

**PRACA DYPLOMOWA**

**WYDZIAŁ  
BUDOWY MASZYN I INFORMATYKI**

**KIERUNEK: Informatyka**

**SPECJALNOŚĆ: TECHNIKI TWORZENIA OPROGRAMOWANIA**

**Maciej Tonderski**

**nr albumu: 62572**

**Praca magisterska**

**ZINTEGROWANY SYSTEM ZARZĄDZANIA  
INFRASTRUKTURĄ IT W ŚRODOWISKU  
HOMELAB**

*Kategoria pracy: projektowa*

Promotor: dr inż. RUSLAN SHEVCHUK

Bielsko-Biała, 2025

## **Oświadczenie dyplomanta do pracy dyplomowej**

Bielsko-Biała, dnia ..... 2025 r.

(imię i nazwisko) Maciej Tonderski

(nr albumu) 62572

(kierunek studiów) Informatyka

(specjalność/specjalizacja) techniki tworzenia oprogramowania

(forma studiów) stacjonarne

## **OŚWIADCZENIE**

Oświadczam, że złożona praca końcowa pt.

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

1. Została napisana przeze mnie samodzielnie.
2. W swojej pracy korzystałem/łam z materiałów źródłowych w granicach dozwolonego użytku, wymieniając autora, tytuł pozycji i źródło jej publikacji.
3. Zamieszczam/łam krótkie fragmenty cudzych utworów w cudzysłowie, a w przypisie podałem/łam źródło tego cytatu. Dotyczy to cytatów zaczerpniętych z publikacji naukowych, takich jak książki, czasopisma, a także z wewnętrznych opracowań przedsiębiorstw, z instrukcji obsługi, prospektów reklamowych oraz z trwałych źródeł informacji w formie elektronicznej.
4. Praca nie ujawnia żadnych danych, informacji i materiałów, których publikacja nie jest prawnie dozwolona.
5. Praca nie była wcześniej podstawą żadnej innej procedury związanej z nadaniem stopni naukowych, dyplomów ani tytułów zawodowych.
6. Jestem świadomymy, że przywłaszczenie sobie autorstwa albo wprowadzenie w błąd co do autorstwa całości lub części cudzego utworu jest przestępstwem - zagrożonym na podstawie ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych - odpowiedzialnością karną.
7. Nie zamieszczam/łam w pracy fragmentów nietrwałych źródeł informacji. Przez nietrwałe źródła informacji rozumie się w szczególności informacje pozyskane za pomocą środków elektronicznych, które ze względu na swój modyfikowalny charakter, jak również brak ich przypisania do określonego wiarygodnego autora lub instytucji nie powinny stanowić rzetelnego źródła informacji stanowiącego podstawę dla realizacji pracy dyplomowej studenta.

Składając to oświadczenie, jestem świadomymy, że jeżeli moja praca narusza przepisy prawa, nie zostanie ona przez Uczelnię przyjęta. Ponadto już po obronie praca może być poddana kontroli następczej, która w przypadku naruszenia przepisów ustawy o ochronie praw autorskich i praw pokrewnych jak również przepisów szczególnych, prowadzić może do wszczęcia postępowania w przedmiocie cofnięcia jej autorowi uzyskanego tytułu zawodowego. Prawdziwość powyższego oświadczenia potwierdzam własnoręcznym podpisem.

---

(podpis studenta/studentki)

## 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 luk 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 użytkownikami, usługami oraz monitorującym system.
- Baza danych SQLite przechowująca informację o zarejestrowanych użytkownikach, prowadzonych 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

<b>1 Wprowadzenie</b>	<b>4</b>
1.1 Cel pracy . . . . .	5
1.2 Zakres pracy . . . . .	6
<b>2 Aspekty funkcjonalne i technologiczne środowiska HomeLab</b>	<b>7</b>
2.1 Definicja HomeLab oraz znaczenie . . . . .	7
2.2 Technologie wykorzystywane w HomeLabach . . . . .	8
2.3 Analiza istniejących systemów do zarządzania homelabem . . . . .	12
<b>3 Projekt Systemu Homelab</b>	<b>16</b>
3.1 Wymagania funkcjonalne i niefunkcjonalne . . . . .	16
3.2 Architektura systemu . . . . .	19
3.3 Technologie i narzędzia użyte w systemie . . . . .	27
<b>4 Implementacja systemu</b>	<b>29</b>
4.1 Backend - API do zarządzania systemem . . . . .	29
4.2 Frontend - Interfejs użytkownika . . . . .	32
4.3 Automatyzacja Konfiguracji i wdrożenie . . . . .	41
<b>5 Analiza efektywności i bezpieczeństwa systemu</b>	<b>47</b>
5.1 Testy jednostkowe i integracyjne . . . . .	47
5.2 Analiza wydajnościowa systemu . . . . .	48
5.3 Analiza odporności systemu na zagrożenia bezpieczeństwa . . . . .	51
<b>6 Podsumowanie i wnioski</b>	<b>54</b>
6.1 Podsumowanie pracy . . . . .	54
6.2 Możliwości dalszego rozwoju systemu . . . . .	57

# 1. Wprowadzenie

Wraz z dynamicznym rozwojem technologii informatycznych, obserwujemy coraz większe zainteresowanie samodzielnym tworzeniem oraz utrzymywaniem środowisk serwerowych poza scentralizowaną infrastrukturą chmurową. Zjawisko to przybiera szczególne znaczenie wśród pasjonatów IT, administratorów systemów, a także inżynierów oprogramowania, którzy decydują się na budowę tzw. *HomeLabów* - domowych laboratoriów IT.

HomeLab to prywatne środowisko serwerowe, skonfigurowane najczęściej w warunkach domowych, które umożliwia użytkownikom testowanie, uruchamianie i rozwijanie różnorodnych usług oraz technologii. Może ono przybierać formę jednego serwera z kilkoma kontenerami, klastra urządzeń Raspberry Pi, a nawet rozbudowanego racka z profesjonalnymi serwerami. Niezależnie od skali, HomeLab spełnia istotną rolę edukacyjną, testową, a także produkcyjną w kontekście usług dostępnych lokalnie lub zdalnie poprzez sieć prywatną lub publiczną.

Jednym z głównych powodów, dla których użytkownicy decydują się na stworzenie HomeLaba, są potrzeby edukacyjne i chęć zdobycia doświadczenia z technologiami wykorzystywanymi w środowiskach korporacyjnych. HomeLab stanowi bezpieczne środowisko, w którym można bez ryzyka testować nowe narzędzia, technologie oraz scenariusze awaryjne. Dla wielu użytkowników jest to również sposób na centralizację usług domowych - takich jak serwery multimedialne, automatyczne kopie zapasowe, monitoring, systemy automatyki domowej czy własne rozwiązania chmurowe (tzw. *self-hosting*).

Współczesne HomeLaby korzystają z szerokiej gamy technologii - od wirtualizacji (np. Proxmox, VMware, Hyper-V), przez konteneryzację (Docker, Podman), aż po automatyzację z wykorzystaniem narzędzi takich jak Ansible, Terraform czy Packer. Pojawienie się lekkich systemów operacyjnych, niskonapięciowych jednostek obliczeniowych oraz otwartoźródłowych rozwiązań zarządzających umożliwiło rozwój wydajnych i energooszczędnich środowisk domowych.

Mimo wielu zalet, wdrożenie i zarządzanie HomeLabem nie jest zadaniem trywialnym. Konieczność konfiguracji sieci, wirtualnych maszyn, kontenerów, bezpieczeństwa czy uwierzytelnienia użytkowników może być wyzwaniem, szczególnie dla osób stawiających pierwsze kroki w świecie infrastruktury IT. Dodatkowo, manualna administracja systemem bywa czasochłonna i podatna na błędy, co może negatywnie wpływać na stabilność działania usług oraz

doświadczenie użytkownika.

W tym kontekście kluczową rolę odgrywa automatyzacja. Dzięki niej możliwe jest ograniczenie liczby powtarzalnych czynności administracyjnych, przyspieszenie wdrożeń oraz zminimalizowanie ryzyka błędów konfiguracyjnych. Automatyzacja umożliwia również realizację bardziej zaawansowanych scenariuszy, takich jak:

- samonaprawiające się klastry,
- dynamiczne skalowanie zasobów,
- automatyczne aktualizacje i testy regresji,
- ciągła integracja i dostarczanie (CI/CD),
- monitorowanie i alertowanie.

Dzięki automatyzacji, HomeLab może stać się nie tylko narzędziem edukacyjnym, ale także realnym środowiskiem produkcyjnym obsługującym usługi użytkownika w sposób niezawodny, elastyczny i bezpieczny.

## 1.1 Cel pracy

Celem niniejszej pracy magisterskiej jest zaprojektowanie i implementacja nowoczesnego systemu zarządzania środowiskiem HomeLab, który będzie wspierał użytkownika w procesie budowy, konfiguracji i obsługi infrastruktury IT w sposób zautomatyzowany, intuicyjny oraz skalowalny.

Proponowane rozwiązanie ma za zadanie:

- Ułatwić wdrażanie i zarządzanie usługami w kontenerach,
- Zapewnić dostęp do intuicyjnego interfejsu użytkownika umożliwiającego kontrolę nad całym środowiskiem,
- Zminimalizować potrzebę ręcznej ingerencji w konfigurację systemów,
- Wspierać integrację z popularnymi rozwiązaniami open-source (np. Docker, SQLite, TailScale),
- Zwiększyć bezpieczeństwo i kontrolę nad uruchamianymi usługami.

System ma na celu obniżenie progu wejścia dla osób rozpoczynających pracę z HomeLabem oraz dostarczenie bardziej zaawansowanym użytkownikom elastycznej i rozszerzalnej platformy do dalszego rozwoju. Całość zostanie udostępniona jako projekt open-source, co umożliwi społeczności jego dalsze rozwijanie i dostosowywanie.

## 1.2 Zakres pracy

W ramach niniejszej pracy zostaną omówione następujące zagadnienia:

- Analiza dostępnych technologii oraz przegląd istniejących rozwiązań open-source w zakresie zarządzania infrastrukturą domową,
- Projektowanie architektury systemu, obejmującej backend, frontend oraz warstwę automatyzującą,
- Implementacja interfejsu użytkownika umożliwiającego zdalne zarządzanie HomeLabem,
- Budowa API służącego do komunikacji z systemem operacyjnym i usługami backendo-wymi,
- Integracja z kontenerami Docker oraz obsługa uruchamiania, zatrzymywania i monitorowania usług,
- Wdrożenie mechanizmów bezpieczeństwa: autoryzacji, uwierzytelnienia i ochrony dostępu,
- Przeprowadzenie testów funkcjonalnych oraz wydajnościowych na urządzeniu Raspberry Pi 5,
- Udokumentowanie i przygotowanie instalatora umożliwiającego łatwe wdrożenie systemu przez użytkownika końcowego.

Opracowane rozwiązanie nie tylko upraszcza proces zarządzania domową infrastrukturą IT, ale stanowi również punkt wyjścia do dalszej rozbudowy. W kolejnych rozdziałach omówione zostaną szczegółowo zarówno decyzje projektowe, jak i konkretne aspekty implementacyjne oraz propozycje dalszego rozwoju systemu. Projekt publikowany jest w repozytorium GitHub, co umożliwia społeczności swobodne korzystanie oraz modyfikowanie aplikacji zgodnie z własnymi potrzebami.

## **2. Aspekty funkcjonalne i technologiczne środowiska HomeLab**

### **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.

#### **2.1.1 Ewolucja HomeLabów**

Historia HomeLabów sięga czasów, gdy entuzjaści IT zaczynali od prostych zestawów serwerów fizycznych w domach, często z pojedynczymi maszynami do nauki i testów. Początkowo były to głównie serwery oparte na systemach Linux lub Windows Server, służące do eksperymentów z sieciami, usługami plików czy prostymi aplikacjami. Wraz z rozwojem technologii

wirtualizacji, takich jak VMware czy Hyper-V, HomeLaby zaczęły ewoluować, umożliwiając uruchamianie wielu maszyn wirtualnych na jednym fizycznym serwerze, co znacznie zwiększyło możliwości testowe i oszczędność zasobów.

Kolejnym etapem było pojawienie się konteneryzacji, przede wszystkim dzięki Dockerowi, która pozwoliła na lekkie i szybkie uruchamianie aplikacji w izolowanych środowiskach.

Docker został zaprezentowany w 2013 roku jako projekt firmy dotCloud, która później zmieniła nazwę na Docker Inc. Technologia ta wywodzi się z systemu konteneryzacji opierającego się na mechanizmach LXC (Linux Containers), które były dostępne w jądrze Linuxa już wcześniej, ale trudne w użyciu. Docker wprowadził prosty interfejs do tworzenia, uruchamiania i zarządzania kontenerami, co zrewolucjonizowało sposób dystrybucji i uruchamiania aplikacji. W kolejnych latach Docker zyskał ogromną popularność, stając się de facto standardem w konteneryzacji, a jego rozwój obejmował m.in. wprowadzenie Docker Swarm (2015) - własnego systemu orkiestracji, integrację z chmurami publicznymi oraz rozwój narzędzi do zarządzania cyklem życia kontenerów. W 2017 roku Docker przekazał część swojej technologii do fundacji CNCF w postaci silnika kontenerowego containerd, a sam projekt coraz częściej był wykorzystywany w połączeniu z Kubernetesem, który stał się dominującym rozwiązaniem orkiestracyjnym.

Wprowadzenie orkiestratorów kontenerów, takich jak Kubernetes, umożliwiło zarządzanie złożonymi systemami rozproszonymi nawet w domowym środowisku. Obecnie HomeLaby coraz częściej korzystają z podejścia cloud-native, integrując narzędzia infrastruktury jako kod (IaC), takie jak Terraform czy Ansible, oraz praktyki GitOps, gdzie konfiguracje i wdrożenia są zarządzane poprzez repozytoria Git, co zwiększa powtarzalność i automatyzację.

Ta ewolucja spowodowała, że dzisiejsze HomeLaby to nie tylko miejsca do nauki, ale pełnoprawne środowiska produkcyjne, które mogą obsługiwać usługi multimedialne, automatyzację domową, a nawet prywatne chmury danych i aplikacji.

## 2.2 Technologie wykorzystywane w HomeLabach

W niniejszej sekcji przedstawiono wybrane technologie i rozwiązania wykorzystywane w środowiskach typu HomeLab. Należy jednak zaznaczyć, że poniższy przegląd ma charakter poglądowy i służy jedynie przedstawieniu gotowych rozwiązań dostępnych na rynku. Opisane narzędzia i systemy mają na celu pogłębienie wiedzy czytelnika w zakresie możliwości technologicznych, ale w większości nie zostały wykorzystane bezpośrednio w implementacji systemu

zaprojektowanego w ramach niniejszej pracy.

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.

Przykład tworzenia maszyny wirtualnej za pomocą CLI Proxmox:

```
qm create 100 --memory 2048 --net0 virtio,bridge=vmbr0 --cdrom /var  
/lib/vz/template/iso/ubuntu.iso  
qm start 100
```

- **VMware ESXi** - profesjonalne narzędzie do wirtualizacji serwerów.

Przykład automatycznego wdrożenia VM z wykorzystaniem PowerCLI:

```
New-VM -Name "TestVM" -ResourcePool "Resources" -Datastore "  
Datastore1" -MemoryGB 4 -NumCPU 2
```

- **Hyper-V** - narzędzie do wirtualizacji dostarczane przez Microsoft wraz z systemem Windows.

Przykład tworzenia maszyny wirtualnej PowerShell:

```
New-VM -Name "LabVM" -MemoryStartupBytes 2GB -Generation 2 -  
NewVHDPath "C:\VMs\LabVM.vhdx" -NewVHDSizeBytes 40GB
```

- **Docker i Kubernetes** - technologie konteneryzacji, pozwalające na elastyczne zarządzanie aplikacjami i zasobami.

Przykład prostego pliku docker-compose.yaml:

```
version: '3'  
services:  
  web:  
    image: nginx:latest  
    ports:  
      - "80:80"
```

Przykład uruchomienia playbooka Kubernetes:

```
kubectl apply -f deployment.yaml
```

## 2.2.2 Automatyzacja i zarządzanie konfiguracją

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

Przykład użycia Ansible do instalacji Nginx:

```
- hosts: homelab_servers
  become: yes
  tasks:
    - name: Install Nginx
      apt:
        name: nginx
        state: present
```

Przykład Terraform do stworzenia maszyny wirtualnej na Proxmox:

```
resource "proxmox_vm_qemu" "vm1" {
  name = "homelab-vm"
  memory = 2048
  cores = 2
  disk {
    size = "20G"
  }
}
```

## 2.2.3 Monitoring i analiza

- **Prometheus i Grafana** - rozwiązania do monitorowania wydajności i wizualizacji danych.

Przykład konfiguracji Prometheus scrape config:

```
scrape_configs:
  - job_name: 'node_exporter'
    static_configs:
      - targets: ['localhost:9100']
```

- **Zabbix** - platforma do monitorowania infrastruktury IT.

Przykład prostego szablonu Zabbix do monitorowania serwera:

```
Hostname: homelab-server
Items:
- CPU load
- Memory usage
- Disk space
```

#### 2.2.4 Bezpieczeństwo i ochrona danych

Bezpieczeństwo jest kluczowym aspektem każdego HomeLaba, zwłaszcza gdy przechowywane są dane prywatne. Wśród najważniejszych technologii i praktyk znajdują się:

- **VPN** - umożliwia bezpieczne połączenie z domową siecią z dowolnego miejsca. Popularne rozwiązania to OpenVPN, WireGuard.

Przykład podstawowej konfiguracji WireGuard:

```
[Interface]
PrivateKey = <private_key>
Address = 10.0.0.1/24

[Peer]
PublicKey = <peer_public_key>
AllowedIPs = 10.0.0.2/32
```

- **Zapory sieciowe (firewallego)** - kontrolują ruch sieciowy, chroniąc przed nieautoryzowanym dostępem. Można używać iptables, ufw lub dedykowanych urządzeń.

Przykład reguły ufw zezwalającej na ruch SSH:

```
sudo ufw allow ssh
sudo ufw enable
```

- **Strategie backupu** - regularne tworzenie kopii zapasowych danych i konfiguracji, np. za pomocą rsync, BorgBackup, czy dedykowanych narzędzi chmurowych.

Przykład backupu katalogu za pomocą rsync:

```
rsync -av --delete /home/user/data /mnt/backup/
```

- **Szyfrowanie danych** - zarówno na poziomie dysków (LUKS, BitLocker), jak i podczas transmisji (TLS).

Przykład szyfrowania dysku z LUKS:

```
cryptsetup luksFormat /dev/sdX  
cryptsetup open /dev/sdX encrypted_disk
```

## 2.3 Analiza istniejących systemów do zarządzania homelabem

### 2.3.1 Przegląd dostępnych rozwiązań

Na rynku istnieje kilka systemów umożliwiających zarządzanie homelabem. Do najpopularniejszych należą:

- Proxmox VE [11] - rozbudowane, open-source rozwiązanie do zarządzania maszynami wirtualnymi i kontenerami, oferujące integrację z Ceph i wysoką dostępność.
- Unraid [17] - popularne rozwiązanie NAS z obsługą wirtualizacji i kontenerów, cenione za łatwość obsługi ale ograniczone zastosowanie korporacyjne.
- OpenStack [8] - potężna platforma chmurowa, która może być używana do zarządzania homelabem, ale jej skomplikowana konfiguracja sprawia, że nie jest przyjazna dla początkujących użytkowników.
- TrueNAS [16] - rozbudowane oprogramowanie do zarządzania przestrzenią dyskową, które umożliwia tworzenie prywatnych chmur danych.
- Docker [3] + Kubernetes [6] - stosowane w bardziej zaawansowanych wdrożeniach do zarządzania kontenerami, ale wymagające większej wiedzy technicznej.
- CasaOS [2] - nowoczesny, łatwy w obsłudze system do zarządzania usługami domowymi i multimedialnymi, skierowany do użytkowników z mniejszym doświadczeniem technicznym.

### 2.3.2 Zalety i ograniczenia konkurencyjnych systemów

System	Zalety	Wady
Proxmox VE	Darmowa wersja open-source; Wsparcie dla maszyn wirtualnych (KVM) i kontenerów (LXC); Możliwość tworzenia klastrów wysokiej dostępności.	Brak pełnej automatyzacji wdrożeń; Wysoki próg wejścia dla początkujących użytkowników.

Tabela 2.1: Charakterystyka systemu Proxmox VE

Tabela przedstawia główne cechy systemu Proxmox VE, podkreślając jego zalety w środowiskach testowych oraz ograniczenia dla początkujących użytkowników.

System	Zalety	Wady
Unraid	Intuicyjny interfejs użytkownika; Łatwa obsługa pamięci masowej i kontrolerów; Idealny do mediów domowych i prostych rozwiązań NAS.	Model licencyjny oparty na opłacie jednorazowej; Ograniczona integracja z systemami chmurowymi; Mniej elastyczny w środowiskach korporacyjnych.

Tabela 2.2: Charakterystyka systemu Unraid

Tabela przedstawia główne cechy systemu Unraid, zwracając uwagę na jego prostotę obsługi oraz ograniczenia w zastosowaniach korporacyjnych.

System	Zalety	Wady
OpenStack	Zaawansowane funkcje chmurowe; Skalowalność i modularność; Odpowiedni dla dużych środowisk testowych i produkcyjnych.	Bardzo wysoka trudność wdrożenia; Wymaga dużej ilości zasobów sprzętowych; Niepraktyczny dla małych i średnich HomeLabów.

Tabela 2.3: Charakterystyka systemu OpenStack

Tabela przedstawia główne cechy systemu OpenStack, podkreślając jego zaawansowane możliwości oraz wysokie wymagania wdrożeniowe.

<b>System</b>	<b>Zalety</b>	<b>Wady</b>
TrueNAS	Silne wsparcie dla przechowywania danych; Wbudowana replikacja i ochrona RAID; Idealny do przechowywania dużych zbiorów danych i backupów.	Skupione głównie na funkcjach NAS; Brak natywnego wsparcia dla maszyn wirtualnych; Ograniczona funkcjonalność w zakresie wirtualizacji.

Tabela 2.4: Charakterystyka systemu TrueNAS

Tabela przedstawia główne cechy systemu TrueNAS, zwracając uwagę na jego mocne strony w obszarze przechowywania danych oraz ograniczenia w zakresie wirtualizacji.

<b>System</b>	<b>Zalety</b>	<b>Wady</b>
Docker + Kubernetes	Elastyczność w zarządzaniu aplikacjami kontenerowymi; Łatwe skalowanie infrastruktury; Idealne dla deweloperów aplikacji mikroserwisowych.	Wymaga dużej wiedzy technicznej; Brak wsparcia dla maszyn wirtualnych; Nieodpowiednie dla użytkowników początkujących.

Tabela 2.5: Charakterystyka systemu Docker + Kubernetes

Tabela przedstawia główne cechy połączenia Docker i Kubernetes, podkreślając ich elastyczność oraz wymagania dotyczące wiedzy technicznej.

System	Zalety	Wady
CasaOS	Bardzo prosty, intuicyjny interfejs użytkownika; Łatwa instalacja i konfiguracja; Skupienie na usługach multimedialnych i domowych; Idealne dla użytkowników domowych chcących zarządzać multimediami i prostymi usługami.	Ograniczona skalowalność i funkcjonalność w porównaniu do rozwiązań enterprise; Brak zaawansowanych funkcji automatyzacji i integracji; Nie nadaje się do rozbudowanych środowisk IT.

Tabela 2.6: Charakterystyka systemu CasaOS

Tabela przedstawia główne cechy systemu CasaOS, podkreślając jego prostotę obsługi oraz ograniczenia w zastosowaniach zaawansowanych.

### 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.

Propozycja systemu 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 Systemu Homelab

### 3.1 Wymagania funkcjonalne i niefunkcjonalne

#### 3.1.1 Wymagania funkcjonalne

Wymagania funkcjonalne określają konkretne działania, funkcje lub usługi, które system powinien realizować z punktu widzenia użytkownika końcowego. Obejmują one zachowania systemu w odpowiedzi na dane wejściowe oraz interakcje z użytkownikiem, np. możliwość logowania, zarządzania usługami czy monitorowania zasobów.

1. **Zarządzanie kontenerami** - możliwość konfigurowania i zarządzania kontenerami Docker w systemie, z planowaną możliwością rozszerzenia o maszyny wirtualne w przyszłości.
2. **Panel administracyjny** - intuicyjny interfejs użytkownika stworzony przy pomocy szablonu TailAdmin<sup>[1]</sup>, umożliwiający zarządzanie zasobami systemu oraz usługami uruchamianymi w kontenerach.
3. **Baza danych** - przechowywanie informacji o konfiguracji systemu i użytkownikach w lokalnej bazie danych SQLite.
4. **Bezpieczny dostęp zdalny** - integracja z Tailscale umożliwiająca dostęp z dowolonego miejsca na ziemi.
5. **Automatyzacja wdrożeń** - wsparcia dla CI/CD za pomocą GitHub Actions.
6. **Obsługa domeny** - integracja z DuckDNS w celu dynamicznego zarządzania domeną, co umożliwia przypisanie przyjaznej nazwy do systemu HomeLab. Posiadanie własnej domeny ułatwia dostęp do usług z różnych urządzeń i lokalizacji, bez konieczności zapamiętywania adresów IP. Dodatkowo, taka domena może wskazywać zarówno na zasoby dostępne publicznie, jak i usługi działające lokalnie w prywatnej sieci domowej, co zwiększa elastyczność i komfort użytkowania.
7. **Monitorowanie zasobów** - mechanizm zbierania informacji o wykorzystaniu CPU, pamięci RAM oraz przestrzeni dyskowej.

8. **Wsparcie dla rozszerzeń** - możliwość dodawania nowych funkcji poprzez kontenery Dockera.
9. **Łatwe wdrażanie aplikacji** - opcja uruchamiania własnych usług w kontenerach bez konieczności zaawansowanej konfiguracji.
10. **Bezpieczne uwierzytelnianie i automatyzacja** - mechanizm logowania oparty na tokenach JWT oraz zarządzanie rolami użytkowników.

### 3.1.2 Wymagania niefunkcjonalne

Wymagania niefunkcjonalne natomiast odnoszą się do cech jakościowych systemu, takich jak wydajność, bezpieczeństwo, skalowalność czy łatwość utrzymania. Nie definiują one konkretnych funkcji, ale określają standardy, jakie system musi spełniać w różnych warunkach działania.

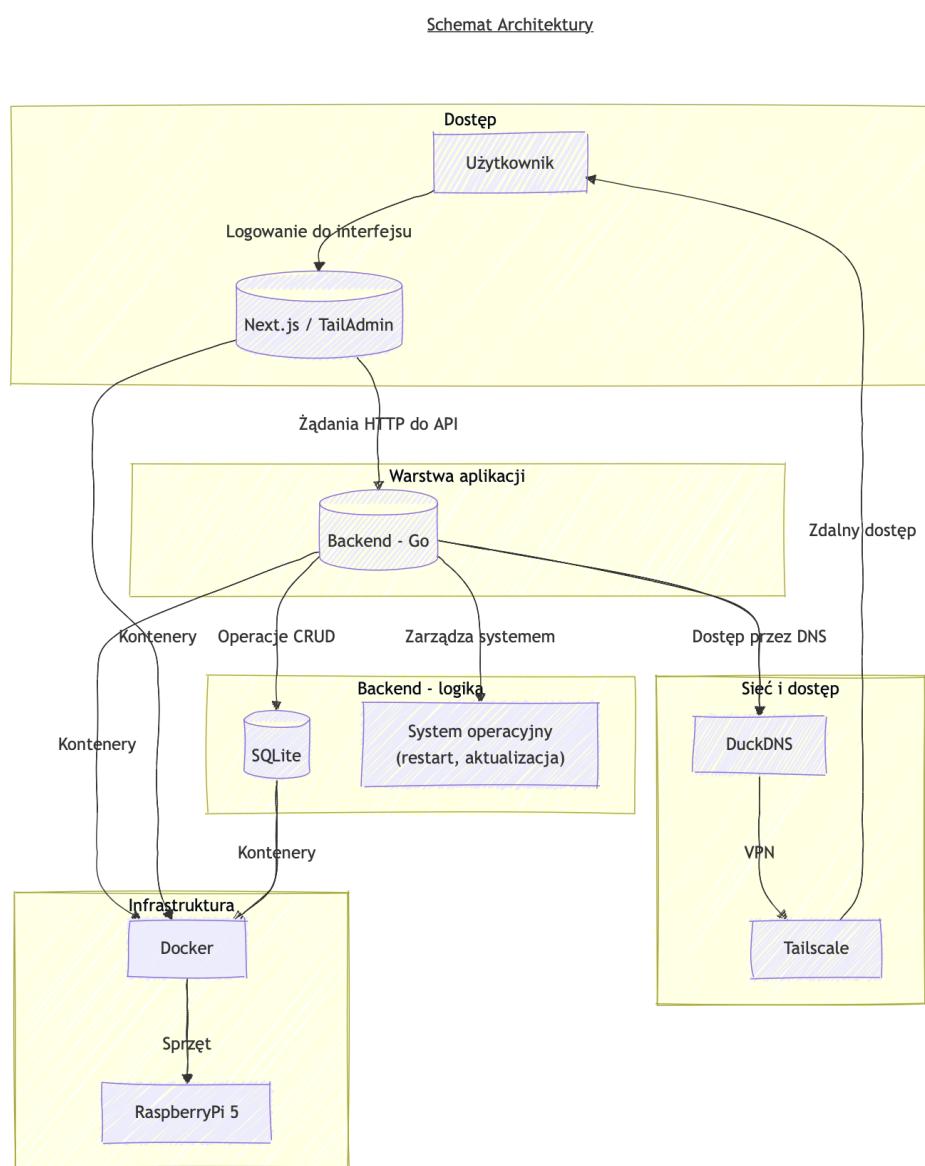
1. **Niski pobór energii** - system wdrażany jest na platformie Raspberry Pi 5, która cechuje się niskim zużyciem energii przy zachowaniu odpowiedniej wydajności. W stanie bezczynności urządzenie zużywa średnio 4-5 W [4], natomiast podczas pełnego obciążenia zapotrzebowanie może wzrosnąć do około 12 W. Zalecane jest zasilanie 5V przy 5A (25 W), co gwarantuje stabilność działania, szczególnie podczas intensywnych operacji takich jak uruchamianie systemu czy dekodowanie wideo. Typowe zużycie prądu w trybie aktywnym wynosi około 0.8 A, a podczas rozruchu lub odtwarzania multimedialnych może chwilowo wzrosnąć do 1.2 A. Dzięki takim parametrom Raspberry Pi 5 stanowi idealną bazę do uruchamiania domowych systemów serwerowych przy zachowaniu niskich kosztów eksploatacji.
2. **Wysoka dostępność** - chociaż obecna wersja systemu nie implementuje jeszcze mechanizmów redundancji, zastosowane technologie takie jak Docker oraz Tailscale VPN otwierają możliwości łatwej rozbudowy architektury o instancje uruchamiane równolegle w wielu lokalizacjach. Dzięki takiemu podejściu możliwe będzie stworzenie systemu odpornego na awarie, zapewniającego ciągłość działania nawet w przypadku niedostępności jednej z lokalizacji. Dodatkowo, rozproszenie instancji umożliwi przyszłe rozłożenie obciążenia pomiędzy węzły oraz łatwe skalowanie systemu w zależności od potrzeb użytkowników.

3. **Łatwość w utrzymaniu** - System powinien cyklicznie sprawdzać dostępność nowej wersji aplikacji poprzez zapytania do repozytorium GitHub. W przypadku wykrycia nowego wydania, użytkownik zostanie poinformowany oraz otrzyma możliwość przeprowadzenia aktualizacji w sposób automatyczny z poziomu interfejsu użytkownika.
4. **Skalowalność** - System wspiera również możliwość dodawania nowych usług poprzez podanie zewnętrznego repozytorium. Użytkownik może w prosty sposób wskazać adres repozytorium zawierającego plik konfiguracyjny Docker Compose, a system automatycznie zainicjuje i uruchomi kontenery na jego podstawie. Obecnie funkcjonalność ta jest ograniczona do podstawowych przypadków, jednak w planach znajduje się rozszerzenie jej o zaawansowane możliwości, takie jak integracja z istniejącymi komponentami systemu, dzielenie wspólnych baz danych pomiędzy różne kontenery oraz optymalizacja użycia zasobów, co pozwoli na minimalizację liczby uruchomionych procesów i zmniejszenie zapotrzebowania na pamięć operacyjną.
5. **Bezpieczeństwo** - szyfrowanie komunikacji oraz kontrola dostępu do zasobów.
6. **Modularność** - podział systemu na niezależne komponenty działające w kontenerach Docker.
7. **Integracja z open-source** - Wsparcie dla narzędzi i technologii dostępnych na licencji open-source.
8. **Minimalizacja kosztów** - niskie koszty sprzętowe i utrzymanie dzięki Raspberry Pi i rozwiązaniom chmurowym typu DuckDNS.
9. **Wydajność** - optymalizacja aplikacji pod Raspberry Pi, aby zapewnić płynne działanie
10. **Łatwość wdrożenia** - uproszczona konfiguracja pozwalająca na szybkie uruchomienie systemu.

## 3.2 Architektura systemu

System HomeLab został zaprojektowany jako aplikacja umożliwiająca zarządzanie kontenerami Docker poprzez backend zrealizowany w języku Go, z frontendem opartym o TailAdmin oraz bazą danych SQLite. Architektura przewiduje możliwość przyszłej rozbudowy o funkcjonalność zarządzania maszynami wirtualnymi.

System HomeLab składa się z kilku kluczowych komponentów:



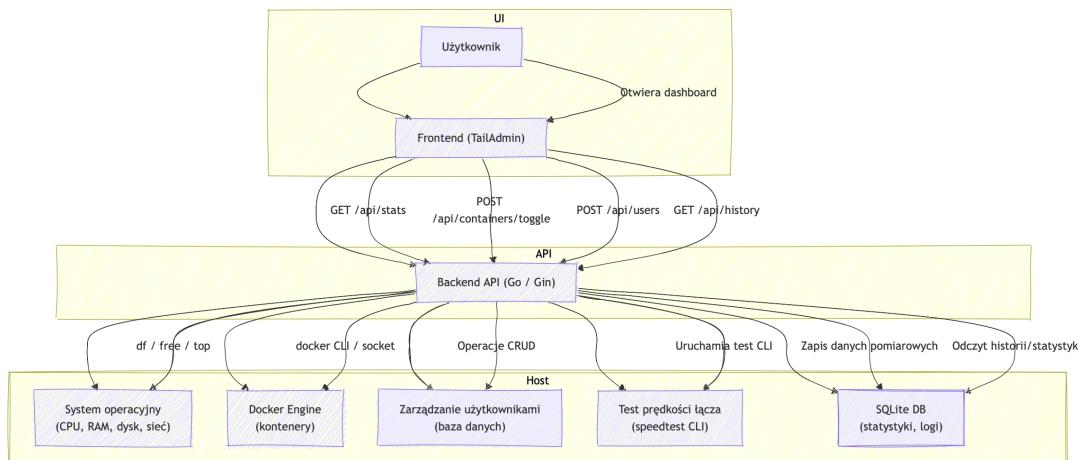
Rysunek 3.1: Schemat architektury działania i komunikacji wewnętrz stworzonego systemu.

Rysunek 3.1 przedstawia ogólną architekturę systemu HomeLab, ukazującą zależności

pomiędzy jego głównymi komponentami. Użytkownik uzyskuje dostęp do systemu poprzez interfejs oparty na Next.js i TailAdmin. Interfejs ten komunikuje się z backendem napisanym w języku Go, który realizuje logikę biznesową oraz zarządza systemem operacyjnym i kontenerami Docker. Dane konfiguracyjne przechowywane są w bazie SQLite. System umożliwia zdalny dostęp dzięki integracji z Tailscale (VPN) oraz DuckDNS (dynamiczny DNS). Całość uruchamiana jest na platformie Raspberry Pi 5, co zapewnia niskie zużycie energii i niski koszt utrzymania.

### 3.2.1 Diagram przepływu danych

W celu lepszego zrozumienia interakcji pomiędzy komponentami systemu, poniżej przedstawiono diagram przepływu danych ilustrujący komunikację pomiędzy frontendem, backendem, bazą danych oraz zewnętrznymi usługami:



Rysunek 3.2: Diagram przepływu danych w systemie HomeLab.

Rysunek 3.2 przedstawia przepływ danych oraz interakcje pomiędzy głównymi komponentami systemu HomeLab. Użytkownik komunikuje się z interfejsem frontendowym (TailAdmin), który z kolei wykonuje żądania HTTP do API opartego na backendzie napisanym w Go z wykorzystaniem frameworka Gin. Backend odpowiada za obsługę logiki biznesowej, zarządzanie kontenerami Docker, wykonywanie operacji na bazie danych SQLite oraz odczyt parametrów systemu operacyjnego. Informacje o stanie systemu są pobierane przy użyciu narzędzi CLI, takich jak df, free, top oraz speedtest, a następnie zapisywane w bazie danych lub zwracane do interfejsu użytkownika.

### 3.2.2 Backend (Go + SQLite)

- Backend został w całości zaimplementowany w języku Go. Odpowiada za całą logikę biznesową aplikacji oraz komunikację z systemem operacyjnym i środowiskiem kontenerowym Docker.
- Aplikacja backendowa zarządza stanem usług, umożliwia ich uruchamianie, zatrzymywanie oraz monitorowanie.
- Dane systemowe oraz informacje o użytkownikach przechowywane są lokalnie w lekkiej bazie danych SQLite, co eliminuje konieczność stosowania zewnętrznego serwera baz danych i upraszcza wdrożenie.
- Backend może być uruchamiany zarówno jako kontener Docker, jak i jako lokalna aplikacja zainstalowana bezpośrednio na systemie operacyjnym.

#### Przykładowy fragment kodu CLI w Go odpowiedzialny za start kontenera:

```
package main

import (
    "fmt"
    "os/exec"
)

func startContainer(containerName string) error {
    cmd := exec.Command("docker", "start", containerName)
    output, err := cmd.CombinedOutput()
    if err != nil {
        return fmt.Errorf("failed to start container: %s, %v", output, err)
    }
    fmt.Printf("Container %s started successfully\n", containerName)
    return nil
}

func main() {
    if err := startContainer("homelab_service"); err != nil {
        fmt.Println(err)
    }
}
```

### 3.2.3 Frontend (TailAdmin)

- Interfejs użytkownika oparty został o szablon TailAdmin, wykorzystujący technologię Next.js i Tailwind CSS.
- Frontend zapewnia nowoczesny, responsywny i intuicyjny interfejs do zarządzania usługami systemu, monitorowania zasobów i konfiguracji ustawień.
- Aplikacja frontendowa komunikuje się z backendem za pomocą REST API.
- Podobnie jak backend, może być uruchamiana jako kontener lub jako statyczna aplikacja hostowana lokalnie.

**Przykładowy fragment kodu React/JSX używanego do pobrania danych systemowych:**

```
import React, { useEffect, useState } from 'react';

function SystemStatus() {
  const [status, setStatus] = useState(null);

  useEffect(() => {
    fetch('/api/system/status')
      .then(res => res.json())
      .then(data => setStatus(data))
      .catch(err => console.error(err));
  }, []);

  if (!status) return <div>Loading...</div>;
}

return (
  <div>
    <h2>Status Systemu</h2>
    <p>CPU: {status.cpuUsage}%</p>
    <p>RAM: {status.memoryUsage} MB</p>
    <p>Dysk: {status.diskSpace} GB wolne</p>
  </div>
);
}
```

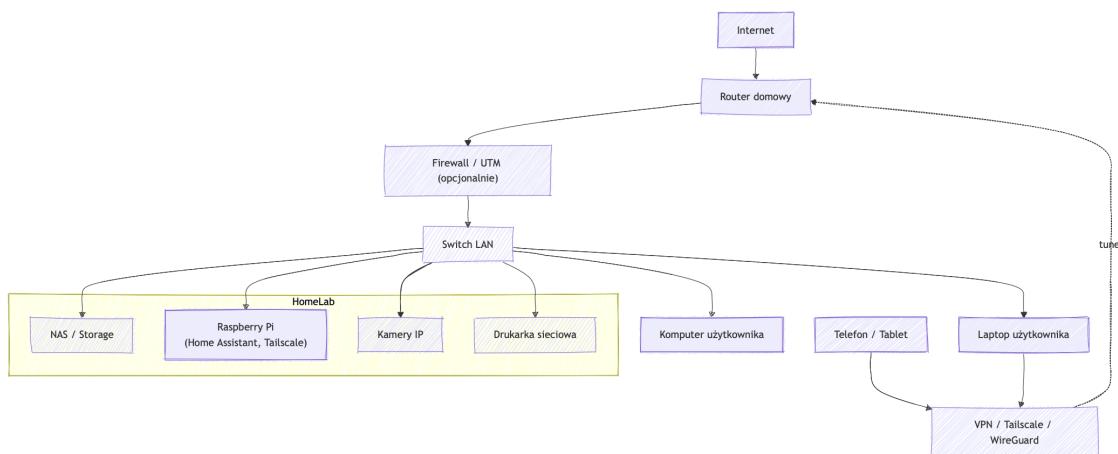
```
export default SystemStatus;
```

### 3.2.4 Warstwa sieciowa

- Połączenia z systemem realizowane są za pomocą sieci VPN opartej o Tailscale, co zapewnia bezpieczny zdalny dostęp bez konieczności stosowania przekierowań portów i stałych adresów IP.
- DuckDNS służy jako system dynamicznego DNS, umożliwiając dostęp do systemu za pomocą przyjaznej domeny.

### Topologia sieci HomeLaba

Aby lepiej zobrazować środowisko, w którym działa system HomeLab, przedstawiono przykładową mapę sieci urządzeń i połączeń w typowej konfiguracji domowej. Schemat ilustruje powiązania między kluczowymi komponentami: serwerem głównym, NAS-em, Raspberry Pi, urządzeniami użytkownika oraz systemami zdalnego dostępu.



Rysunek 3.3: Mapa sieci HomeLaba z uwzględnieniem kluczowych komponentów i połączeń.

### Przykładowa konfiguracja Tailscale z urządzeniem:

```
# Instalacja Tailscale na Raspberry Pi
curl -fsSL https://tailscale.com/install.sh | sh

# Uruchomienie i logowanie do sieci Tailscale
sudo tailscale up --authkey tskey-xxxxxxxxxxxxxx
```

```
# Sprawdzenie statusu połączenia  
tailscale status
```

Konfiguracja umożliwia urządzeniu automatyczne połączenie się z siecią VPN Tailscale, co pozwala na bezpieczny dostęp zdalny do usług HomeLab bez konieczności konfiguracji przekierowań portów.

### 3.2.5 Środowisko kontenerowe

- Docker wykorzystywany jest do zarządzania usługami uruchamianymi w ramach Home-Laba.
- System umożliwia instalowanie, uruchamianie i monitorowanie usług jako niezależnych kontenerów, bez potrzeby ręcznej ingerencji użytkownika.
- Sam system może być uruchamiany jako kontener lub lokalnie, ale niezależnie od tego zarządza innymi kontenerami poprzez lokalne API Dockera.

### 3.2.6 Automatyzacja CI/CD

- GitHub Actions odpowiadają za automatyczne uruchamianie testów jednostkowych oraz analizę statyczną kodu (linter) po każdej zmianie w repozytorium.
- Po pomyślnym przejściu testów generowana jest nowa wersja aplikacji backendowej, frontendowej oraz instalatora.
- Nowa wersja publikowana jest jako release na GitHubie.
- Nie dochodzi do automatycznego wdrożenia na urządzeniu produkcyjnym - użytkownik sam decyduje o pobraniu i uruchomieniu nowej wersji.

**Przykładowy plik ‘.github/workflows/build.yml‘:**

```
name: Build and Test  
  
on:  
  push:  
    branches:  
      - main  
  pull_request:
```

```
jobs:
  build:
    runs-on: ubuntu-latest

  steps:
    - uses: actions/checkout@v3

    - name: Set up Go
      uses: actions/setup-go@v3
      with:
        go-version: 1.20

    - name: Build Backend
      run: go build -v ./backend/...

    - name: Run Backend Tests
      run: go test -v ./backend/...

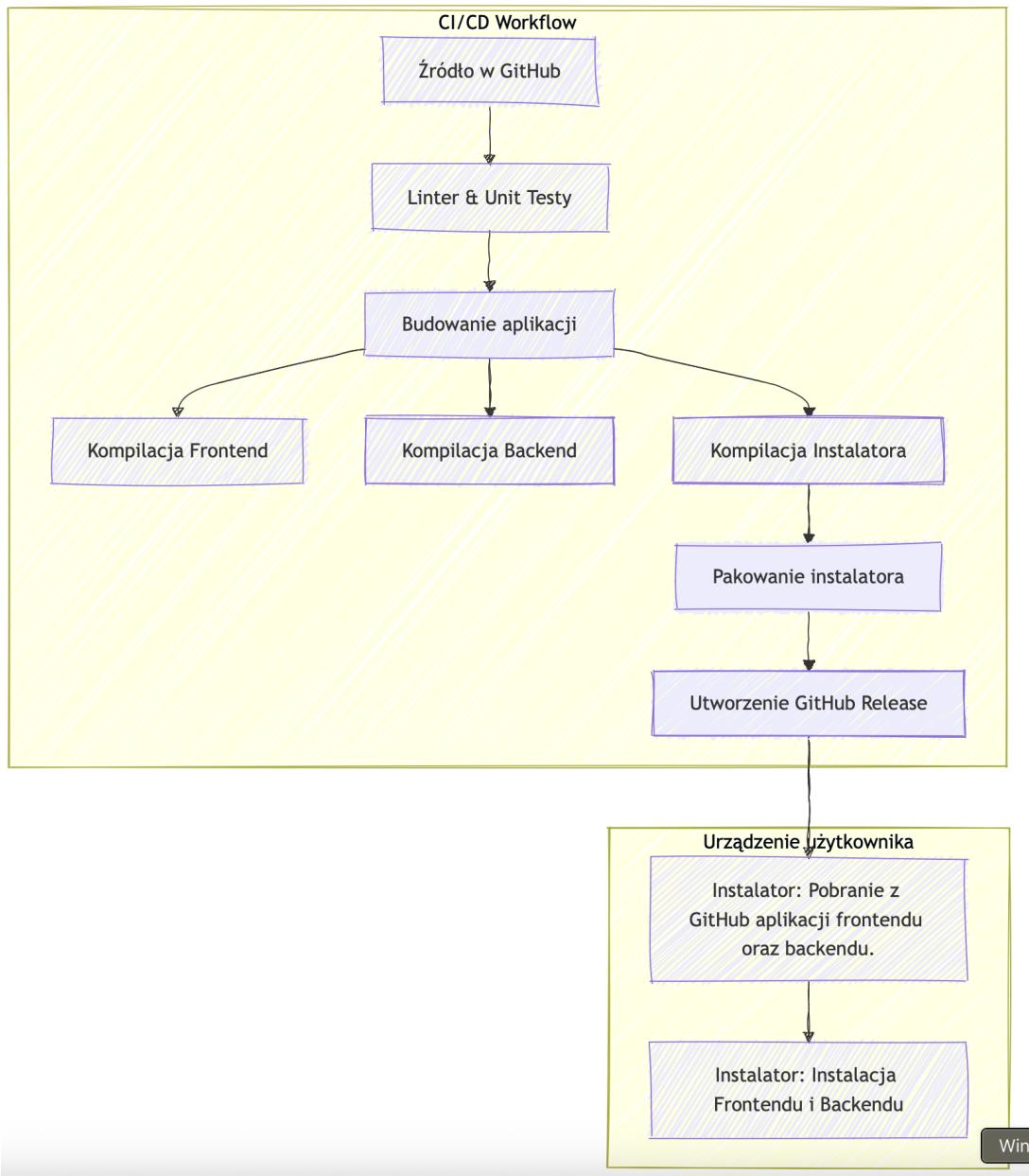
    - name: Build Frontend
      run:
        cd frontend
        npm install
        npm run build

    - name: Run Frontend Tests
      run:
        cd frontend
        npm test

    - name: Create Release
      uses: softprops/action-gh-release@v1
      with:
        tag_name: ${{ github.ref }}
      env:
        GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

### 3.2.7 Wizualizacja procesu CI/CD

Dla pełnego zrozumienia cyklu automatyzacji wdrożenia systemu, przedstawiono uproszczony wykres przepływu operacji CI/CD:



Rysunek 3.4: Schemat procesu CI/CD od momentu zatwierdzenia zmiany aż do utworzenia wydania.

Diagram przedstawia proces automatyzacji CI/CD zastosowany w projekcie HomeLab. Cały proces rozpoczyna się od zatwierdzenia zmian w repozytorium GitHub, co uruchamia mechanizm testów jednostkowych oraz analizę statyczną kodu (linter). Następnie aplikacja jest

budowana - zarówno frontend, backend, jak i instalator. Po zakończeniu komplikacji instalator zostaje spakowany, a całość publikowana w formie GitHub Release. Użytkownik końcowy może następnie pobrać instalator, który automatycznie pobierze najnowsze wersje komponentów z GitHuba i zainstaluje je lokalnie.

### 3.2.8 Urządzenie docelowe

System docelowo uruchamiany jest na urządzeniu Raspberry Pi 5, które zapewnia odpowiednią wydajność przy bardzo niskim zużyciu energii. Dzięki temu możliwe jest ciągłe działanie systemu w warunkach domowych, przy minimalnych kosztach utrzymania.

## 3.3 Technologie i narzędzia użyte w systemie

System HomeLab wykorzystuje następujące technologie:

### 3.3.1 Backend

- Go - zapewnia wysoką wydajność i prostotę wdrożenia na systemach typu embedded jak Raspberry Pi.
- SQLite - lekka, lokalna baza danych eliminująca potrzebę zewnętrznego serwera, idealna do przechowywania konfiguracji i danych użytkowników.

### 3.3.2 Frontend

- TailAdmin - nowoczesny szablon interfejsu użytkownika oparty na Next.js i Tailwind CSS, zapewniający responsywność i łatwość rozbudowy.
- REST API - umożliwia komunikację między frontendem a backendem w sposób prosty i efektywny.

### 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 umożliwiający łatwy dostęp do systemu.

### **3.3.4 Środowisko uruchomieniowe**

- Docker - używany do konteneryzacji aplikacji i zarządzania zależnościami, umożliwiając izolację i łatwe wdrażanie usług.
- Raspberry Pi 5 - host systemu zapewniający energooszczędność i niski koszt.

### **3.3.5 Automatyzacja CI/CD**

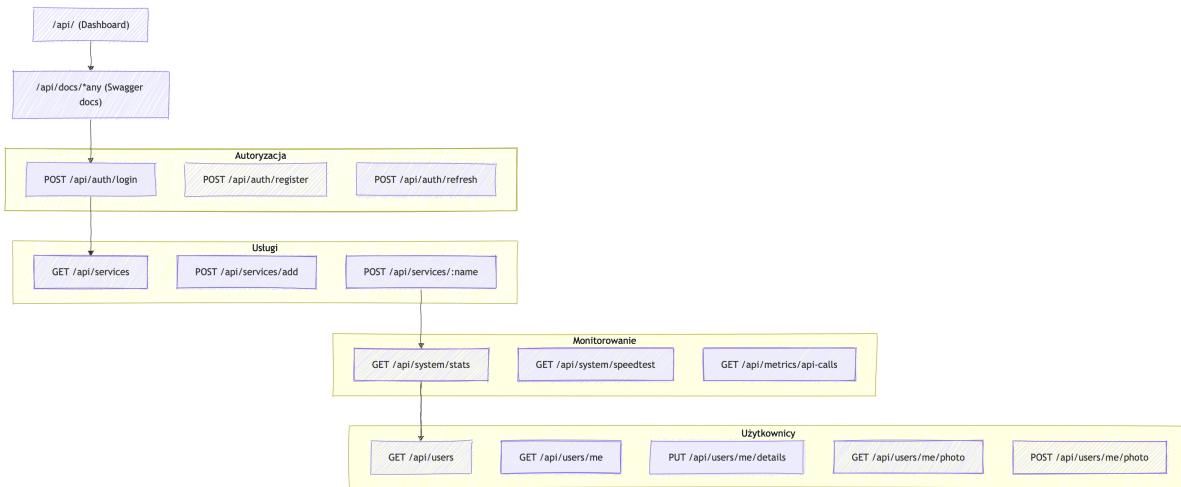
- GitHub Actions - narzędzie do automatyzacji wdrożeń i testowania kodu, umożliwiające szybkie i powtarzalne procesy buildów i testów.
- Pipeline CI/CD - automatyczne testowanie, budowanie i wdrażanie aplikacji, zwiększające jakość i niezawodność systemu.

Dzięki zastosowaniu powyższych technologii system Homelab będzie nowoczesnym, skalowalnym i energooszczędnym rozwiązaniem dla użytkowników domowych.

## 4. Implementacja systemu

### 4.1 Backend - API do zarządzania systemem

#### 4.1.1 Struktura API i kluczowe endpointy



Rysunek 4.1: Schemat architektury API aplikacji HomeNest.

System został zaprojektowany jako aplikacja webowa z backendem opartym na języku Go, wykorzystującym framework Gin, zapewniającym wysoką wydajność oraz możliwość łatwego definiowania tras i obsługi żądań HTTP. API udostępnia szereg endpointów pozwalających na zarządzanie użytkownikami, usługami, monitorowanie systemu oraz uwierzytelnianie.

#### Autoryzacja i uwierzytelnianie

- **POST /api/auth/login** - logowanie użytkownika i zwrócenie tokena JWT,
- **POST /api/auth/register** - rejestracja nowego użytkownika,
- **POST /api/auth/refresh** - odświeżenie tokena JWT.

## Zarządzanie usługami

- **GET /api/services** - pobranie listy aktualnie zarejestrowanych usług,
- **POST /api/services/add** - dodanie nowej usługi do systemu,
- **POST /api/services/:name** - przełączenie (uruchomienie/zatrzymanie) wybranej usługi według nazwy.

## Monitorowanie systemu

- **GET /api/system/stats** - pobranie statystyk systemowych, takich jak zużycie CPU, RAM, przestrzeń dyskowa,
- **GET /api/system/speedtest** - pobranie wyników testu prędkości połączenia internetowego,
- **GET /api/metrics/api-calls** - pobranie liczby zapytań API z ostatnich dni.

## Zarządzanie użytkownikami

- **GET /api/users** - lista użytkowników (dostępna tylko dla administratora),
- **GET /api/users/me** - pobranie danych zalogowanego użytkownika,
- **PUT /api/users/me/details** - aktualizacja danych zalogowanego użytkownika,
- **POST /api/users/me/photo** - przesłanie zdjęcia profilowego,
- **GET /api/users/me/photo** - pobranie zdjęcia profilowego użytkownika.

## Dashboard i dokumentacja

- **GET /api/** - główny endpoint dashboardu aplikacji,
- **GET /api/docs/\*any** - dokumentacja API generowana automatycznie przy użyciu narzędzia Swagger.

## **Podsumowanie**

Zaprojektowane API jest modularne, bezpieczne i łatwe do rozszerzenia. Struktura endpointów została opracowana w sposób zgodny z dobrymi praktykami REST, umożliwiając wygodne zarządzanie zasobami systemowymi i użytkownikami, a dzięki wykorzystaniu frameworka Gin osiągnięto wysoką wydajność obsługi żądań. Endpointy są chronione przez mechanizmy JWT i w zależności od kontekstu użytkownika odpowiednio ograniczają dostęp do operacji administracyjnych.

### **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 użytkownika**

### **4.2.1 Projekt UI/UX**

Z założenia interfejs użytkownika powinien być jak najprostszy oraz jak najbardziej ułatwiać użytkownikowi korzystanie z systemu. Powinien on umożliwiać użytkownikowi korzystanie z systemu w sposób zamierzony oraz uniemożliwiać korzystanie z systemu w sposób niezgodny z przeznaczeniem.

#### **Założenia projektowe**

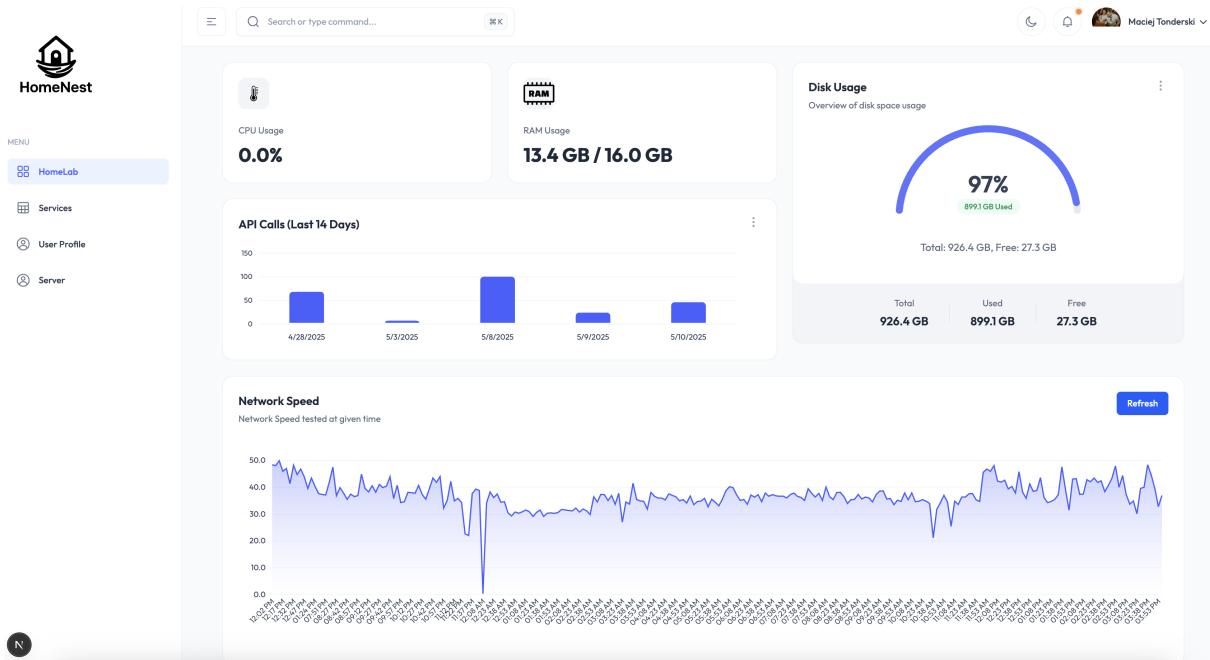
Podstawowe cele projektowe interfejsu obejmowały:

- Czytelność i prostota obsługi,
- Spójność wizualną oraz intuicyjna nawigacja,
- Minimalizacja liczby kliknięć wymaganych do wykonania operacji,
- Odpowiednia organizacja informacji w oparciu o hierarchię wizualną.

#### **Struktura interfejsu**

Projekt składa się z kilku widoków które zostaną opisane poniżej:

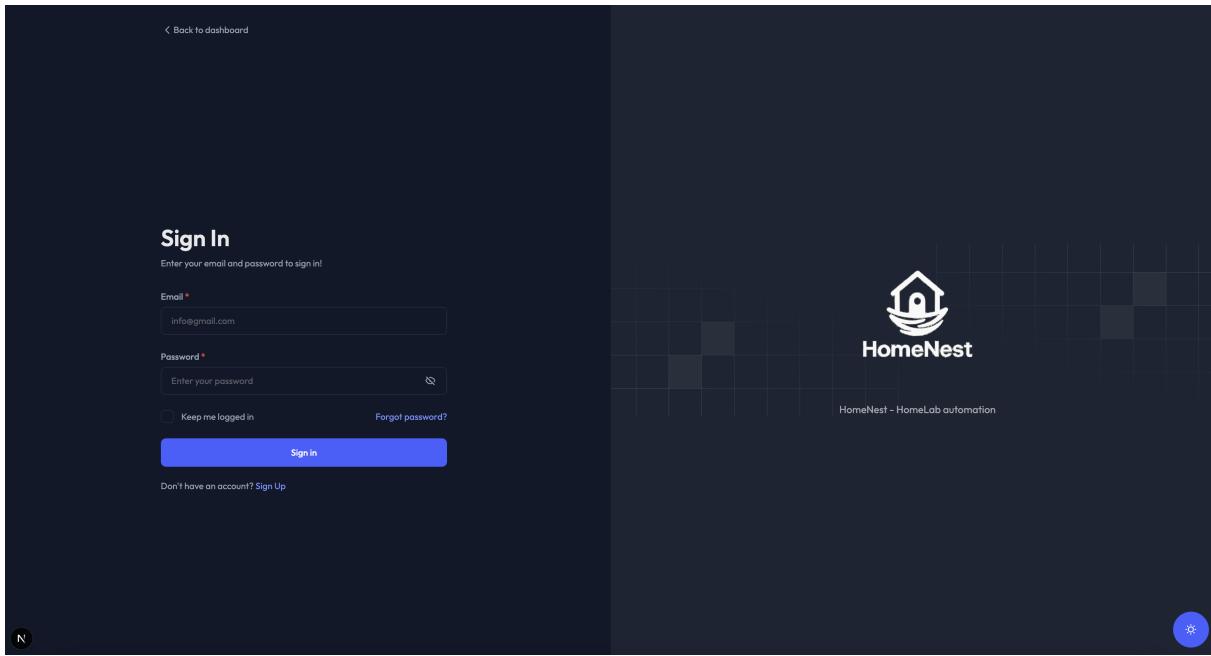
**Strona Główna** - Strona przedstawia statystyki wykorzystania systemu oraz podstawowe informacje dotyczące środowiska uruchomieniowego.



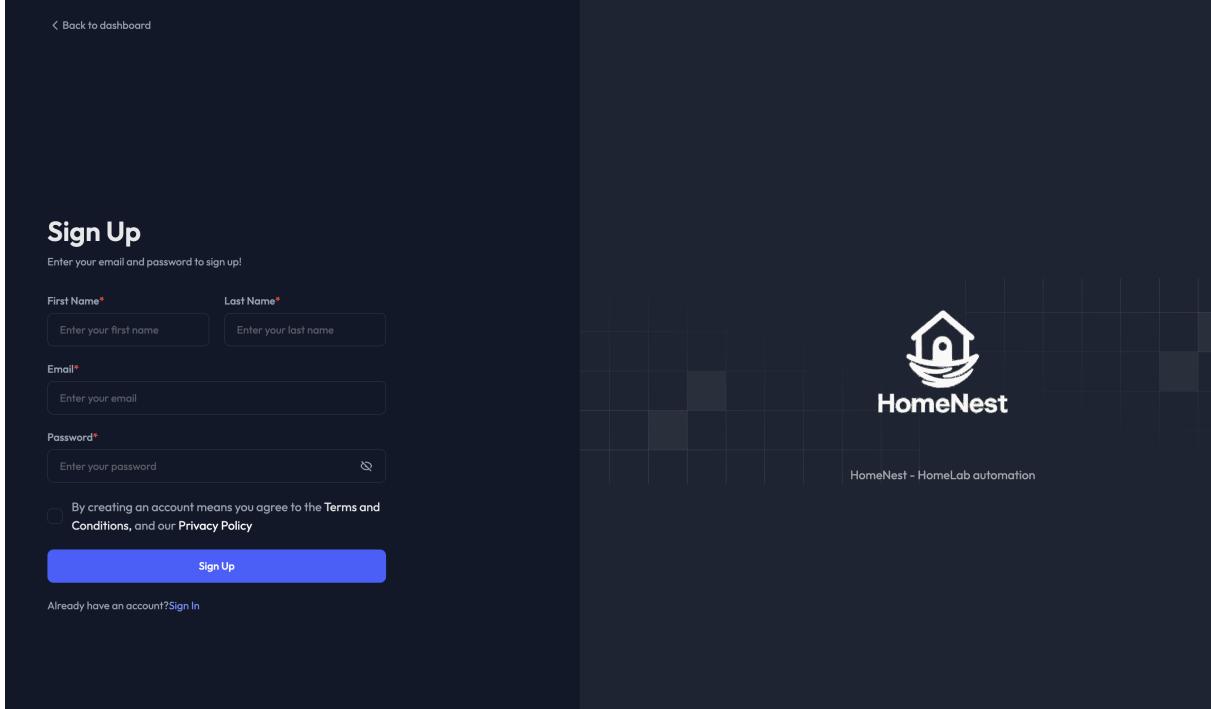
Rysunek 4.2: Strona główna aplikacji Homelab.

- **Panel Informacyjny - wykorzystanie CPU** - Zawiera procentową wartość użycia procesora.
- **Panel Informacyjny - wykorzystanie RAM** - Zawiera informację ile GB pamięci RAM jest obecnie w użyciu oraz dostępną ilość pamięci RAM.
- **Wykres - Wykorzystanie Dysku** - Zawiera informację o wykorzystaniu dysku - ilość zajętej oraz dostępnej przestrzeni dyskowej.
- **Wykres - Ilość zapytań API aplikacji w ostatnich 14 dniach** - Pokazuje ile zapytań do aplikacji API zostało wykonanych w ostatnich 14 danich. Jeżeli aplikacja nie wykonywała zapytań do bazy przez ostatnie 14 dni codziennie umieszczone jest ostatnie 14 wpisów z bazy danych.
- **Wykres - Prędkość internetu** - System zgodnie z harmonogramem sprawdza prędkość podłączonego łącza internetowego i wyświetla je w formie tabeli w tym miejscu.

**Strona Logowania oraz Rejestracji** - Strona umożliwia użytkownikowi wprowadzenie danych niezbędnych do umożliwienia korzystania z systemu. Użytkownik może się zalogować lub zarejestrować.



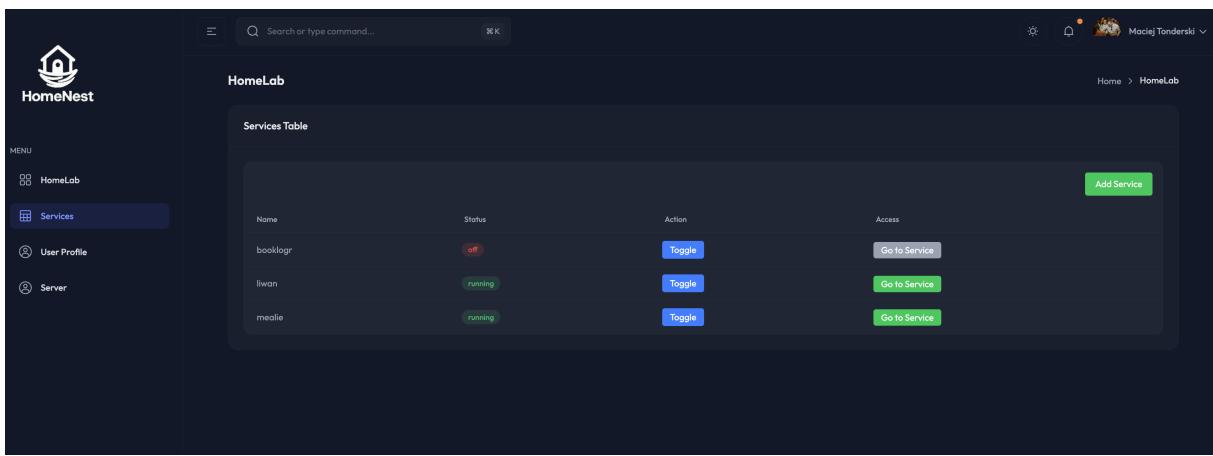
Rysunek 4.3: Strona logowania do serwisu wyświetlająca logo oraz pola do wprowadzenia danych logowania



Rysunek 4.4: Strona Rejestracji do serwisu wyświetlająca logo oraz pola do wprowadzenia danych rejestracji

Użytkownik może się w tym miejscu zalogować lub zarejestrować przy pomocy adresu email oraz hasła. Rejestracja wymaga podania również swojego imienia i nazwiska oraz zaakceptowania polityk korzystania z serwisu, które będą dodane w późniejszym etapie rozwoju aplikacji. Możliwa jest również zmiana kolorystyki poprzez naciśnięcie przycisku w prawym dolnym rogu ekranu, co spowoduje zmianę motywu z jasnego na ciemny lub przeciwnie.

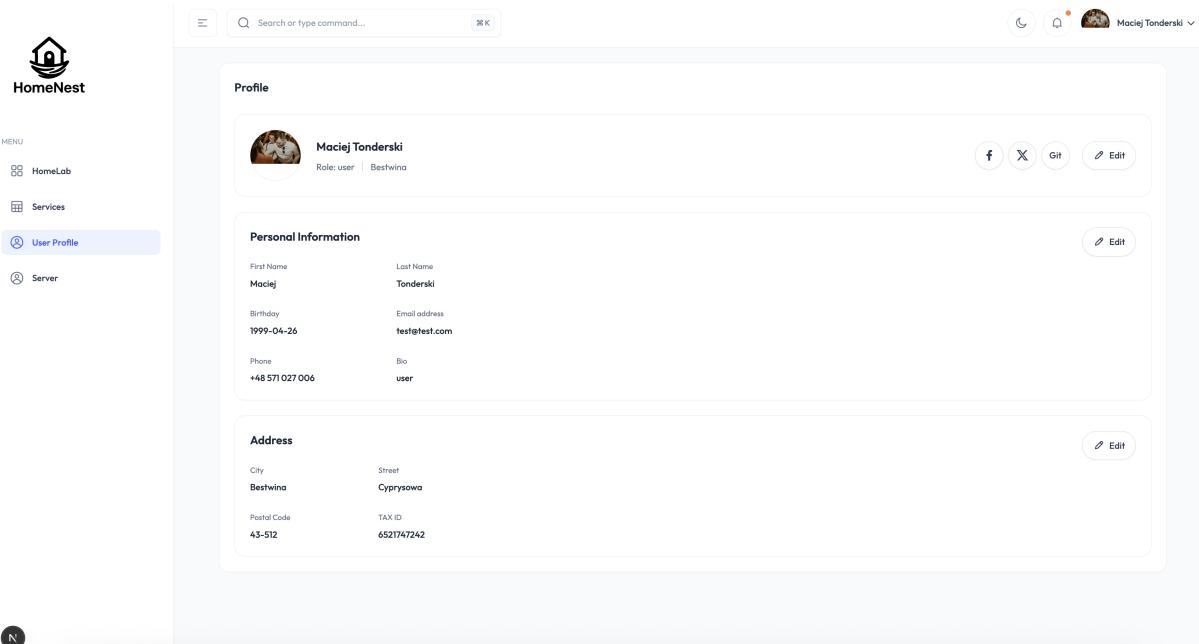
**Strona przedstawienia oraz integracji Serwisów** - Strona pokazuje obecnie dodane serwisy dostępne w systemie. Umożliwia dodanie własnego serwisu, uruchomienie lub zatrzymanie go oraz przejście do strony HTTP serwisu.



Rysunek 4.5: Podstrona aplikacji odpowiedzialna za wyświetlanie stanu obecnie uruchomionych aplikacji, pozwalająca na ich zarządzanie oraz przekierowująca użytkownika do serwisu.

### Strona profilu użytkownika

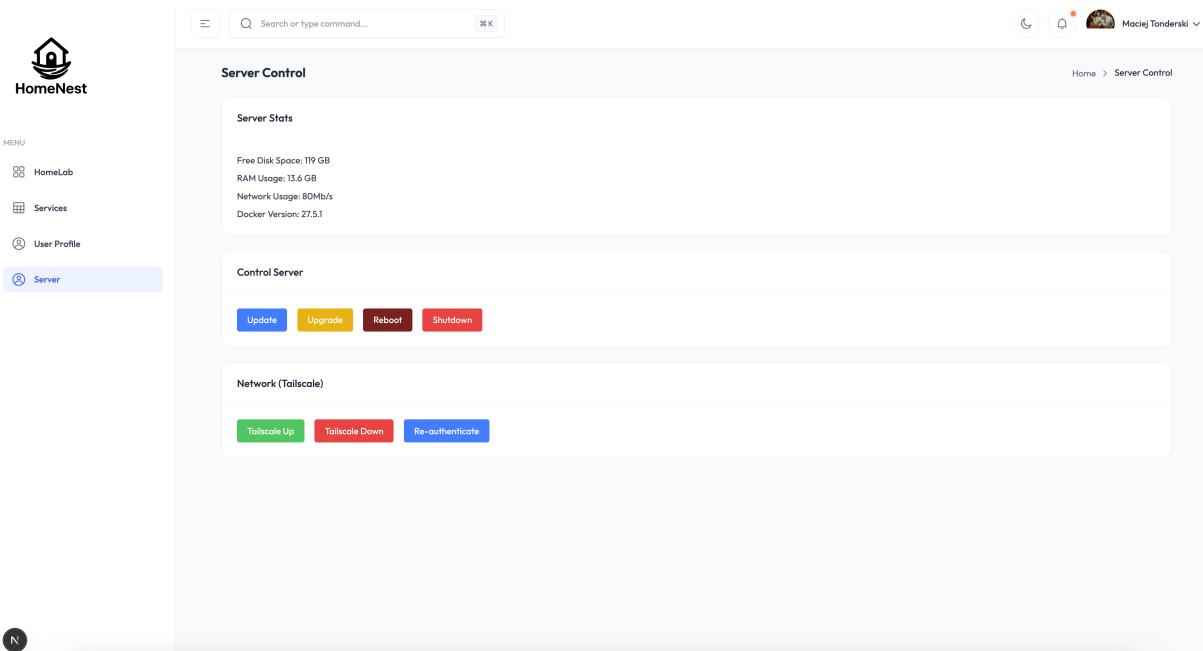
Strona przedstawia dane użytkownika oraz umożliwia ich edycję. Użytkownik może zmienić swoje imię, nazwisko oraz zdjęcie profilowe. Możliwe jest również wylogowanie się z systemu.



Rysunek 4.6: Podstrona aplikacji odpowiedzialna za wyświetlanie stanu obecnie uruchomionych aplikacji, pozwalająca na ich zarządzanie oraz przekierowująca użytkownika do serwisu.

## Strona zarządzania serwerem

Strona została zaprojektowana z myślą o administratorach systemu, zapewniając im intuicyjne komendy zarządzania serwerem oraz monitorowania jego wydajności i stabilności.



Rysunek 4.7: Podstrona wyświetlająca informacje o stanie serwera, jego zasobach oraz umożliwiająca zarządzanie nimi.

Strona przedstawiona na Rys. 4.7 zawiera następujące elementy:

- **Panel Informacyjny - Server Stats** - Zawiera informację na temat wykorzystanych zasobów serwera, takich jak CPU, RAM oraz przestrzeń dyskowa,
- **Panel Informacyjny - Control Server** - Umożliwia użytkownikowi uruchomienie lub zatrzymanie serwera, a także zrestartowanie go. Użytkownik może również zaktualizować system operacyjny oraz zainstalować aktualizacje systemowe,
- **Panel Informacyjny - Network (Tailscale[13])** - Umożliwia użytkownikowi zarządzanie połączeniem z siecią Tailscale, w tym wyłączenie lub włączenie połączenia, przeprowadzenie procesu autoryzacji serwera.

## Kolorystyka i typografia

Projekt interfejsu użytkownika aplikacji HomeNest wykorzystuje nowoczesną i spójną kolorystykę opartą na jasnym tle oraz delikatnych akcentach kolorystycznych. Dominującą barwą tła jest biel (#ffffff), która nadaje przejrzystości oraz zapewnia wysoki kontrast względem elementów graficznych i tekstu.

Główne akcenty kolorystyczne w interfejsie stanowią odcienie niebieskiego (#465fff, #0ba5ec) wykorzystywane m.in. w wykresach, przyciskach akcji oraz wskaźnikach postępu. Barwy te pełnią funkcję informacyjną i nawigacyjną, kierując uwagę użytkownika na istotne elementy interfejsu. Kolorystyka została dodatkowo uzupełniona o neutralne odcienie szarości (#667085, #344054), które służą jako tło dla ikon, ramek i tekstów pomocniczych, umożliwiając ich subtelne wydzielenie bez nadmiernego rozpraszań użytkownika.

Zastosowana typografia oparta jest na nowoczesnym, bezszeryfowym kroju pisma, takim jak Outfit, który zapewnia wysoką czytelność niezależnie od rozdzielczości ekranu. Teksty nagłówków oraz wartości liczbowych zostały wyróżnione większym rozmiarem i wzmacnioną wagą fontu (font-medium, font-bold), co wspiera hierarchizację treści. Z kolei opisy pomocnicze oraz etykiety elementów interfejsu zostały zaprezentowane mniejszym rozmiarem i lżejszą barwą, dzięki czemu nie dominują wizualnie nad informacjami kluczowymi.

Zarówno kolorystyka, jak i typografia wspólnie tworzą minimalistyczny, intuicyjny interfejs, dostosowany do długotrwałego użytkowania oraz zgodny z aktualnymi standardami projektowania systemów zarządzania.

Aplikacja HomeNest wspiera również ciemny motyw interfejsu, co zostało zaprezentowane na rysunku poniżej. W tym trybie dominującym kolorem tła jest głęboka barwa granatowo-grafitowa (#0c111d do #1d2939), która w połączeniu z jasnymi akcentami (np. odcienie niebieskiego i bieli) zapewnia wysoki kontrast oraz elegancki, nowoczesny wygląd interfejsu.

Elementy interaktywne, jak przyciski czy wykresy, zachowują swoją kolorystykę, jednak są odpowiednio dostosowane do ciemnego tła - np. niebieskie słupki w wykresie API Calls, czy wskaźnik użycia dysku są wyraźnie widoczne również w warunkach ograniczonego oświetlenia.

Typografia w trybie ciemnym pozostaje spójna — fonty bezszeryfowe o wysokiej czytelności są renderowane z jasnymi odcieniami (#ffffff, #98a2b3), co sprzyja redukcji zmęczenia wzroku podczas pracy wieczorowej lub w zaciemnionym otoczeniu.

## Korzyści wynikające z obsługi dwóch motywów kolorystycznych

Zaprojektowanie interfejsu w dwóch wersjach — jasnej i ciemnej — jest coraz częściej stosowaną praktyką w nowoczesnych aplikacjach webowych i mobilnych. Taka decyzja projektowa niesie ze sobą szereg korzyści:

- Dostosowanie do preferencji użytkownika Użytkownicy mają różne preferencje dotyczące komfortu pracy z interfejsem. Niektórzy wolą jasne motywy, które przypominają papier i są czytelne w jasnym otoczeniu. Inni preferują ciemne tryby, szczególnie wieczorem, gdy ograniczają zmęczenie wzroku i minimalizują emisję światła niebieskiego.
- Lepsze doświadczenie w różnych warunkach oświetleniowych Tryb jasny jest bardziej funkcjonalny w warunkach dziennego oświetlenia, natomiast tryb ciemny sprawdza się lepiej przy słabym świetle, np. podczas pracy nocą lub w pomieszczeniach bez dostępu do światła dziennego.
- Efektywność energetyczna (w przypadku ekranów OLED) W urządzeniach z ekranami OLED tryb ciemny może realnie wpływać na oszczędność energii, gdyż piksele wyświetlające kolor czarny są całkowicie wyłączone.
- Nowoczesny wygląd aplikacji Obsługa trybu ciemnego jest standardem UX w aplikacjach klasy premium i znaczaco wpływa na odbiór estetyczny systemu, co może mieć pozytywne znaczenie marketingowe i wizerunkowe.

#### **4.2.2 Implementacja interfejsu użytkownika**

Interfejs użytkownika aplikacji HomeNest został zaimplementowany w oparciu o gotowy szablon **TailAdmin[1]**, który dostępny jest publicznie na licencji **MIT**. TailAdmin to nowoczesny szablon frontendowy oparty na technologii **React.js**, zbudowany przy użyciu frameworka **Next.js** oraz systemu stylów **Tailwind CSS**. Dzięki otwartej licencji możliwe było swobodne dostosowanie szablonu do potrzeb aplikacji oraz jego dalszy rozwój.

#### **Proces dostosowania szablonu**

W ramach prac wdrożeniowych wykonano następujące kroki:

- 1. Pobranie i uruchomienie szablonu** - Skonfigurowano środowisko developerskie oraz uruchomiono projekt TailAdmin w trybie deweloperskim.
- 2. Dostosowanie struktury aplikacji** - Zmodyfikowano układ nawigacji, menu, system routingu oraz strukturę stron zgodnie z architekturą systemu HomeNest.
- 3. Implementacja nowych widoków** - Zbudowano dedykowane widoki i komponenty, takie jak: panel główny, zarządzanie usługami, profil użytkownika oraz sekcja administracyjna.

cyjna serwera.

4. **Integracja z API backendu** - Komponenty frontendowe zostały zintegrowane z API stworzonym w technologii Go. Dostosowanie szablonu TailAdmin do współpracy z tym API było zadaniem stosunkowo prostym, lecz wymagającym głębszej analizy w celu zoptymalizowania liczby zapytań wykonywanych do backendu oraz zapewnienia wydajnej komunikacji między warstwami systemu.
5. **Wprowadzenie trybu ciemnego i jasnego** - Rozbudowano system motywów o możliwość przełączania pomiędzy trybem jasnym i ciemnym, z uwzględnieniem pełnej zgodności stylistycznej.
6. **Refaktoryzacja i optymalizacja** - Usunięto zbędne fragmenty kodu, zoptymalizowano komponenty oraz uproszczono logikę interfejsu.

### **Zalety podejścia opartego na gotowym szablonie**

- Znaczaco skrócony czas implementacji frontendowej,
- Wysoka jakość kodu źródłowego i zgodność ze standardami branżowymi,
- Responsywne komponenty i nowoczesny design,
- Możliwość pełnej modyfikacji dzięki licencji MIT,
- Spójność wizualna i łatwość integracji z backendem.

### **Wyzwania podczas adaptacji szablonu**

- Konieczność analizy i zrozumienia istniejącej struktury projektu TailAdmin,
- Ręczne usuwanie niepotrzebnych widoków i kodu demonstracyjnego,
- Potrzeba dostosowania gotowych komponentów do wymagań aplikacji HomeNest,
- Zachowanie spójności wizualnej po wprowadzeniu nowych funkcjonalności.

Podsumowując, wykorzystanie TailAdmin jako szablonu bazowego pozwoliło na szybkie uruchomienie estetycznego i funkcjonalnego interfejsu użytkownika, a dzięki jego modularności i elastyczności możliwe było skuteczne dostosowanie go do specyfiki systemu HomeNest.

## 4.3 Automatyzacja Konfiguracji i wdrożenie

### 4.3.1 Instalacja rozwiązania

W ramach tworzenia rozwiązania powstał również program instalacyjny, który zostanie umieszczony na stronie internetowej rozwiązania umożliwiając pobranie przez zainteresowane korzystaniem z rozwiązania osoby.

#### Program instalacyjny

W celu ułatwienia instalacji aplikacji na systemach użytkowników końcowych przygotowano dedykowany program instalacyjny napisany w języku Go. Aplikacja ta jest odpowiedzialna za:

- Weryfikację obecności Dockera w systemie,
- Automatyczną instalację Dockera w systemach Linux (w pozostałych przypadkach użytkownik zostaje poinformowany o konieczności ręcznej instalacji),
- Pobranie najnowszej wersji aplikacji HomeNest z repozytorium GitHub,
- Rozpakowanie archiwum ZIP zawierającego aplikację do wskazanego katalogu,
- Nadanie odpowiednich uprawnień do uruchomienia aplikacji oraz jej uruchomienie.

Program wykorzystuje standardowe biblioteki Go do obsługi pobierania plików (`net/http`), rozpakowywania archiwów ZIP (`archive/zip`) oraz uruchamiania procesów lokalnych (`os/exec`). Dzięki temu działa w pełni samodzielnie, bez konieczności instalowania zewnętrznych zależności.

Logika działania programu obejmuje:

1. Sprawdzenie czy polecenie `docker` znajduje się w ścieżce systemowej.
2. Jeżeli Docker nie jest zainstalowany — uruchomienie skryptu instalacyjnego (tylko w systemach Linux).
3. Pobranie pliku ZIP z GitHub Releases.

4. Rozpakowanie pliku do katalogu roboczego.

5. Uruchomienie pliku binarnego aplikacji.

Instalator został zaprojektowany z myślą o prostocie użytkowania oraz możliwości łatwego wdrożenia aplikacji przez osoby nietechniczne. Wersje programu można przygotować na platformy Linux, Windows i macOS z wykorzystaniem narzędzia GOOS / GOARCH, umożliwiającego cross-kompilację.

### 4.3.2 Integracja z narzędziami CI/CD

System HomeNest wykorzystuje mechanizmy CI/CD oparte na GitHub Actions[5] w celu automatyzacji kluczowych procesów programistycznych — testowania, budowania, wersjonowania i publikacji aplikacji. Dzięki integracji z CI/CD możliwe jest zapewnienie ciągłości rozwoju i wysokiej jakości kodu bez potrzeby ręcznego wykonywania zadań wdrożeniowych.

#### Schemat działania

Po każdej zmianie wprowadzonej do głównej gałęzi `main`, automatycznie uruchamiane są dwa niezależne pipeline'y GitHub Actions:

1. **Pipeline testujący** — weryfikuje jakość i poprawność kodu,
2. **Pipeline release'owy** — buduje artefakty i tworzy release.

#### Pipeline testujący

Pipeline testujący pełni funkcję strażnika jakości kodu. W jego ramach uruchamiane są:

- **Linter** — analizuje kod źródłowy pod kątem zgodności ze standardami formatowania i stylu,
- **Testy jednostkowe backendu** — realizowane za pomocą `go test`.

Dzięki automatycznemu uruchamianiu testów możliwe jest wykrycie regresji i błędów już na etapie developmentu, zanim zostaną wprowadzone do środowiska produkcyjnego.

## Pipeline release'owy

Po pozytywnym zakończeniu testów uruchamiany jest pipeline odpowiedzialny za zbudowanie artefaktów aplikacji oraz ich publikację jako nowego release'u na GitHubie. Proces ten jest realizowany przy pomocy pliku Makefile, który automatyzuje cały proces budowania i pakowania aplikacji. Makefile definiuje następujące cele:

- **build-frontend** - instalacja zależności npm oraz zbudowanie aplikacji frontendowej; pliki wynikowe umieszczane są w katalogu `backend/static`,
- **build-backend** - komplikacja aplikacji backendowej w Go z ustawieniem zmiennych środowiskowych `GOOS=linux` i `GOARCH=amd64`; wynikowy binarny plik jest zapisywany w katalogu `dist`,
- **package** - kopiowanie plików frontendowych do katalogu dystrybucji oraz utworzenie archiwum `homelab.tar.gz`,
- **clean** - usunięcie katalogów zbudowanych artefaktów i archiwum,
- **all** - domyślna akcja wykonująca czyszczenie, budowę frontendu i backendu oraz pakowanie.

Pipeline GitHub Actions wywołuje polecenie `make all`, co zapewnia spójność budowy na każdym etapie. Dzięki temu proces tworzenia release'u jest w pełni zautomatyzowany, powtarzalny i prosty do uruchomienia lokalnie.

## Przykładowy workflow GitHub Actions

W poniższym przykładzie przedstawiono uproszczony workflow GitHub Actions, który wykorzystuje Makefile do budowania wszystkich komponentów systemu — frontend, backend oraz instalator:

```
name: Release Build

on:
  push:
    branches: [main]

jobs:
```

```
build:
  runs-on: ubuntu-latest

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

    - name: Set up Go
      uses: actions/setup-go@v4
      with:
        go-version: '1.21'

    - name: Set up Node.js
      uses: actions/setup-node@v3
      with:
        node-version: '20'

    - name: Install frontend dependencies
      run: cd frontend && npm ci

    - name: Build all components using Makefile
      run: make all

    - name: Build installer binaries
      run:
        cd instalator
        make prepare
        make all

    - name: Archive release artifacts
      run:
        mkdir release
        cp dist/homelab release/
        cp -r dist/static release/static
        cp instalator/build/* release/
        tar -czf release.tar.gz -C release .

    - name: Create GitHub Release
      uses: softprops/action-gh-release@v1
```

```
with:  
  tag_name: v${{ github.run_number }}  
env:  
  GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}  
files: |  
  release.tar.gz
```

## Strategia wersjonowania

Każdy release oznaczany jest automatycznie na podstawie numeru buildu. W przyszłości możliwe będzie również wdrożenie semantycznego wersjonowania (SemVer) w oparciu o analizę commitów.

## Weryfikacja aktualności aplikacji

W przyszłości system zostanie rozszerzony o mechanizm, który cyklicznie będzie odpytywał GitHub API w celu sprawdzenia dostępności nowego release'u. Porównując aktualnie zainstalowaną wersję z najnowszym tagiem, system będzie w stanie:

- Powiadomić użytkownika o dostępnej aktualizacji,
- Wyświetlić changelog (opis zmian z release notes),
- Zaproponować pobranie i automatyczne wdrożenie nowej wersji.

Taka funkcjonalność przyczyni się do utrzymania systemu w najnowszej, stabilnej wersji bez potrzeby ręcznej ingerencji.

## Korzyści z automatyzacji CI/CD

Integracja GitHub Actions pozwala na:

- Skrócenie czasu wdrożeń,
- Unifikację procesu testowania i publikacji,
- Eliminację błędów ludzkich,
- Wzrost zaufania do jakości aplikacji,

- Łatwą replikację procesu na nowych środowiskach (np. staging, produkcja).

Dzięki powyższym cechom proces CI/CD jest nieodzownym elementem systemu HomeNest i fundamentem jego dalszego skalowania.

# 5. Analiza efektywności i bezpieczeństwa 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 oczekiwaniemi. Do przeprowadzenia testów jednostkowych zastosowano **go test**

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.

Przeprowadzenie testów jednostkowych umożliwiło rozwój serwisu API bez konieczności posiadania gotowego rozwiązania frontend. Dzięki czemu od samego początku możliwa była weryfikacja oraz wychwycenie błędów logicznych w aplikacji. Takie podejście umożliwia szybki rozwój programu bez obawy o działanie pojedynczych metod i funkcji. Testy jednostkowe nie weryfikują całości działania aplikacji a jedynie indywidualnie jej najmniejsze komponenty. Należy mieć na uwadze, że zgodnie ze sztuką testy jednostkowe powinny nie być zależne od innych metod i funkcji a jeśli dana funkcja lub metoda posiada taką zależność należy

ją zasymulować w kodzie poprzez stosowanie tzw. **Mocków**. Jest to nic innego jak symulowanie działania innej funkcji. Umożliwia ono np. w środowisku testowym uzyskiwać odpowiedzi HTTP poprzez zwracanie stałych odpowiedzi bez konieczności wykonywania zapytań HTTP. Pozwala na integrację z plikami bez fizycznej manipulacji plików. Testy jednostkowe to podstawowy element procesu sprawdzania poprawności oprogramowania. Testów tych powinno być najwięcej zgodnie z paradygmem **Shift Left**.

### 5.1.2 Testy integracyjne

Testy integracyjne koncentrują się na weryfikacji współpracy między różnymi komponentami systemu. Ich celem jest zapewnienie, że poszczególne moduły działają poprawnie w połączeniu z innymi oraz że komunikacja między nimi przebiega zgodnie z oczekiwaniemi. Testy integracyjne są kluczowe dla identyfikacji problemów związanych z interakcjami między komponentami, które mogą nie być widoczne podczas testów jednostkowych. W ramach testów integracyjnych przeprowadzono:

- Weryfikację poprawności działania endpointów API w kontekście komunikacji z bazą danych,
- Weryfikację parsowania danych otrzyamanych z zewnętrznych źródeł (np. speedtest) oraz ich poprawnego zapisu do bazy danych,
- Testy funkcjonalne dla kluczowych scenariuszy użytkowania, takich jak rejestracja, logowanie, dodawanie usług,

Testy integracyjne zostały zautomatyzowane przy użyciu narzędzi takich jak Postman oraz Newman, co umożliwiło ich łatwe uruchamianie i monitorowanie wyników. Dzięki temu możliwe było szybkie wykrywanie problemów związanych z integracją różnych komponentów systemu oraz ich eliminacja.

## 5.2 Analiza wydajnościowa systemu

### 5.2.1 Testowanie wydajności systemu

Testowanie wydajności aplikacji jest kluczowe do zweryfikowania zachowania działania programu pod obciążeniem. Przeprowadzenie takiego testu pozwala na znalezienie tzw. "Wąskich garder" programu czyli ścieżek logicznych, które najbardziej obciążają działającą aplikację. Dzięki analizie takich testów możemy odpowiednio nadać wagę zadaniom optymalizacyjnym,

żeby poprawić odbiór aplikacji poprzez końcowego użytkownika. Testy wydajnościowe to również testy długoterminowe, dzięki nim można zrozumieć jak aplikacja zachowuje się z długotrwałym obciążeniem i poznać jak reaguje na nagłe skoki obciążenia w różnych momentach czasu działania programu.

Do przeprowadzenia testów wydajnościowych użyty został darmowy program **ddosify** - jego prosta obsługa oraz konfiguracja, pozwoliła na szybkie jego wdrożenie, a przejrzysty raport wykonania testu na sprawną analizę wydajności programu.

### 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.
- Zweryfikowanie czy w aplikacji nie występują wycieki pamięci podczas długotrwałych testów.

### Metodyka i analiza testów wydajnościowych

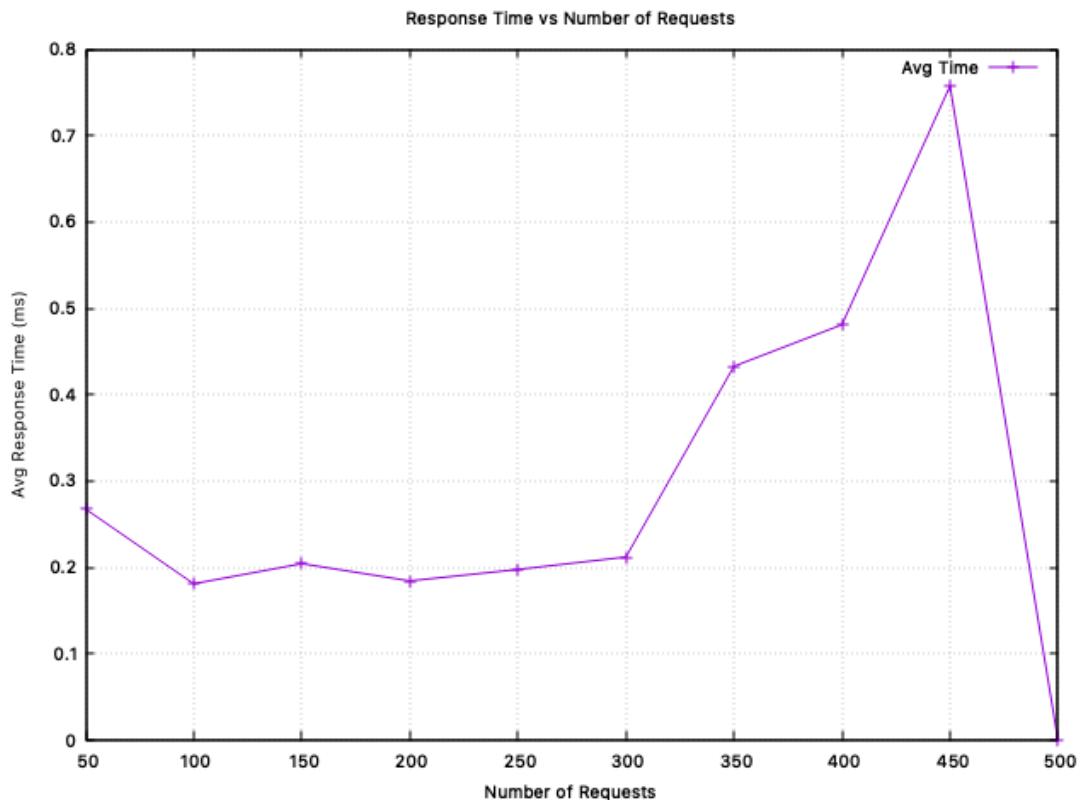
Testy wydajnościowe zostały przeprowadzone w celu oceny stabilności oraz czasu reakcji systemu backendowego pod rosnącym obciążeniem. Do realizacji testów wykorzystano narzędzie **Ddosify**, które umożliwia generowanie ruchu HTTP z określona liczbą żądań oraz mierzenie parametrów takich jak średni czas odpowiedzi, liczba błędów oraz przepustowość systemu. Narzędzie to zostało wybrane ze względu na łatwość integracji oraz możliwość uruchamiania testów z poziomu skryptów bashowych.

Celem testów było:

- Określenie, jak system reaguje na wzrastające obciążenie,
- Pomiar średniego czasu odpowiedzi dla różnych poziomów liczby zapytań,
- Weryfikacja stabilności systemu i pojawiających się błędów przy dużym ruchu,

- Wskazanie potencjalnych miejsc wymagających optymalizacji.

W ramach testu wykonano serię zapytań do jednego z endpointów API, zwiększając liczbę żądań w krokach od 50 do 500. Dla każdego poziomu obciążenia zarejestrowano średni czas odpowiedzi. Dane zostały zapisane i zwizualizowane w postaci wykresu.



Rysunek 5.1: Średni czas odpowiedzi API w zależności od liczby żądań

Na powyższym wykresie zauważalna jest względna stabilność średniego czasu odpowiedzi dla zakresu od 50 do około 300 zapytań. W tym przedziale wartości oscylowały wokół 0.18-0.22 ms. Po przekroczeniu 300 żądań widoczny jest wyraźny wzrost opóźnień, a największy skok nastąpił przy 450 żądaniach, gdzie średni czas odpowiedzi osiągnął ponad 0.75 ms. W ostatniej próbie (500 żądań) wszystkie zapytania zakończyły się błędem (500/500 błędów), co skutkowało średnim czasem odpowiedzi równym 0 ms, gdyż odpowiedzi nie zostały zwrócone w ogóle lub zakończyły się błędami.

Za główną przyczynę wystąpienia błędów uznano ograniczenia wydajnościowe bazy danych SQLite, która nie jest przystosowana do obsługi wielu równoczesnych zapytań w środowiskach wysokiego obciążenia. W szczególności brak wsparcia dla wielu jednoczesnych zapisów prowadził do blokowania operacji i przekroczenia limitów czasu odpowiedzi. Dodatkowym

czynnikiem była ograniczona liczba zasobów przydzielona aplikacji w środowisku testowym.

**Wnioski:**

- Aplikacja działa stabilnie przy niskim i umiarkowanym obciążeniu (do 300 równoczesnych zapytań),
- Wzrost liczby równoczesnych żądań znacząco wpływa na opóźnienia odpowiedzi, co wskazuje na konieczność optymalizacji backendu,
- Wystąpienie 100% błędów przy maksymalnym obciążeniu (500 zapytań) wskazuje na poważne ograniczenie w warstwie komunikacji z bazą danych.

**Rekomendowane działania optymalizacyjne:**

- Zastosowanie mechanizmu *connection poolingu* oraz zwiększenie maksymalnej liczby jednocześnie połączonych do bazy danych,
- Wprowadzenie cache'owania odpowiedzi API dla często wywoływanych zapytań,
- Refaktoryzacja zapytań do bazy danych - redukcja złożoności oraz optymalizacja indeksów,
- Przeprowadzenie testów w środowisku zbliżonym do produkcyjnego z większą wydajnością I/O bazy danych.

Należy zaznaczyć, że testy wydajnościowe w przedstawionej formie przypominają bardziej atak typu DDoS niż rzeczywiste wykorzystanie systemu przez użytkowników. Mimo to są one niezwykle przydatne w procesie optymalizacji aplikacji - pozwalają zidentyfikować komponenty wymagające poprawy wydajnościowej. Nawet jeśli dana optymalizacja może wydawać się zbędna na etapie prototypowania, warto mieć na uwadze, że zoptymalizowany kod zużywa mniej zasobów systemowych i lepiej skaluje się w środowiskach produkcyjnych.

### **5.3 Analiza odporności systemu na zagrożenia bezpieczeństwa**

#### **stwa**

Bezpieczeństwo aplikacji webowych jest jednym z kluczowych aspektów projektowania i wdrażania nowoczesnych systemów. W kontekście niniejszego projektu, który zakłada zarządzanie usługami i użytkownikami, istotne jest zapewnienie odpowiedniego poziomu zabezpieczeń, aby zapobiec nieautoryzowanemu dostępowi, manipulacji danymi oraz wyciekom informacji.

### **5.3.1 Zakres testów bezpieczeństwa**

Testowanie bezpieczeństwa przeprowadzono w ograniczonym zakresie, skupiając się na ręcznej weryfikacji najczęściej występujących zagrożeń w aplikacjach webowych, zgodnie z zestawieniem OWASP Top 10. Weryfikacji poddano następujące obszary:

**1. Uwierzytelnianie i autoryzacja** System wykorzystuje tokeny JWT jako mechanizm autoryzacji. W ramach testów:

- Sprawdzono odporność logowania na ataki brute-force - nie wykryto możliwości obejścia zabezpieczeń, jednak brak limitu prób logowania może stanowić potencjalne ryzyko.
- Zweryfikowano, że po wygaśnięciu tokena nie jest on akceptowany przez serwer.
- Potwierdzono, że użytkownicy nieautoryzowani nie mają dostępu do zasobów innych użytkowników.

**2. Odporność na SQL Injection** Wszystkie zapytania do bazy danych wykonywane są za pomocą ORM (GORM), co zapewnia parametryzację i zmniejsza ryzyko podatności na SQL Injection. Próby wstrzyknięcia kodu SQL w pola formularzy zakończyły się niepowodzeniem - aplikacja poprawnie odrzucała nieprawidłowe dane wejściowe.

### **3. Odporność na XSS i CSRF**

- Testy XSS (reflected i stored) nie przyniosły rezultatów - aplikacja nie wstrzykuje niezweryfikowanych danych wejściowych do HTML.
- Aplikacja wykorzystuje token JWT w nagłówkach, co redukuje ryzyko CSRF, ponieważ brak ciasteczek sesyjnych eliminuje możliwość automatycznego przesyłania uwierzytelnienia.

**4. Bezpieczeństwo API** Przeprowadzono testy zabezpieczeń API:

- Wszystkie endpointy wymagające autoryzacji poprawnie zwracały kod 401 w przypadku braku lub niewłaściwego tokena.
- System nie pozwalał na wykonanie operacji z tokenem innego użytkownika.

- Weryfikowano także poprawność nagłówków CORS oraz odpowiedzi serwera na próby nieautoryzowanego dostępu.

**5. Odporność na przeciążenie (DoS)** W ramach testów wydajnościowych przeprowadzono symulację ataku DoS:

- Wysyłano dużą liczbę zapytań w krótkim czasie - powyżej 400 jednocześnie zapytań powodowało błędy odpowiedzi serwera.
- Błędy te były spowodowane ograniczeniami bazy SQLite, która nie wspiera wielu równoczesnych zapisów.

### 5.3.2 Rekomendacje i dalsze działania

Choć aplikacja wykazała odporność na najczęstsze zagrożenia, zaleca się:

- Dodanie limitu prób logowania i mechanizmu blokowania IP,
- Wdrożenie pełnych testów penetracyjnych (np. z użyciem Burp Suite [10] lub OWASP ZAP[9]),
- Monitorowanie logów bezpieczeństwa i integracja z systemem alertów,
- Przejście na bardziej wydajną bazę danych, wspierającą konkurencyjne zapisy (np. PostgreSQL) w przypadku planowanej produkcyjnej eksploatacji systemu.

**Podsumowanie:** Przeprowadzone testy bezpieczeństwa potwierdzają, że aplikacja w obecnym stanie spełnia podstawowe wymogi bezpieczeństwa, w tym poprawną autoryzację, separację danych użytkowników oraz odporność na podstawowe ataki webowe, takie jak XSS, CSRF czy SQL Injection. Skuteczna implementacja mechanizmu JWT, ochrona endpointów oraz brak dostępu do zasobów bez ważnego tokena świadczą o przemyślanej strukturze kontroli dostępu. Niemniej jednak, dalszy rozwój aplikacji - zwłaszcza w kontekście wdrożenia produkcyjnego - powinien uwzględniać wdrażanie zaawansowanych technik testowania i obrony, takich jak automatyczne testy penetracyjne, analiza statyczna kodu źródłowego, monitorowanie logów pod kątem prób ataków oraz integracja z systemami SIEM (Security Information and Event Management). Rekomendowane jest również regularne wykonywanie audytów bezpieczeństwa oraz testów regresyjnych po każdej większej zmianie kodu lub zależności. Tylko takie podejście pozwoli zapewnić długofalowe bezpieczeństwo systemu oraz ochronę danych użytkowników.

# **6. Podsumowanie i wnioski**

## **6.1 Podsumowanie pracy**

W niniejszej pracy magisterskiej przedstawiono projekt, implementację oraz analizę systemu zarządzania usługami i monitorowania serwera, napisanego w języku Go. System składa się z backendu (API) stworzonego w Go oraz frontendowej części zbudowanej w oparciu o szablon TailAdmin. 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 implementacji API w Go, integracji z interfejsem użytkownika TailAdmin, zarządzaniu użytkownikami, optymalizacji wydajności oraz aspektach związanych z bezpieczeństwem aplikacji webowych. Każdy z tych elementów został szczegółowo opisany i przeanalizowany, co pozwoliło na wyciągnięcie cennych wniosków dotyczących zarówno języka Go, jak i nowoczesnych narzędzi frontendowych i backendowych stosowanych w zarządzaniu infrastrukturą IT.

Instrukcja instalacji oraz uruchomienia aplikacji została zamieszczona w pliku README .md w repozytorium projektu na platformie GitHub[15].

### **6.1.1 Osiągnięcia i rezultaty pracy**

W ramach realizacji pracy magisterskiej osiągnięto szereg istotnych rezultatów, które potwierdzają praktyczną użyteczność oraz skalowalność zaprojektowanego systemu. Najważniejsze z nich to:

- Zaprojektowano oraz zaimplementowano backend systemu przy użyciu języka Go. Dzięki zastosowaniu tej technologii aplikacja cechuje się wysoką wydajnością, prostotą dystrybucji oraz łatwością utrzymania.
- Opracowano nowoczesny, responsywny interfejs użytkownika wykorzystując szablon TailAdmin. Interfejs ten umożliwia intuicyjne zarządzanie usługami systemowymi i użytkownikami, a także prezentuje dane o stanie systemu w przejrzystej formie.
- Stworzono kompletne REST API umożliwiające zarządzanie użytkownikami, usługami

oraz monitorowanie parametrów systemowych. API zostało zaprojektowane w sposób modularny i rozszerzalny.

- Wdrożono mechanizmy uwierzytelniania i autoryzacji użytkowników z wykorzystaniem JWT. System obsługuje różne poziomy uprawnień oraz zabezpiecza dostęp do krytycznych funkcjonalności.
- Zaimplementowano funkcjonalności administracyjne obejmujące dodawanie, edytowanie, usuwanie i przeglądanie użytkowników z poziomu panelu administratora.
- Umożliwiono rejestrowanie i zarządzanie usługami systemowymi - uruchamianie, zatrzymywanie oraz monitorowanie ich stanu w czasie rzeczywistym.
- Zintegrowano backend z systemem operacyjnym w zakresie kontroli nad aktywnymi usługami systemowymi i dostępem do danych o zużyciu zasobów (RAM, CPU, dysk).
- Przeprowadzono testy wydajnościowe z użyciem narzędzia Ddosify, które wykazały odporność aplikacji na duże obciążenia oraz pomogły zidentyfikować krytyczne momenty wymagające optymalizacji.
- Zrealizowano testy bezpieczeństwa, w tym odporności na ataki typu brute-force, dostęp nieautoryzowany, SQL Injection oraz Cross-Site Scripting, potwierdzając bezpieczeństwo podstawowych mechanizmów aplikacji.
- Opracowano system budowania i pakowania aplikacji za pomocą Makefile, umożliwiający łatwą komplikację na wiele platform oraz automatyczne tworzenie paczek instalacyjnych.
- Udokumentowano projekt w postaci pliku README .md na GitHubie, w którym zawarto instrukcję instalacji, konfiguracji i uruchomienia systemu zarówno na lokalnych maszynach, jak i w środowiskach produkcyjnych.

System powstały w wyniku realizacji pracy magisterskiej został zaprojektowany zgodnie z zasadami czystej architektury i modularności, co pozwala na jego dalsze rozwijanie i dostosowywanie do indywidualnych potrzeb użytkowników oraz specyfiki środowiska produkcyjnego. Aktualna wersja spełnia wszystkie założenia funkcjonalne, stanowiąc stabilną podstawę dla rzeczywistego wdrożenia.

## **6.1.2 Wnioski i przyszłe kierunki rozwoju**

Na podstawie przeprowadzonych badań, testów oraz implementacji można sformułować następujące wnioski:

- Połączenie języka Go w warstwie backendu oraz szablonu TailAdmin w warstwie frontendowej pozwala na szybkie tworzenie aplikacji o wysokiej wydajności i przejrzystym interfejsie użytkownika.
- Architektura systemu została zaprojektowana w sposób umożliwiający jego łatwą rozbudowę - zarówno poprzez dodawanie nowych endpointów API, jak i integrację z zewnętrznymi usługami.
- Testy wydajnościowe ujawniły, że system dobrze radzi sobie z typowym obciążeniem, jednak należy zwrócić uwagę na ograniczenia wynikające z zastosowania SQLite, szczególnie przy dużej liczbie jednocześnie działających operacji zapisu.
- Mechanizmy bezpieczeństwa wdrożone w systemie (autoryzacja JWT, kontrola ról, odporność na podstawowe ataki webowe) są wystarczające na etapie prototypowania, jednak wymagają dalszej rozbudowy przed wdrożeniem produkcyjnym.

W przyszłości system może zostać rozszerzony o:

- Wdrożenie kolejki zadań (np. z użyciem Redis lub RabbitMQ) w celu obsługi operacji asynchronicznych oraz przetwarzania zadań wymagających więcej czasu.
- Rozszerzenie API o nowe funkcjonalności, takie jak integracja z narzędziami CI/CD, mechanizmy webhooków, czy eksport danych do zewnętrznych systemów.
- Zastosowanie bazy danych wspierającej jednocześnie zapisy (np. PostgreSQL) dla zwiększenia skalowalności.
- Integrację z narzędziami monitorującymi (np. Prometheus, Grafana) i systemami SIEM w celu zwiększenia widoczności operacyjnej i poziomu bezpieczeństwa.
- Wprowadzenie funkcjonalności cache'owania danych, aby zmniejszyć obciążenie backendu i przyspieszyć odpowiedzi aplikacji.

- Wdrożenie mechanizmów uwierzytelniania wieloskładnikowego (MFA) oraz integrację z zewnętrznymi systemami tożsamości (np. LDAP, OAuth).

Podsumowując, opracowana aplikacja stanowi solidne i nowoczesne rozwiązanie, które może być wykorzystane zarówno w małych środowiskach serwerowych, jak i jako baza pod rozbudowane systemy administracji IT. Projekt może zostać rozwinięty o nowe funkcje i zintegrowany z narzędziami stosowanymi w profesjonalnym zarządzaniu infrastrukturą informatyczną.

## 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 [7], Slack [12], Telegram [14]),
- **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 ролей 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.

# Bibliografia

- [1] Tail Admin. *Tail Admin Docs*. URL: <https://tailadmin.com/docs>. (odwiedzono: 10.05.2025).
- [2] CasaOS. *CasaOS Documentation*. URL: <https://casaos.zimaspace.com/>. (odwiedzono: 11.05.2025).
- [3] Docker. *Docker Documentation*. URL: <https://docs.docker.com>. (odwiedzono: 02.02.2025).
- [4] Home-Assistant Forum. *Zużycie prądu przez urządzenie Raspberry Pi 5*. URL: <https://community.home-assistant.io/t/raspberry-pi-5/619702/26?page=2>. (odwiedzono: 11.05.2025).
- [5] GitHub. *GitHub Actions*. URL: <https://github.com/features/actions>. (odwiedzono: 11.05.2025).
- [6] Kubernetes. *Kubernetes Documentation*. URL: <https://kubernetes.io/docs/>. (odwiedzono: 02.02.2025).
- [7] NTFY. *NTFY Documentation*. URL: <https://ntfy.sh/docs/>. (odwiedzono: 02.02.2025).
- [8] OpenStack. *OpenStack Documentation*. URL: <https://docs.openstack.org>. (odwiedzono: 02.02.2025).
- [9] OWASP. *OWASP ZAP Documentation*. URL: <https://www.zaproxy.org>. (odwiedzono: 10.05.2025).
- [10] PortSwigger. *Burp Suite Documentation*. URL: <https://portswigger.net/burp>. (odwiedzono: 10.05.2025).
- [11] Proxmox. *Proxmox Documentation*. URL: [https://pve.proxmox.com/wiki/Main\\_Page](https://pve.proxmox.com/wiki/Main_Page). (odwiedzono: 02.02.2025).
- [12] Slack. *Slack API Documentation*. URL: <https://api.slack.com/docs>. (odwiedzono: 02.02.2025).
- [13] Tailscale. *Tailscale Install Guide*. URL: <https://tailscale.com/kb/1017/install>. (odwiedzono: 11.05.2025).

- [14] Telegram. *Telegram Bot API Documentation*. URL: <https://core.telegram.org/bots/api>. (odwiedzono: 02.02.2025).
- [15] Maciej Tonderski. *NestOpsV2*. URL: <https://github.com/McTonderski/NestOpsV2>. (odwiedzono: 11.05.2025).
- [16] TrueNAS. *TrueNAS Documentation*. URL: <https://www.truenas.com/docs/>. (odwiedzono: 02.02.2025).
- [17] Unraid. *Unraid Documentation*. URL: <https://docs.unraid.net>. (odwiedzono: 02.02.2025).

## Spis rysunków

3.1 Schemat architektury działania i komunikacji wewnątrz stworzonego systemu . . . . .	19
3.2 Diagram przepływu danych w systemie HomeLab. . . . .	20
3.3 Mapa sieci HomeLaba z uwzględnieniem kluczowych komponentów i połączeń. . . . .	23
3.4 Schemat procesu CI/CD od momentu zatwierdzenia zmiany aż do utworzenia wydania. . . . .	26
 4.1 Schemat architektury API aplikacji HomeNest. . . . .	 29
4.2 Strona główna aplikacji Homelab. . . . .	33
4.3 Strona logowania do serwisu wyświetlająca logo oraz pola do wprowadzenia danych logowania . . . . .	34
4.4 Strona Rejestracji do serwisu wyświetlająca logo oraz pola do wprowadzenia danych rejestracji . . . . .	34
4.5 Podstrona aplikacji odpowiedzialna za wyświetlanie stanu obecnie uruchomionych aplikacji, pozwalająca na ich zarządzanie oraz przekierowująca użytkownika do serwisu. . . . .	35
4.6 Podstrona aplikacji odpowiedzialna za wyświetlanie stanu obecnie uruchomionych aplikacji, pozwalająca na ich zarządzanie oraz przekierowującą użytkownika do serwisu. . . . .	36

4.7	Podstrona wyświetlająca informacje o stanie serwera, jego zasobach oraz umożliwiająca zarządzanie nimi. . . . .	37
5.1	Średni czas odpowiedzi API w zależności od liczby żądań . . . . .	50

# Spis tabel

2.1	Charakterystyka systemu Proxmox VE . . . . .	13
2.2	Charakterystyka systemu Unraid . . . . .	13
2.3	Charakterystyka systemu OpenStack . . . . .	13
2.4	Charakterystyka systemu TrueNAS . . . . .	14
2.5	Charakterystyka systemu Docker + Kubernetes . . . . .	14
2.6	Charakterystyka systemu CasaOS . . . . .	15