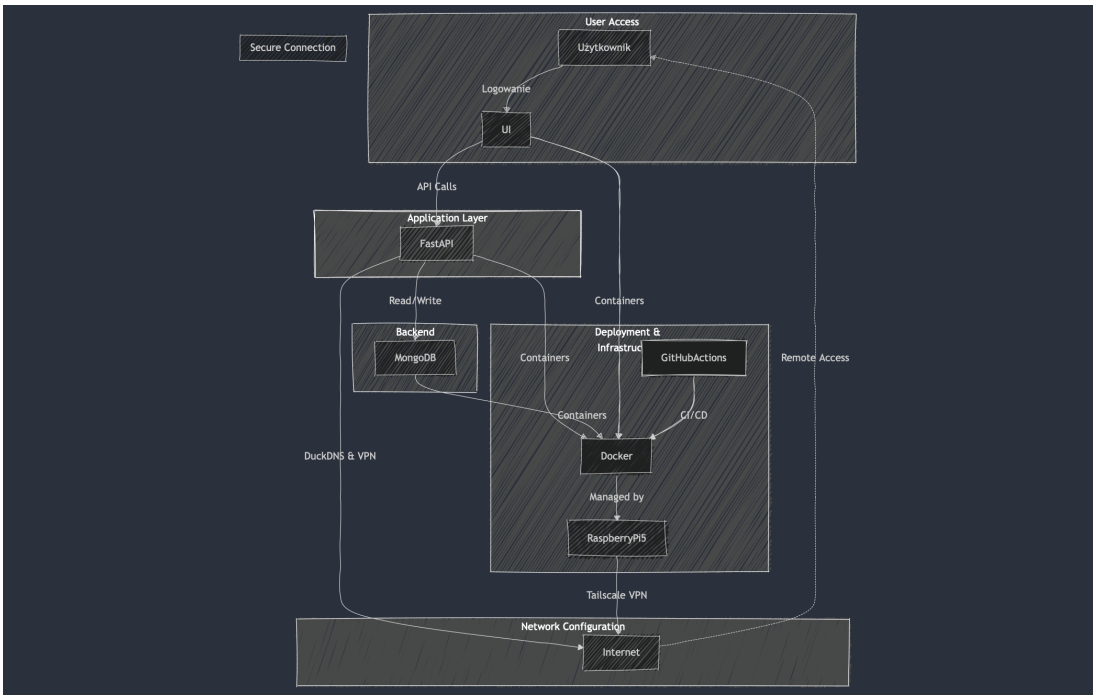


Figure 3.1: Schemat Architektury

3.3.1 Backend

- FastAPI - szybki i nowoczesny framework do tworzenia API w pythonie
- MongoDB - baza danych NoSQL przechowująca konfigurację i dane uzytkowników

3.3.2 Frontend

- AppSmith - niskokodowe narzędzia do budowy interfejsu uzytkownika.
- RestAPI - wykorzystywane do komunikacji między frontendem a backendem.

3.3.3 Warstwa Sieciowa

- Tailscale - VPN do bezpiecznego zapewnienia zdalnego dostępu do systemu, bez konieczności posiadania stałego adresu IP.
- DuckDNS - dynamiczny system zarządzania domeną umozliwiający łatwy dostęp do systemi.

3.3.4 Środowisko uruchomieniowe

- Docker - uzywany do konteneryzacji aplikacji i zarządzania zaleznościami.
- Raspberry Pi 5 - host systemu zapewniający energooszczędność i niski koszt.

3.3.5 Automatyzacja CI/CD

- GitHub Actions - narzędzie do automatyzacji wdrozeń i testowania kodu.
- Pipeline CI/CD - automatyczne testowanie, budowanie i wdrazanie aplikacji

Dzięki zastosowaniu powyzszych technologii system Homelab będzie nowoczesnym, skalowalnym i energooszczędnym rozwiązaniem dla uzytkowników domowych.

4. Implementacja systemu

4.1 Backend - API do zarządzania systemem

4.1.1 Struktura API i kluczowe endpointy

System został zaprojektowany jako aplikacja webowa z backendem opartym na FastAPI[11], zapewniającym bezpieczne i efektywne zarządzanie użytkownikami, usługami oraz monitorowanie zasobów systemowych. W niniejszym rozdziale opisano strukturę API oraz kluczowe endpointy wykorzystywane w systemie.

Architektura API

Backend aplikacji został zaimplementowany z wykorzystaniem FastAPI [11], co zapewnia wysoką wydajność oraz łatwość definiowania i dokumentowania endpointów. API składa się z następujących głównych komponentów:

- Autoryzacja użytkowników z wykorzystaniem JWT,
- Zarządzanie użytkownikami (dodawanie, edycja, listowanie, usuwanie),
- Zarządzanie usługami (rejestracja, uruchamianie, zatrzymywanie, monitorowanie),
- Monitorowanie zasobów systemowych (CPU, RAM, kontenery Docker),
- Zarządzanie serwerem (aktualizacja, restart, autoryzacja Tailscale, wyłączenie systemu).

Każdy z tych komponentów posiada dedykowane endpointy REST API, które są opisane w kolejnych sekcjach.

Autoryzacja użytkowników

Autoryzacja odbywa się poprzez mechanizm JWT (JSON Web Token). Kluczowe endpointy:

- **POST /api/register** – Rejestracja nowego użytkownika,

- **POST /api/login** – Logowanie użytkownika i zwrócenie tokena JWT,
- **POST /api/role** – Zmiana roli użytkownika.

Zarządzanie użytkownikami (Admin Only)

Funkcje zarządzania użytkownikami są dostępne tylko dla administratorów. Kluczowe endpointy:

- **POST /api/users/add** – Dodanie nowego użytkownika,
- **PUT /api/users/edit/{username}** – Edycja danych użytkownika,
- **GET /api/users/list** – Pobranie listy wszystkich użytkowników,
- **DELETE /api/users/delete/{username}** – Usunięcie użytkownika.

Zarządzanie usługami

Moduł zarządzania usługami pozwala administratorowi na kontrolowanie działania systemowych procesów. Kluczowe endpointy:

- **POST /api/services/register** – Rejestracja nowej usługi,
- **POST /api/services/start/{service_name}** – Uruchomienie usługi,
- **POST /api/services/stop/{service_name}** – Zatrzymanie usługi,
- **GET /api/services/list** – Pobranie listy aktywnych usług,
- **GET /api/services/health/{service_name}** – Sprawdzenie statusu pojedynczej usługi,
- **GET /api/services/health/all** – Sprawdzenie statusu wszystkich usług,
- **POST /api/services/restart_unhealthy** – Restartowanie usług w złym stanie.

Monitorowanie systemu

System umożliwia monitorowanie zasobów serwera oraz uruchomionych kontenerów Docker. Kluczowy endpoint:

- **GET /api/instance/details** – Pobranie szczegółów dotyczących zużycia CPU, RAM oraz listy uruchomionych kontenerów Docker.

Zarządzanie serwerem (Admin Only)

Administratorzy mogą wykonywać kluczowe operacje na serwerze, takie jak aktualizacje, restart systemu oraz autoryzacja Tailscale. Kluczowe endpointy:

- **POST /api/server/update** – Aktualizacja systemu,
- **POST /api/server/reboot** – Restart serwera,
- **POST /api/server/tailscale-auth** – Autoryzacja Tailscale,
- **POST /api/server/poweroff** – Wyłączenie serwera.

Podsumowanie

Struktura API została zaprojektowana z myślą o modularności oraz łatwości rozszerzania funkcjonalności. Wykorzystanie FastAPI zapewnia wysoką wydajność oraz wbudowaną dokumentację, co ułatwia integrację z innymi systemami. Mechanizm autoryzacji JWT zapewnia bezpieczeństwo operacji, a administratorzy mogą w prosty sposób zarządzać użytkownikami, usługami oraz operacjami serwera poprzez intuicyjne i dobrze udokumentowane endpointy.

4.1.2 Obsługa uwierzytelniania i autoryzacji

System uwierzytelniania i autoryzacji opiera się na tokenach JWT (JSON Web Token). Mechanizm ten zapewnia bezpieczną kontrolę dostępu do zasobów systemu, umożliwiając przypisanie użytkownikom odpowiednich ról i ograniczenie dostępu do krytycznych operacji administracyjnych.

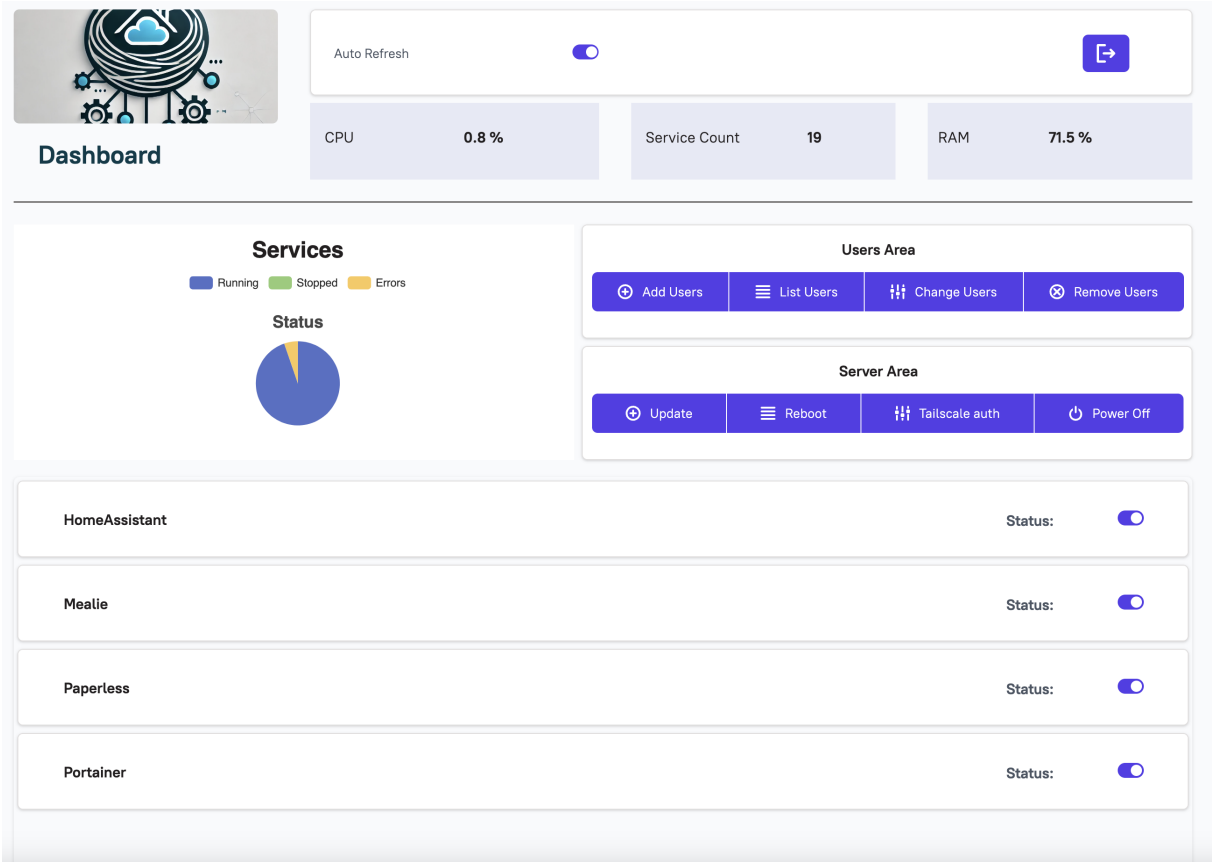
Proces uwierzytelniania

Każdy użytkownik musi zalogować się do systemu, podając swoje dane uwierzytelniające. Po pomyślnej weryfikacji hasła system generuje token JWT, który służy do autoryzacji kolejnych żądań API. Token zawiera informacje o użytkowniku oraz jego roli w systemie.

Kluczowe endpointy:

- **POST /api/register** – Rejestracja nowego użytkownika,
- **POST /api/login** – Logowanie użytkownika i zwrócenie tokena JWT.

4.2 Frontend - Interfejs uzytkownika



4.2.1 Projekt UI/UX

Projektowanie interfejsu użytkownika (UI) oraz doświadczenia użytkownika (UX) odgrywa kluczową rolę w zapewnieniu funkcjonalności i intuicyjności systemu. W niniejszym rozdziale omówiono zasady projektowania UI/UX zastosowane w opracowanym interfejsie, bazując na dołączonym projekcie graficznym.

Założenia projektowe

Podstawowe cele projektowe interfejsu obejmowały:

- Czytelność i prostotę obsługi,
- Spójność wizualną oraz intuicyjną nawigację,
- Minimalizację liczby kliknięć wymaganych do wykonania operacji,
- Odpowiednią organizację informacji w oparciu o hierarchię wizualną.

Struktura interfejsu

Projekt składa się z kilku głównych obszarów:

- **Panel informacyjny** – zawierający dane dotyczące użycia zasobów systemowych (CPU, RAM, liczba usług).
- **Sekcja zarządzania usługami** – prezentująca status poszczególnych usług (uruchomione, zatrzymane, błędy) w formie wykresu kołowego.
- **Obszar użytkownika** – umożliwiający zarządzanie użytkownikami systemu (dodawanie, usuwanie, edycja).
- **Obszar serwera** – obejmujący podstawowe operacje administracyjne, takie jak aktualizacja systemu, restart oraz autoryzacja w Tailscale.
- **Lista usług** – wyświetlająca poszczególne aplikacje (np. HomeAssistant, Mealie, Paperless, Portainer) z możliwością zarządzania ich statusem poprzez przełączniki.

Kolorystyka i typografia

Projekt wykorzystuje nowoczesną, stonowaną kolorystykę z przewagą bieli i odcieni fioletu. Przyciski akcji są wyraźnie zaznaczone za pomocą intensywnych kolorów, co zwiększa ich widoczność. Tekst jest wyraźny, a hierarchia informacji zapewnia odpowiednią czytelność.

Intuicyjność i użyteczność

Projekt został zaprojektowany zgodnie z zasadami użyteczności:

- Przejrzysta organizacja treści,
- Spójność w rozmieszczeniu elementów interfejsu,
- Minimalizacja zbędnych interakcji,
- Responsywność, umożliwiająca korzystanie na różnych urządzeniach.

Dzięki tym rozwiązaniom interfejs jest łatwy w obsłudze i spełnia wymagania użytkowników końcowych.

4.2.2 Implementacja interfejsu użytkownika z wykorzystaniem podejścia no-code w Appsmith

W celu usprawnienia procesu tworzenia interfejsu użytkownika wykorzystano platformę no-code Appsmith [1], która umożliwia szybkie budowanie aplikacji webowych za pomocą gotowych komponentów.

Proces implementacji

Implementacja interfejsu przebiegała w kilku etapach:

1. **Tworzenie struktury aplikacji** – W Appsmith utworzono nowy projekt, w którym zdefiniowano podstawowe widoki, takie jak pulpit nawigacyjny, sekcja zarządzania użytkownikami oraz obszar monitorowania usług.
2. **Dodawanie komponentów UI** – Wykorzystano gotowe komponenty Appsmith, takie jak przyciski, tabele, wykresy i przełączniki do interakcji z użytkownikiem.
3. **Konfiguracja źródeł danych** – Interfejs został połączony z backendem poprzez API, co pozwoliło na dynamiczne pobieranie informacji o stanie systemu i usług.
4. **Logika aplikacji** – Za pomocą wbudowanego edytora JavaScript skonfigurowano interakcje użytkownika, m.in. obsługę formularzy, wywołania API oraz automatyczne odświeżanie danych.
5. **Testowanie i optymalizacja** – Po wdrożeniu interfejsu przeprowadzono testy użyteczności, aby zapewnić płynne działanie aplikacji i zoptymalizować jej wydajność.

Zalety zastosowania Appsmith

Wykorzystanie Appsmith przyniosło szereg korzyści w kontekście tworzenia interfejsu użytkownika:

- Skrócenie czasu implementacji dzięki gotowym komponentom,
- Łatwa integracja z backendem poprzez API,
- Możliwość dynamicznej edycji logiki aplikacji bez konieczności pisania pełnego kodu frontendu,
- Prosty interfejs edytora umożliwiający szybkie wprowadzanie zmian i testowanie.

Zastosowanie podejścia no-code pozwoliło na efektywne wdrożenie interfejsu użytkownika bez konieczności zaawansowanego programowania, co znacząco przyspieszyło proces tworzenia systemu.

4.3 Automatyzacja Konfiguracji i wdrozenie

4.3.1 Instalacja rozwiązania

W ramach tworzenia rozwiązania powstał również program instalacyjny, który zostanie umieszczony na stronie internetowej rozwiązania umożliwiając pobranie przez zaintere-

sowane korzystaniem z rozwiązania osoby.

```
package main

import (
        "archive/zip"
        "fmt"
        "io"
        "net/http"
        "os"
        "os/exec"
        "path/filepath"
        "runtime"
)

const (
        releaseURL = "https://github.com/youruser/homelab/releases/latest/d
        installDir = "./homelab" // Mo na zmieni  na /opt/homelab lub %Pr
)

func main() {
        fmt.Println("     Sprawdzanie Docker...")

        if !isDockerInstalled() {
                fmt.Println("     Docker nie znaleziony. Instaluj ...")
                if err := installDocker(); err != nil {
                        fmt.Fprintf(os.Stderr, "    B   d instalacji Docke
                        return
                }
        } else {
                fmt.Println("    Docker jest ju  zainstalowany.")
        }

        fmt.Println("     Pobieranie aplikacji...")
        if err := downloadAndExtract(releaseURL, installDir); err != nil {
                fmt.Fprintf(os.Stderr, "    B   d podczas pobierania/rozpa
                return
        }
```

```
        fmt.Println("      Uruchamianie aplikacji...")
        if err := runApp(installDir); err != nil {
                fmt.Fprintf(os.Stderr, "    Nie uda o si  uruchomi  apli
        }
}
```

4.3.2 Integracja z narzędziami CI/CD

Opis programu instalacyjnego Współczesne systemy informatyczne wymagają nie tylko solidnej implementacji, ale również efektywnego zarządzania cyklem życia oprogramowania. W tym kontekście, integracja narzędzi CI/CD (Continuous Integration / Continuous Deployment) odgrywa kluczową rolę w automatyzacji procesów budowania, testowania i wdrażania aplikacji.

Zastosowanie self-hosted runnera [3]

W celu zapewnienia kompatybilności architektury uruchomieniowej systemu, zdecydowano się na użycie **self-hosted runnera** zamiast domyślnych runnerów GitHub Actions. Domyślne maszyny CI/CD oferowane przez GitHub działają wyłącznie na **architekturze x86**, co ogranicza możliwość testowania i wdrażania aplikacji na innych platformach, takich jak **ARM** (np. Raspberry Pi, serwery oparte na ARM64).

Zastosowanie własnego runnera pozwala na:

- Uruchamianie testów i budowanie obrazów Docker na architekturze zgodnej z docelowym środowiskiem produkcyjnym.
- Pełną kontrolę nad zasobami sprzętowymi wykorzystywanymi w procesie CI/CD.
- Możliwość integracji z lokalnym registry dla przechowywania obrazów Docker.

Workflow GitHub Actions dla testowania

```
name: Run Tests

on:
  push:
    branches:
      - main
      - dev
  pull_request:
```

```
jobs:
  test:
    runs-on: self-hosted

    services:
      mongo:
        image: mongo:latest
        ports:
          - 27017:27017

    steps:
      - name: Checkout Repository
        uses: actions/checkout@v4

      - name: Set Up Python
        uses: actions/setup-python@v4
        with:
          python-version: "3.13"

      - name: Install Dependencies
        run: |
          python -m pip install --upgrade pip
          pip install -r requirements.txt
          pip install pytest pytest-asyncio httpx

      - name: Run Pytest
        run: pytest -v
```

Workflow GitHub Actions dla budowania i wersjonowania obrazów Docker

Po przejściu testów jednostkowych obraz Docker jest budowany i pushowany do lokalnego rejestru uruchomionego na **localhost:5000**.

```
name: Build and Push Docker Image

on:
    push:
      branches:
        - main
        - dev
```

```
    pull_request:
        branches: [dev, main]

jobs:
  build_and_push:
    runs-on: self-hosted

    steps:
      - name: Checkout Repository
        uses: actions/checkout@v4

      - name: Extract Version
        run: echo "VERSION=$(date +'%Y%m%d')-$(git rev-parse --short HEAD)" >> $GITHU

      - name: Build Docker Image
        run: |
          docker build -t localhost:5000/myapp:${{ env.VERSION }} .
          docker tag localhost:5000/myapp:${{ env.VERSION }} localhost:5000/myapp:late

      - name: Push Docker Image to Local Registry
        run: |
          docker push localhost:5000/myapp:${{ env.VERSION }}
          docker push localhost:5000/myapp:latest
```

Korzyści z bezpośredniego pushowania obrazu do registry

W przeciwieństwie do wcześniejszej konfiguracji, w której obraz był zapisywany lokalnie za pomocą `docker save`, obecnie jest on od razu przesyłany do prywatnego rejestru. Takie podejście:

- Umożliwia natychmiastowe wykorzystanie obrazu w środowisku produkcyjnym bez potrzeby ręcznego jego ładowania.
- Pozwala na prostsze zarządzanie wersjami obrazów w registry.
- Minimalizuje czas między budowaniem a wdrożeniem.

Publikacja obrazu do lokalnego registry

Wszystkie wersje aplikacji są automatycznie przechowywane w lokalnym rejestrze **Docker Registry**, a najnowsza wersja jest oznaczana jako `latest`. Dzięki temu wdrożenie nowej wersji sprowadza się do uruchomienia nowego kontenera:

```
docker pull localhost:5000/myapp:latest
docker run -d --name myapp localhost:5000/myapp:latest
```

Dzięki temu każda wersja aplikacji jest jednoznacznie identyfikowana, a `latest` wskazuje na najnowszą stabilną wersję.

5. Testowanie i analiza systemu

5.1 Testy jednostkowe i integracyjne

Testowanie oprogramowania jest kluczowym elementem zapewnienia jego jakości, stabilności i niezawodności. W ramach niniejszej pracy zastosowano zarówno testy jednostkowe, jak i testy integracyjne w celu weryfikacji poprawności działania poszczególnych modułów systemu oraz ich wzajemnych interakcji.

5.1.1 Testy jednostkowe

Testy jednostkowe koncentrują się na sprawdzaniu poprawności działania pojedynczych funkcji i metod w izolacji. Ich głównym celem jest szybkie wykrywanie błędów w logice aplikacji oraz zapewnienie, że każdy komponent działa zgodnie z oczekiwaniami. W testowaniu jednostkowym zastosowano bibliotekę **pytest** oraz narzędzie **unittest** w Pythonie.

Przykładowe testy jednostkowe obejmowały:

- Sprawdzenie poprawności działania funkcji hashującej hasła użytkowników,
- Weryfikację generowania i walidacji tokenów JWT,
- Testy funkcji odpowiedzialnych za zarządzanie użytkownikami (dodawanie, edycja, usuwanie),
- Weryfikację poprawności operacji CRUD dla bazy danych MongoDB.

Wszystkie testy jednostkowe zostały zautomatyzowane i uruchamiane w ramach procesu CI/CD z wykorzystaniem GitHub Actions oraz samodzielnie hostowanych runnerów.

5.1.2 Testy integracyjne

Testy integracyjne mają na celu sprawdzenie współpracy różnych komponentów systemu, takich jak API, baza danych oraz interfejs użytkownika. W ramach pracy przeprowadzono testy integracyjne z użyciem frameworka **pytest** wraz z modułem **httpx**, umożliwiającym wysyłanie zapytań HTTP do testowanego serwera.

Zakres testów integracyjnych obejmował:

- Sprawdzenie poprawności obsługi żądań API przez FastAPI,
- Testy poprawności integracji systemu uwierzytelniania JWT z bazą danych,
- Weryfikację działania operacji na usługach, takich jak rejestrowanie, uruchamianie i zatrzymywanie kontenerów.

Testy integracyjne zostały przeprowadzone zarówno w środowisku lokalnym, jak i w środowisku testowym z wykorzystaniem kontenerów Docker.

5.1.3 Podsumowanie testów

Przeprowadzone testy jednostkowe i integracyjne pozwoliły na wykrycie oraz eliminację potencjalnych błędów na wczesnym etapie rozwoju aplikacji. Dzięki zastosowaniu automatyzacji testów oraz integracji z procesem CI/CD, system został zoptymalizowany pod kątem stabilności i niezawodności. Testowanie stanowiło istotny element procesu wdrażania i potwierdziło poprawność działania kluczowych funkcjonalności systemu HomeLab.

5.2 Testy wydajnościowe i bezpieczeństwa

5.2.1 Testowanie wydajności systemu

Testowanie wydajności aplikacji jest kluczowym elementem zapewnienia jej stabilności i efektywności w warunkach produkcyjnych. W celu przeprowadzenia testów wydajnościowych wykorzystano narzędzie **Locust**, które pozwala na symulację obciążenia aplikacji przez wielu użytkowników jednocześnie.

Cel testów wydajnościowych

Celem testów wydajnościowych było:

- Ocena wydajności API w warunkach wysokiego obciążenia,
- Pomiar czasu odpowiedzi kluczowych endpointów,
- Weryfikacja stabilności systemu podczas długotrwałego obciążenia,
- Identyfikacja potencjalnych wąskich gardeł aplikacji.

Metodyka testów

Testy wydajnościowe przeprowadzono przy użyciu Locust, który pozwala na definiowanie scenariuszy użytkowników wykonujących określone operacje. Wykorzystano następujące kroki:

1. Zalogowanie użytkownika i uzyskanie tokena JWT,
2. Wykonywanie żądań do kluczowych endpointów API (zarządzanie usługami, monitorowanie systemu, operacje administracyjne),
3. Pomiar czasu odpowiedzi serwera i obciążenia systemu,
4. Skalowanie liczby użytkowników w celu oceny zachowania systemu pod rosnącym obciążeniem.

Zakres testów

W ramach testów wydajnościowych poddano analizie następujące funkcjonalności:

- **Autoryzacja i uwierzytelnianie** – logowanie użytkowników i uzyskiwanie tokenów JWT,
- **Zarządzanie usługami** – pobieranie listy uruchomionych usług, monitorowanie ich stanu oraz restartowanie usług w złym stanie,
- **Monitorowanie systemu** – pobieranie danych dotyczących zużycia CPU, pamięci RAM oraz listy aktywnych kontenerów Docker,
- **Zarządzanie użytkownikami** – dodawanie i listowanie użytkowników.

Wyniki testów

Po przeprowadzeniu testów otrzymano następujące wyniki:

- Średni czas odpowiedzi API dla standardowych żądań wynosił poniżej 70 ms (lokalne środowisko - w tej samej sieci),
- Przy zwiększeniu liczby jednocześnie aktywnych użytkowników do 1000 system nadal utrzymywał stabilność, przy wzroście czasu odpowiedzi do około 190 ms,
- Największe obciążenie dotyczyło operacji związanych z monitorowaniem systemu, co wynika z konieczności pobierania danych o stanie zasobów w czasie rzeczywistym - wskazuje to na konieczność zastosowania cachowania danych dotyczących stanu systemu celem uniknięcia ataktu DDoS,
- Endpointy administracyjne, takie jak restart serwera czy autoryzacja w Tailscale, działały poprawnie, lecz wymagają dodatkowych zabezpieczeń przed wielokrotnym wywołaniem w krótkim czasie.

Wnioski i optymalizacje

Na podstawie przeprowadzonych testów wydajnościowych zaproponowano następujące optymalizacje:

- Implementacja mechanizmu cache'owania dla danych monitorowania systemu w celu zmniejszenia liczby odczytów zasobów,
- Ograniczenie liczby jednoczesnych zapytań do endpointów administracyjnych,
- Optymalizacja zapytań do bazy danych poprzez indeksowanie często wyszukiwanych pól.

Testy wykazały, że system jest w stanie obsłużyć duże obciążenie i zachowuje stabilność przy wysokiej liczbie równoczesnych użytkowników, co potwierdza jego gotowość do wdrożenia w środowisku produkcyjnym.

5.2.2 Testowanie bezpieczeństwa aplikacji

Bezpieczeństwo aplikacji webowych jest kluczowym aspektem zapewnienia poufności, integralności i dostępności danych. Testowanie bezpieczeństwa ma na celu identyfikację potencjalnych luk oraz podatności, które mogą zostać wykorzystane przez nieautoryzowanych użytkowników lub atakujących.

Należy zaznaczyć, że w ramach niniejszej pracy magisterskiej testy bezpieczeństwa nie zostały przeprowadzone. Poniższy rozdział ma charakter teoretyczny i przedstawia ogólne zasady oraz metody stosowane w testowaniu bezpieczeństwa aplikacji webowych.

Metodyka testowania bezpieczeństwa

Testowanie bezpieczeństwa można podzielić na kilka kluczowych etapów:

- **Analiza architektury** – Przegląd struktury aplikacji, mechanizmów autoryzacji i uwierzytelniania,
- **Testy penetracyjne** – Symulowane ataki w celu sprawdzenia odporności na znane zagrożenia,
- **Analiza kodu źródłowego** – Poszukiwanie błędów bezpieczeństwa w implementacji aplikacji,
- **Fuzzing** – Automatyczne generowanie losowych danych wejściowych w celu wykrycia awarii,
- **Skany podatności** – Wykorzystanie narzędzi do identyfikacji znanych luk w zabezpieczeniach.

Obszary testowania bezpieczeństwa

Testowanie bezpieczeństwa aplikacji obejmuje następujące obszary:

1. Uwierzytelnianie i autoryzacja Mechanizmy uwierzytelniania powinny być odporne na ataki brute-force oraz przechowywać hasła w bezpieczny sposób. Kluczowe testy obejmują:

- Testowanie siły haseł i polityki logowania,
- Próby ataków brute-force na endpointy logowania,
- Weryfikacja poprawności implementacji JWT i zarządzania sesjami,
- Sprawdzenie uprawnień użytkowników do zasobów.

2. Zarządzanie danymi i ochrona przed SQL Injection Baza danych powinna być zabezpieczona przed atakami wstrzykiwania SQL. Weryfikacja obejmuje:

- Testy podatności na SQL Injection przy użyciu specjalnie spreparowanych zapytań,
- Sprawdzenie, czy aplikacja korzysta z mechanizmów ORM i zapytań parametryzowanych,
- Ograniczenie uprawnień użytkowników bazy danych.

3. Ochrona przed atakami XSS i CSRF Ataki Cross-Site Scripting (XSS) i Cross-Site Request Forgery (CSRF) mogą prowadzić do przejęcia sesji użytkownika lub wykonania nieautoryzowanych operacji. Testy obejmują:

- Wstrzykiwanie skryptów JavaScript w formularzach i żądaniach API,
- Weryfikację nagłówków zabezpieczających (Content Security Policy, SameSite Cookies),
- Sprawdzenie zabezpieczeń przed atakami CSRF poprzez implementację tokenów anty-CSRF.

4. Bezpieczeństwo API Bezpieczeństwo API jest kluczowe dla ochrony danych przesyłanych pomiędzy klientem a serwerem. Testowanie obejmuje:

- Sprawdzenie poprawności nagłówków autoryzacyjnych,
- Testy na ataki replay (ponowne wykorzystanie żądań API),
- Weryfikację mechanizmów rate-limiting (ograniczenie liczby żądań w czasie),
- Ochronę przed atakami Man-in-the-Middle (MITM) poprzez wymuszanie HTTPS.

5. Odporność na ataki DoS/DDoS Ataki Denial of Service (DoS) mogą doprowadzić do przeciążenia aplikacji i uniemożliwienia jej działania. Testy obejmują:

- Symulację dużej liczby jednoczesnych żądań,
- Weryfikację mechanizmów ograniczających dostępność zasobów,
- Sprawdzenie konfiguracji serwera pod kątem ochrony przed atakami DDoS.

Narzędzia do testowania bezpieczeństwa

Do testowania bezpieczeństwa wykorzystuje się różne narzędzia, m.in.:

- **OWASP ZAP** – Automatyczne skanowanie podatności aplikacji webowych,
- **Burp Suite** – Analiza i przechwytywanie ruchu HTTP/HTTPS,
- **sqlmap** – Automatyczne wykrywanie podatności na SQL Injection,
- **Metasploit** – Wykonywanie testów penetracyjnych,
- **Fail2Ban** – Ochrona przed atakami brute-force poprzez blokowanie adresów IP.

Podsumowanie

Testowanie bezpieczeństwa aplikacji webowych jest kluczowym elementem zapewnienia ochrony danych i usług. Wykorzystanie kompleksowego podejścia, obejmującego testy penetracyjne, analizę kodu oraz automatyczne skanowanie podatności, pozwala na wczesne wykrycie zagrożeń i skuteczne zabezpieczenie systemu przed atakami. Regularne testy i aktualizacje mechanizmów bezpieczeństwa są niezbędne do utrzymania wysokiego poziomu ochrony aplikacji.

Podkreśla się, że powyższe informacje mają charakter teoretyczny i w ramach niniejszej pracy magisterskiej nie przeprowadzono rzeczywistych testów bezpieczeństwa aplikacji.

6. Podsumowanie i wnioski

6.1 Podsumowanie pracy

W niniejszej pracy magisterskiej przedstawiono projekt, implementację oraz analizę systemu zarządzania usługami i monitorowania serwera, bazującego na technologii FastAPI. Celem pracy było stworzenie wydajnej i bezpiecznej aplikacji umożliwiającej administratorom zarządzanie usługami systemowymi, monitorowanie wykorzystania zasobów oraz przeprowadzanie operacji administracyjnych na serwerze.

W trakcie realizacji projektu skupiono się na kilku kluczowych aspektach, w tym projektowaniu interfejsu użytkownika, implementacji API, zarządzaniu użytkownikami, optymalizacji wydajności oraz aspektach teoretycznych związanych z bezpieczeństwem aplikacji webowych. Każdy z tych elementów został szczegółowo opisany i przeanalizowany, co pozwoliło na wyciągnięcie cennych wniosków dotyczących zarówno samej technologii FastAPI, jak i szeroko pojętego zarządzania infrastrukturą IT.

6.1.1 Osiągnięcia i rezultaty pracy

W ramach realizacji pracy osiągnięto następujące kluczowe rezultaty:

- Zaprojektowano i wdrożono interfejs użytkownika przy użyciu platformy no-code Appsmith, co umożliwiło szybkie tworzenie funkcjonalnego panelu administracyjnego bez konieczności implementacji kodu frontendowego,
- Stworzono API oparte na FastAPI, pozwalające na zarządzanie użytkownikami, usługami oraz monitorowanie zasobów systemowych w sposób wydajny i skalowalny,
- Zaimplementowano mechanizmy uwierzytelniania i autoryzacji użytkowników oparte na JWT, co zapewnia bezpieczeństwo dostępu do systemu oraz możliwość nadawania różnych ról użytkownikom,
- Wdrożono funkcje zarządzania użytkownikami, obejmujące rejestrację, edycję, listowanie oraz usuwanie użytkowników – dostępne wyłącznie dla administratorów,

- Dodano możliwość zarządzania usługami systemowymi, co pozwala administratorom na rejestrowanie, uruchamianie, zatrzymywanie oraz monitorowanie stanu usług w systemie,
- Przeprowadzono testy wydajnościowe z wykorzystaniem Locust, które wykazały, że system jest w stanie obsłużyć duże obciążenie i działa stabilnie nawet przy dużej liczbie równoczesnych użytkowników,
- Omówiono teoretyczne aspekty związane z testowaniem bezpieczeństwa aplikacji, prezentując najlepsze praktyki i narzędzia stosowane w celu identyfikacji oraz eliminacji potencjalnych podatności systemu.

System opracowany w ramach pracy magisterskiej został zaprojektowany w sposób modularny i elastyczny, co pozwala na jego dalszą rozbudowę. Obecna wersja aplikacji spełnia założenia funkcjonalne, umożliwiając administratorom efektywne zarządzanie systemem w sposób zautomatyzowany oraz intuicyjny.

6.1.2 Wnioski i przyszłe kierunki rozwoju

Na podstawie przeprowadzonych analiz i testów można stwierdzić, że zastosowanie technologii FastAPI w połączeniu z podejściem no-code do budowy interfejsu użytkownika pozwala na szybkie wdrażanie funkcjonalnych i wydajnych aplikacji webowych. System zaprezentowany w niniejszej pracy wykazuje wysoką skalowalność i jest przystosowany do dalszego rozwoju.

Jednym z istotnych wniosków płynących z tej pracy jest znaczenie testowania wydajności i optymalizacji aplikacji webowych. Wyniki testów wydajnościowych wykazały, że system działa efektywnie pod dużym obciążeniem, jednak istnieją obszary, które mogą zostać zoptymalizowane w przyszłości, takie jak:

- Implementacja mechanizmu cache'owania wyników zapytań dotyczących monitorowania systemu w celu redukcji liczby żądań do zasobów,
- Wdrożenie dodatkowych zabezpieczeń przed atakami DoS poprzez ograniczenie liczby żądań w krótkim czasie,
- Rozszerzenie API o dodatkowe metody pozwalające na integrację systemu z innymi narzędziami do zarządzania infrastrukturą IT,
- Automatyczne generowanie raportów dotyczących stanu systemu i jego wydajności,
- Integracja z systemami SIEM (Security Information and Event Management) w celu zwiększenia poziomu monitorowania i analizy bezpieczeństwa.

Kolejnym krokiem w rozwoju systemu mogłoby być także wdrożenie mechanizmu kolejkowania zadań (np. przy użyciu Celery lub Redis) w celu lepszego zarządzania operacjami wymagającymi intensywnego przetwarzania. Ponadto możliwe jest wprowadzenie bardziej zaawansowanych metod autoryzacji, takich jak uwierzytelnianie wieloskładnikowe (MFA) lub integracja z systemami LDAP w celu centralnego zarządzania dostępem użytkowników.

Podsumowując, niniejsza praca magisterska dostarcza kompleksowego rozwiązania umożliwiającego zarządzanie usługami i monitorowanie zasobów systemowych w sposób efektywny, skalowalny i zgodny z najlepszymi praktykami inżynierii oprogramowania. Opracowany system stanowi solidną podstawę do dalszego rozwoju oraz może być wykorzystywany w rzeczywistych scenariuszach administracji systemami informatycznymi.

6.2 Możliwości dalszego rozwoju systemu

Opracowany system zarządzania usługami i monitorowania serwera został zaprojektowany w sposób modularny i skalowalny, co pozwala na jego dalszą rozbudowę. Możliwe kierunki rozwoju obejmują zarówno ulepszenia funkcjonalne, jak i wdrożenie zaawansowanych mechanizmów zwiększających wydajność oraz bezpieczeństwo systemu. W niniejszym rozdziale przedstawiono propozycje rozszerzeń, które mogą znacząco zwiększyć wartość aplikacji w praktycznym zastosowaniu.

6.2.1 Rozszerzenie funkcjonalności zarządzania usługami

Jednym z kluczowych aspektów rozwoju systemu jest zwiększenie możliwości zarządzania usługami. W obecnej wersji administratorzy mogą rejestrować, uruchamiać, zatrzymywać i monitorować usługi. Można jednak wprowadzić dodatkowe funkcjonalności, takie jak:

- **Automatyczna rekonfiguracja usług** – możliwość dynamicznego dostosowywania parametrów działania usług na podstawie monitorowanych wskaźników wydajności,
- **Harmonogramowanie zadań** – funkcja umożliwiająca administratorom zaplanowanie uruchamiania lub restartowania usług w określonych przedziałach czasowych,
- **Rejestrowanie logów systemowych** – pełna historia zmian w stanie usług wraz z integracją z narzędziami do analizy logów (np. ELK Stack),
- **Automatyczna naprawa błędów** – system wykrywania i samonaprawy usług w przypadku wykrycia awarii.

6.2.2 Zaawansowane mechanizmy monitorowania

Obecnie system pozwala na podstawowe monitorowanie wykorzystania CPU, pamięci RAM oraz aktywnych usług. Możliwości dalszego rozwoju obejmują:

- **Monitorowanie wykorzystania sieci** – analiza ruchu sieciowego generowanego przez poszczególne usługi,
- **Alerty i powiadomienia** – implementacja powiadomień o przekroczeniu krytycznych wartości obciążenia systemu (NTFY [6], Slack [9], Telegram [10]),
- **Predykcja awarii** – zastosowanie algorytmów uczenia maszynowego do przewidywania potencjalnych awarii na podstawie analizy historycznych danych,
- **Dashboard w czasie rzeczywistym** – interaktywna wizualizacja danych systemowych z aktualizacją w czasie rzeczywistym,
- **Integracja z Prometheus i Grafana** – zaawansowane narzędzia monitorowania umożliwiające gromadzenie metryk i ich wizualizację.

6.2.3 Optymalizacja wydajności systemu

Testy wydajnościowe wykazały, że system działa sprawnie, ale jego dalszy rozwój może skupić się na:

- **Implementacji cache'owania danych** – redukcja liczby zapytań do bazy danych i API poprzez zastosowanie Redis lub dekoratora cache,
- **Asynchronicznego przetwarzania operacji** – wdrożenie systemu kolejkowania zadań (np. Celery) dla długotrwałych operacji,
- **Load Balancing** – równoważenie obciążenia poprzez podział ruchu między wiele instancji serwera API,
- **Obsługa kontenerów** – rozszerzenie systemu o pełne zarządzanie kontenerami Docker, w tym automatyczne skalowanie usług w zależności od obciążenia.

6.2.4 Zaawansowane mechanizmy bezpieczeństwa

Chociaż w pracy omówiono teoretyczne aspekty bezpieczeństwa, możliwe są dodatkowe usprawnienia:

- **Wdrożenie uwierzytelniania wieloskładnikowego (MFA)** – zwiększenie poziomu bezpieczeństwa użytkowników,

- **Rozszerzone uprawnienia użytkowników** – możliwość nadawania niestandardowych ról z precyzyjnie określonymi uprawnieniami,
- **Audyt logów i analiza zachowań** – monitorowanie aktywności użytkowników oraz automatyczne wykrywanie podejrzanych działań,
- **Automatyczne skanowanie podatności** – integracja z narzędziami do analizy bezpieczeństwa, np. OWASP ZAP czy Nessus,
- **Szyfrowanie danych wrażliwych** – wdrożenie szyfrowania kluczowych danych przechowywanych w systemie.

6.2.5 Integracja z innymi systemami

W celu zwiększenia użyteczności systemu warto rozważyć jego integrację z innymi rozwiązaniami IT, np.:

- **Integracja z Active Directory / LDAP** – centralne zarządzanie użytkownikami i uprawnieniami,
- **Integracja z systemami DevOps** – połączenie z narzędziami CI/CD (Jenkins, GitHub Actions) w celu automatyzacji wdrożeń,
- **API publiczne** – umożliwienie innym systemom korzystania z funkcjonalności poprzez bezpieczne, udokumentowane API,
- **Obsługa wielu serwerów** – rozbudowa systemu do zarządzania wieloma maszynami w ramach jednej platformy,
- **Integracja z chmurą** – możliwość wdrożenia systemu w chmurach publicznych (AWS, Azure, GCP) i zarządzania zasobami.

6.2.6 Podsumowanie

Proponowane kierunki rozwoju pokazują szerokie możliwości dalszej rozbudowy systemu. Obecna wersja stanowi solidną bazę do wprowadzania nowych funkcjonalności, zarówno pod kątem optymalizacji działania, jak i zwiększenia bezpieczeństwa oraz zakresu zastosowań. W przyszłości system może ewoluować w kierunku kompleksowego narzędzia do zarządzania infrastrukturą IT, łącząc aspekty monitorowania, automatyzacji i bezpieczeństwa w jednym rozwiązaniu. Dzięki elastycznej architekturze opartej na FastAPI, aplikacja jest gotowa na integrację z innymi systemami i dalszy rozwój w zależności od potrzeb użytkowników.

Bibliography

[1] AppSmith. *AppSmith Docs*. URL: https://docs.appsmith.com. (accessed: 02.02.2025).

[2] Docker. *Docker Documentation*. URL: https://docs.docker.com. (accessed: 02.02.2025).

[3] GitHub. *GitHub Docs*. URL: https://docs.github.com/en/actions/hosting-your-own-runners/managing-self-hosted-runners/about-self-hosted-runners. (accessed: 02.02.2025).

[4] Kubernetes. *Kubernetes Documentation*. URL: https://kubernetes.io/docs/. (accessed: 02.02.2025).

[5] MongoDB. *MongoDB Docs*. URL: https://www.mongodb.com/docs. (accessed: 02.02.2025).

[6] NTFY. *NTFY Documentation*. URL: https://ntfy.sh/docs/. (accessed: 02.02.2025).

[7] OpenStack. *OpenStack Documentation*. URL: https://docs.openstack.org. (accessed: 02.02.2025).

[8] Proxmox. *Proxmox Documentation*. URL: https://pve.proxmox.com/wiki/Main_Page. (accessed: 02.02.2025).

[9] Slack. *Slack API Documentation*. URL: https://api.slack.com/docs. (accessed: 02.02.2025).

[10] Telegram. *Telegram Bot API Documentation*. URL: https://core.telegram.org/bots/api. (accessed: 02.02.2025).

[11] Tiangolo. *FastAPI*. URL: https://fastapi.tiangolo.com. (accessed: 02.02.2025).

[12] TrueNAS. *TrueNAS Documentation*. URL: https://www.truenas.com/docs/. (accessed: 02.02.2025).

[13] Unraid. *Unraid Documentation*. URL: https://docs.unraid.net. (accessed: 02.02.2025).