WYŻSZA SZKOŁA
INFORMATYKI i ZARZĄDZANIA
z siedzibą w Rzeszowie

KOLEGIUM INFORMATYKI STOSOWANEJ

Kierunek: INFORMATYKA

Technologie IoT – Internetu Rzeczy

Filip Walat
Nr albumu studenta w67204

Aplikacja pozwalająca na szybkie sprawdzenie wersji serwisów na serwerach Linux

Prowadzący: Dr. Zofia Matusiewicz

Praca projektowa technologie programistyczne IoT - Python

Rzeszów 2024

Spis treści

Wstęp **3**

1 Opis założeń projektu **4**
1.1 Cele projektu 4
1.2 Wymagania funkcjonalne i niefunkcjonalne 4
1.3 Wymagania Funkcjonalne 5
1.4 Wymagania Niefunkcjonalne 5
1.5 Oczekiwania jakościowe aplikacji dedykowanej 5

2 Opis struktury projektu **6**
2.1 Komponenty i Organizacja Aplikacji 6
2.2 Opis Techniczny Projektu 6
2.3 Instalacja i uruchomienie po stronie serwera 7
2.4 Po stronie użytkownika 7
2.4.1 Wymagania systemowe 7
2.4.2 Mechanizm zarządzania danymi 8
2.5 Repozytorium i System Kontroli Wersji 8

3 Harmonogram Realizacji Projektu **9**

4 Prezentacja warstwy użytkowej projektu **11**
4.1 Warstwa Użytkowa Projektu 11
4.2 Opis interakcji z użytkownikiem 13

5 Podsumowanie **14**

Spis rysunków **16**

Wstęp

W dzisiejszych czasach zarządzanie wersjami serwisów na serwerach Linux staje się coraz bardziej skomplikowane z powodu rosnącej liczby serwisów i ich zależności. Codzienne wyzwania związane z utrzymaniem aktualności i bezpieczeństwa systemów wymagają innowacyjnych i efektywnych rozwiązań. Nasz projekt, wykorzystujący zaawansowane technologie Python i framework Flask, ma na celu wprowadzenie kompleksowego systemu do sprawdzania wersji serwisów, który zarówno usprawni zarządzanie serwerami, jak i znacząco podniesie komfort pracy administratorów systemów.

Dążymy do rozwiązania problemów związanych z monitorowaniem wersji serwisów przez automatyzację procesów, stosując nowoczesne biblioteki takie jak Paramiko do bezpiecznego połączenia z serwerami oraz interaktywne rozwiązania frontendowe. Nasze podejście ma na celu nie tylko ułatwienie zarządzania wersjami oprogramowania, ale także zapewnienie bezpieczeństwa i stabilności systemów poprzez szybkie i efektywne aktualizowanie informacji o serwisach. W ten sposób przyczyniamy się do poprawy ogólnej jakości zarządzania infrastrukturą IT, łącząc praktyczne umiejętności programistyczne z realnymi wyzwaniami administracji serwerami.

Poprzez ten projekt chcemy zademonstrować, jak nowoczesne technologie mogą być wykorzystane do automatyzacji i optymalizacji procesów zarządzania systemami informatycznymi, co w efekcie przyniesie korzyści zarówno dla administratorów, jak i użytkowników końcowych.

Rozdział 1

Opis założeń projektu

1.1 Cele projektu

Celem naszego projektu jest stworzenie zaawansowanej aplikacji do sprawdzania wersji serwisów działających na serwerach Linux. Dążymy do zaprojektowania i implementacji aplikacji, która umożliwia administratorom szybkie i łatwe sprawdzenie wersji zainstalowanych serwisów. Projekt ten ma kluczowe znaczenie dla naszych studiów informatycznych, koncentrując się na wykorzystaniu praktycznych umiejętności programistycznych do rozwiązania realnych problemów w zarządzaniu serwerami.

- **Jaki jest cel projektu?** Stworzenie aplikacji do zarządzania wersjami serwisów na serwerach Linux, który usprawnia monitorowanie i zarządzanie wersjami oprogramowania.
- **Jaki jest problem, który będzie rozwiązywany oraz proszę wskazać podstawowe źródło problemu?** Problemem jest brak zautomatyzowanego narzędzia do monitorowania wersji serwisów na wielu serwerach, co prowadzi do trudności w utrzymaniu aktualności oprogramowania.
- **Dlaczego ten problem jest ważny oraz jakie są dowody potwierdzające jego istnienie?** Niezaktualizowane oprogramowanie może prowadzić do problemów z bezpieczeństwem i wydajnością. Automatyzacja tego procesu jest niezbędna do utrzymania stabilności i bezpieczeństwa systemów.
- **Co jest niezbędne, aby problem został rozwiązany przez zespół i dlaczego?** Niezbędne jest zastosowanie nowoczesnych technologii i metod programowania, aby stworzyć elastyczny i skalowalny system do monitorowania wersji serwisów.
- **W jaki sposób problem zostanie rozwiązany?** Poprzez zaprojektowanie i implementację aplikacji w Pythonie, wykorzystującej framework Flask oraz bibliotekę Paramiko do zarządzania serwerami Linux.

1.2 Wymagania funkcjonalne i niefunkcjonalne

Definicja:

Wymagania funkcjonalne określają konkretną funkcjonalność lub zachowanie aplikacji, które musi zostać zaimplementowane. Obejmują one specyficzne zadania lub funkcje, które aplikacja powinna być w stanie wykonać, takie jak przetwarzanie danych, wykonanie obliczeń, reakcja na określone wejścia użytkownika, i inne wymagane operacje.

Wymagania niefunkcjonalne dotyczą ogólnych jakości aplikcaji, takich jak wydajność, bezpieczeństwo, skalowalność, niezawodność, łatwość użytkowania, i zgodność ze standardami. Te wymagania nie opisują bezpośrednio działań aplikacji, ale określają atrybuty, które muszą być spełnione, aby aplikacja była użyteczna i efektywna w swoim środowisku pracy.

1.3 Wymagania Funkcjonalne

Wymagania funkcjonalne aplikacji obejmują:

- Sprawdzanie wersji serwisów: Użytkownik za pomocą prostego interfejsu może szybko zweryfikować, jakie wersje serwisów są zainstalowane na serwerze.
- Aktualizacja informacji o serwerze: Użytkownik może zaktualizować informacje o serwerze, takie jak lista monitorowanych serwisów.
- Bezpieczne zakończenie pracy z aplikacją przez użytkownika.

1.4 Wymagania Niefunkcjonalne

Wymagania niefunkcjonalne projektu są równie istotne, zapewniając:

- Skalowalność i wydajność: Aplikacja została zaprojektowana z myślą o obsłudze dużej liczby serwerów i użytkowników, zapewniając płynną pracę nawet przy wysokim obciążeniu, co jest kluczowe dla zapewnienia ciągłości działania.
- Szybki czas odpowiedzi i efektywność działania aplikacji.
- Utrzymywalność i łatwość modyfikacji: Kod źródłowy aplikacji jest zgodny z najlepszymi praktykami programistycznymi, co ułatwia wprowadzanie zmian, aktualizacji oraz szybką diagnozę i naprawę ewentualnych błędów.
- Możliwość przeprowadzania testów jednostkowych i integracyjnych.

1.5 Oczekiwania jakościowe aplikacji dedykowanej

Teraz nadchodzi część, w której definiujemy oczekiwania jakościowe aplikacji dedykowanej zarządzania wersjami serwisów. Te atrybuty opisują sposoby, w jakie oczekujemy, że aplikacja będzie się zachowywała:

- **Użyteczność produktu:** Aplikacja powinna charakteryzować się intuicyjnym i łatwym w użyciu interfejsem, minimalizującym potrzebę szkoleń i umożliwiającym szybki dostęp do wszystkich kluczowych funkcji.
- **Dostępność aplikacji:** Aplikacja powiniena być dostępna 24/7/365, z zapewnieniem ciągłości działania nawet w przypadku awarii czy nieprzewidzianych sytuacji.
- **Wydajność aplikacji:** Oczekuje się, że czas odpowiedzi aplikcaji na kluczowe operacje (np. sprawdzanie wersji serwisów) nie będzie przekraczał 10 sekund, a funkcje offline będą dostępne przez co najmniej 24h.

Rozdział 2

Opis struktury projektu

2.1 Komponenty i Organizacja Aplikacji

Projekt Systemu Sprawdzania Wersji Serwisów na Serwerach Linux składa się z kilku głównych komponentów:

- **Backend**: Flask, Python
- **Frontend**: HTML, CSS, JavaScript
- **Skomunikowanie z serwerami Linux**: Paramiko do obsługi SSH
- **Repozytorium kodu i kontrola wersji**: Git, GitHub

Główne skrypty i pliki:

- `app.py`: Główny plik aplikacji Flask, zarządza routingiem i logiką backendu.
- `handler/watcher.py`: Skrypt do monitorowania zmian w folderze hostów.
- `templates/`: Katalog zawierający szablony HTML dla Flask.
- `frontend/static/`: Katalog zawierający pliki statyczne takie jak CSS i JavaScript.
- `handler/results/`: Katalog przechowujący wyniki wersji serwisów dla poszczególnych hostów.
- `handler/fetch/hosts/`: Katalog przechowujący pliki JSON z danymi hostów.

Struktura projektu jest zaprojektowana w taki sposób, aby maksymalizować ponowne wykorzystanie kodu i ułatwić rozszerzanie systemu o nowe funkcjonalności.

2.2 Opis Techniczny Projektu

Projekt został zrealizowany w języku Python z wykorzystaniem frameworka Flask do obsługi backendu oraz HTML, CSS i JavaScript do frontendowej części aplikacji. Do zarządzania projektem i kodem źródłowym wykorzystano środowisko Visual Studio Code oraz system kontroli wersji Git.

2.3 Instalacja i uruchomienie po stronie serwera

Aby uruchomić aplikację na serwerze, należy wykonać następujące kroki:

1. Sklonuj repozytorium z GitHub:

```
git clone https://github.com/filwalu/VersionChecker.git
```

2. Przejdź na gałąź `main`:

```
git checkout main
```

3. Przejdź do katalogu `app`:

```
cd app/
```

4. Utwórz wirtualne środowisko:

```
python3 -m venv .venv
```

5. Zainstaluj wymagane pakiety:

```
pip3 install -r requirements.txt
```

6. Uruchom aplikację:

```
python3 app.py
```

2.4 Po stronie użytkownika

Aby uzyskać dostęp do aplikacji, użytkownik powinien w przeglądarce internetowej wejść na adres:

`127.0.0.1:5001`

W ten sposób użytkownik połączy się z uruchomioną aplikacją.

2.4.1 Wymagania systemowe

Aplikacja jest zaprojektowana z myślą o niskich wymaganiach sprzętowych:

- Procesor: 1 GHz lub szybszy.
- Pamięć RAM: 512 MB.
- Przestrzeń na dysku: 100 MB.
- System operacyjny: Linux, Windows, MacOS.

2.4.2 Mechanizm zarządzania danymi

Projekt wykorzystuje pliki JSON do przechowywania danych o serwisach i wersjach, co pozwala na prostą i efektywną manipulację danymi bez potrzeby korzystania z zewnętrznych systemów DBMS. Struktura plików JSON jest zaprojektowana w taki sposób, aby umożliwić szybkie odczytywanie i zapisywanie stanu serwisów oraz informacji o wersjach, co zapewnia wysoką wydajność działania systemu.

2.5 Repozytorium i System Kontroli Wersji

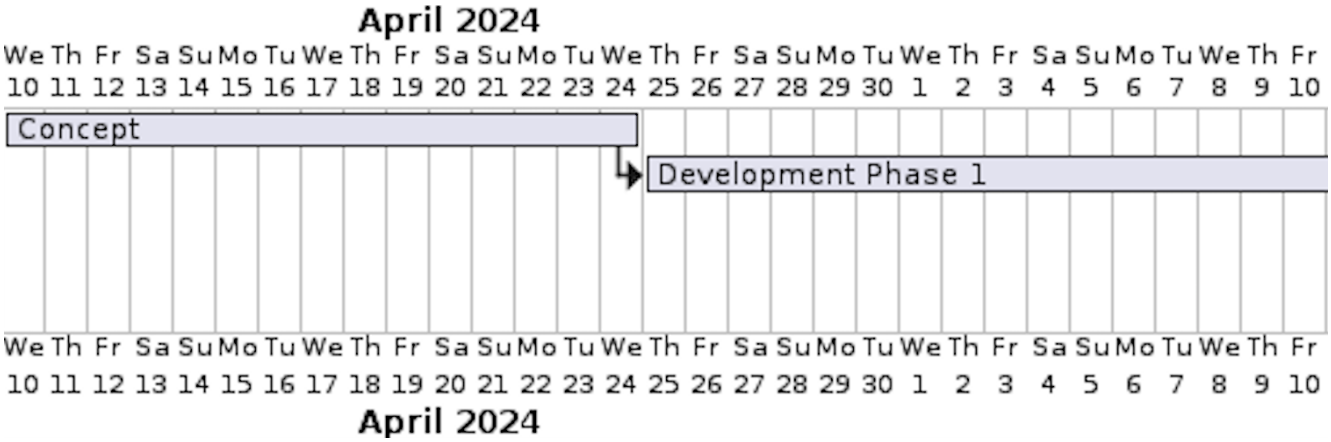
Projekt wykorzystuje system kontroli wersji Git, co umożliwia skuteczne zarządzanie historią zmian kodu źródłowego. Repozytorium kodu znajduje się na platformie GitHub pod adresem: `https://github.com/filwalu/VersionChecker` i będzie dostępne publicznie do dnia 30.09.2024. Bezpieczne połączenie z repozytorium zabezpieczono za pomocą pary kluczy SSH. Poniżej przedstawiono opis użytych poleceń Git:

1. `git init` - inicjalizacja nowego repozytorium Git.
2. `git clone [URL]` - klonowanie repozytorium przy użyciu SSH.
3. `git add -A` - dodawanie zmian do kolejki commitów.
4. `git status` - sprawdzanie statusu zmian.
5. `git commit -m "[wiadomość]"` - commitowanie zmian z opisem.
6. `git push` - wysyłanie zmian do zdalnego repozytorium przez SSH.
7. `git merge [branch]` - scalanie zmian z wybranej gałęzi do bieżącej gałęzi.

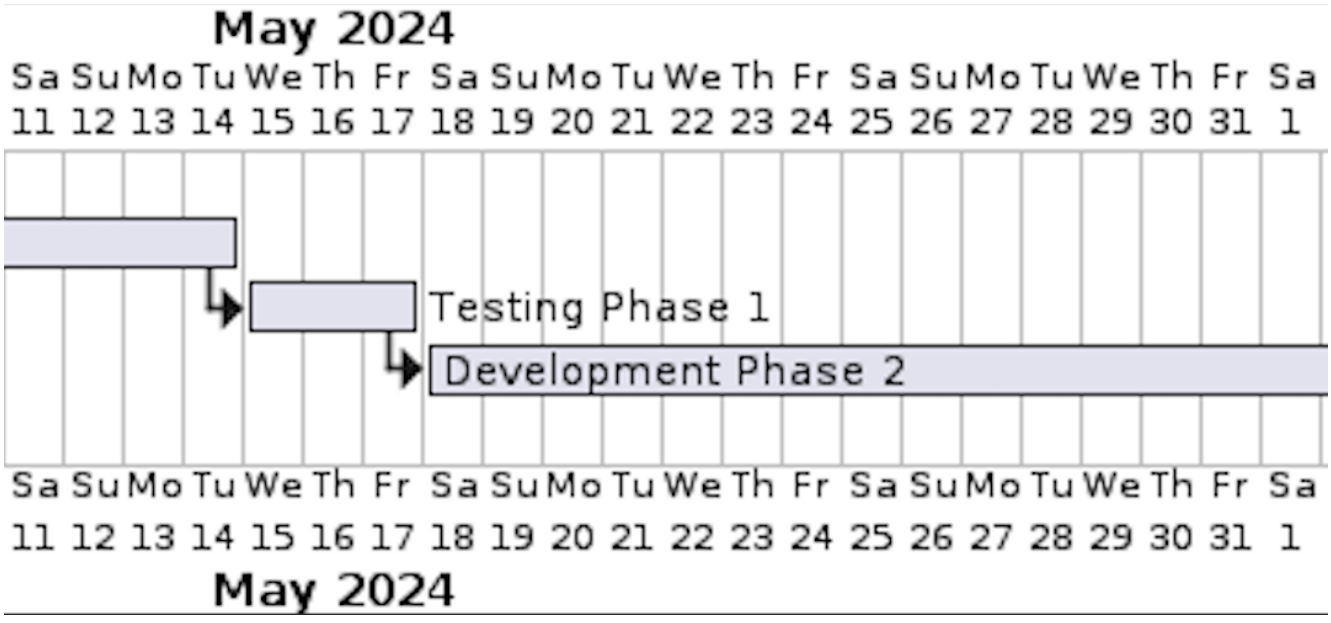
Rozdział 3

Harmonogram Realizacji Projektu

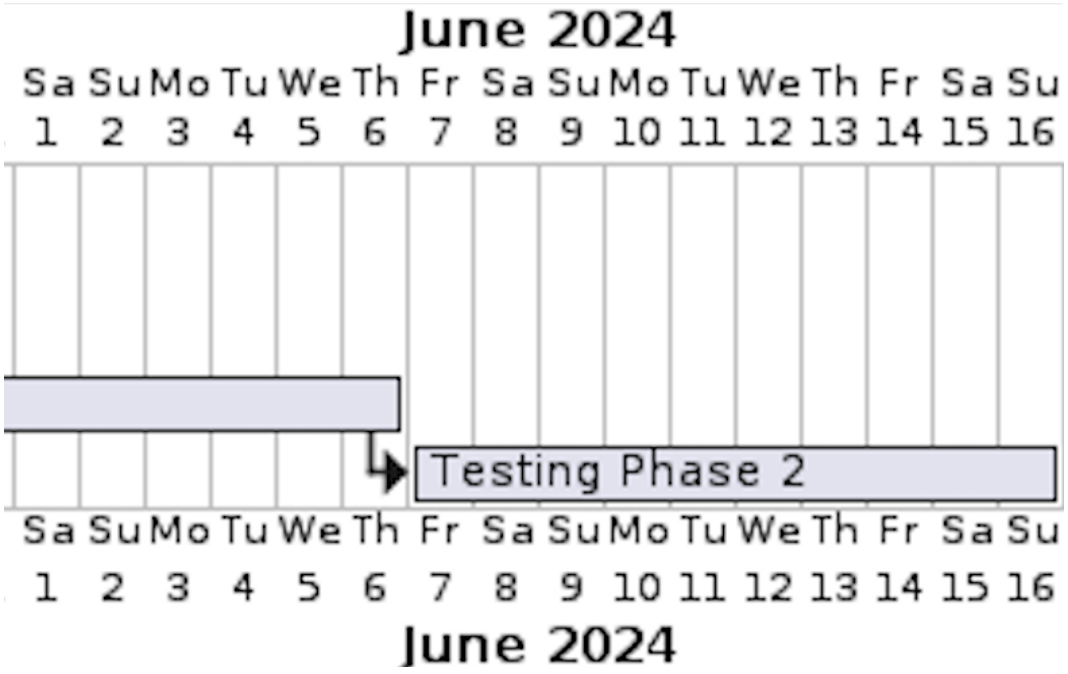
Harmonogram realizacji projektu został zaplanowany z wykorzystaniem diagramu Gantta, który ilustruje kluczowe etapy rozwoju projektu, ich zależności czasowe oraz alokację zasobów. Poniżej przedstawiono diagram Gantta dla projektu aplikacji pozwalającej na szybkie sprawdzenie wersji serwisów na serwerach Linux.



Rysunek 3.1: Diagram Gantta



Rysunek 3.2: Diagram Gantta



Rysunek 3.3: Diagram Gantta

Rozdział 4

Prezentacja warstwy użytkowej projektu

4.1 Warstwa Użytkowa Projektu

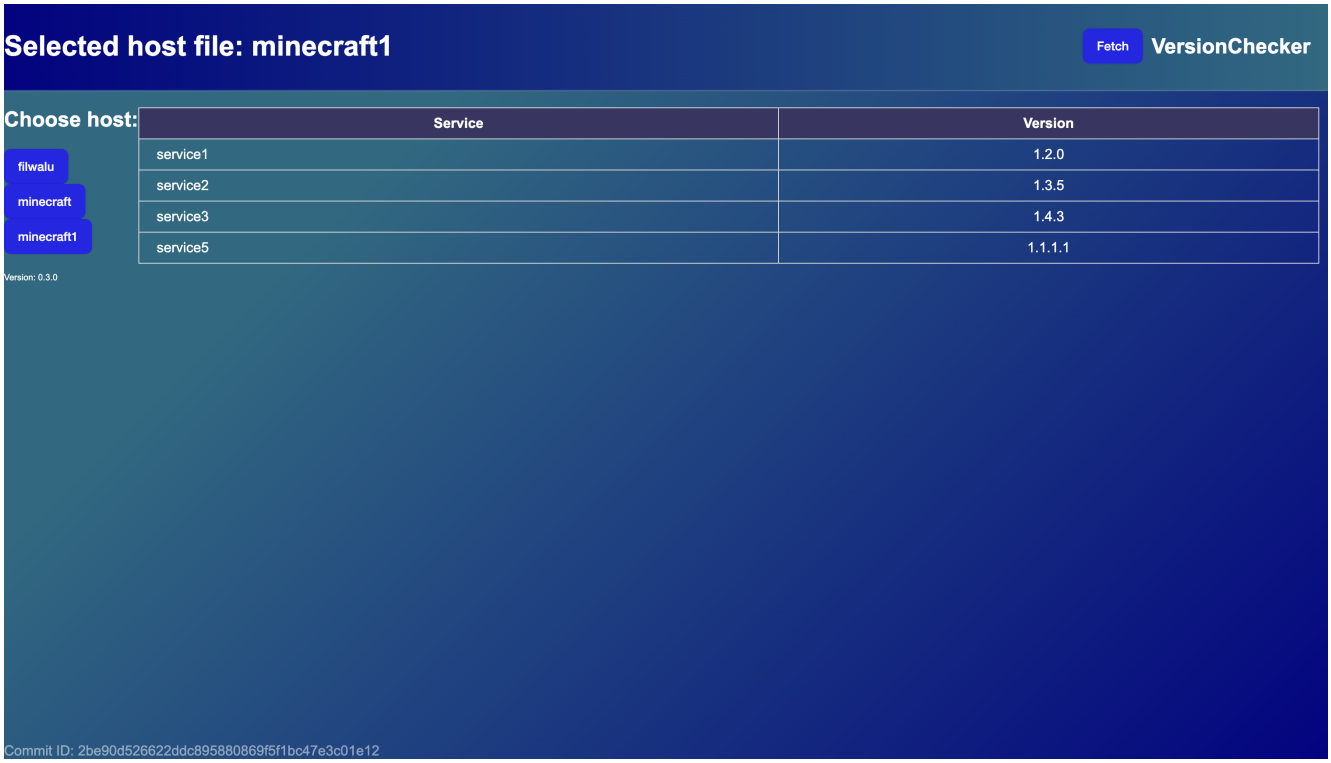
Projekt Aplikacji pozwalającej na szybkie sprawdzenie wersji serwisów na serwerach Linux oferuje intuicyjny i prosty w obsłudze interfejs użytkownika, który umożliwia szybkie sprawdzenie wersji serwisów na serwerach Linux.

Aplikacja umożliwia użytkownikom wykonanie następujących akcji:

- Sprawdzenie wersji serwisów na danym hoście.
- Wywołanie funkcji, która sprawdzi czy wersje serwisów zmieniły się na danym hoście i w razie potrzeby je zaaktualizuje na stronie.
- Zmianę wyświetlanych hostów za pomocą przycisków.

Interfejs skupia się na minimalizmie i łatwości nawigacji, co pozwala na szybkie odnalezienie potrzebnych informacji i funkcji.

W tej sekcji zostaną umieszczone zrzuty ekranu przedstawiające kluczowe funkcjonalności aplikacji oraz jej interfejs użytkownika.



Rysunek 4.1: Strona główna.

W tej sekcji omówione zostaną kluczowe funkcjonalności aplikacji instrukcje dotyczące ich używania.

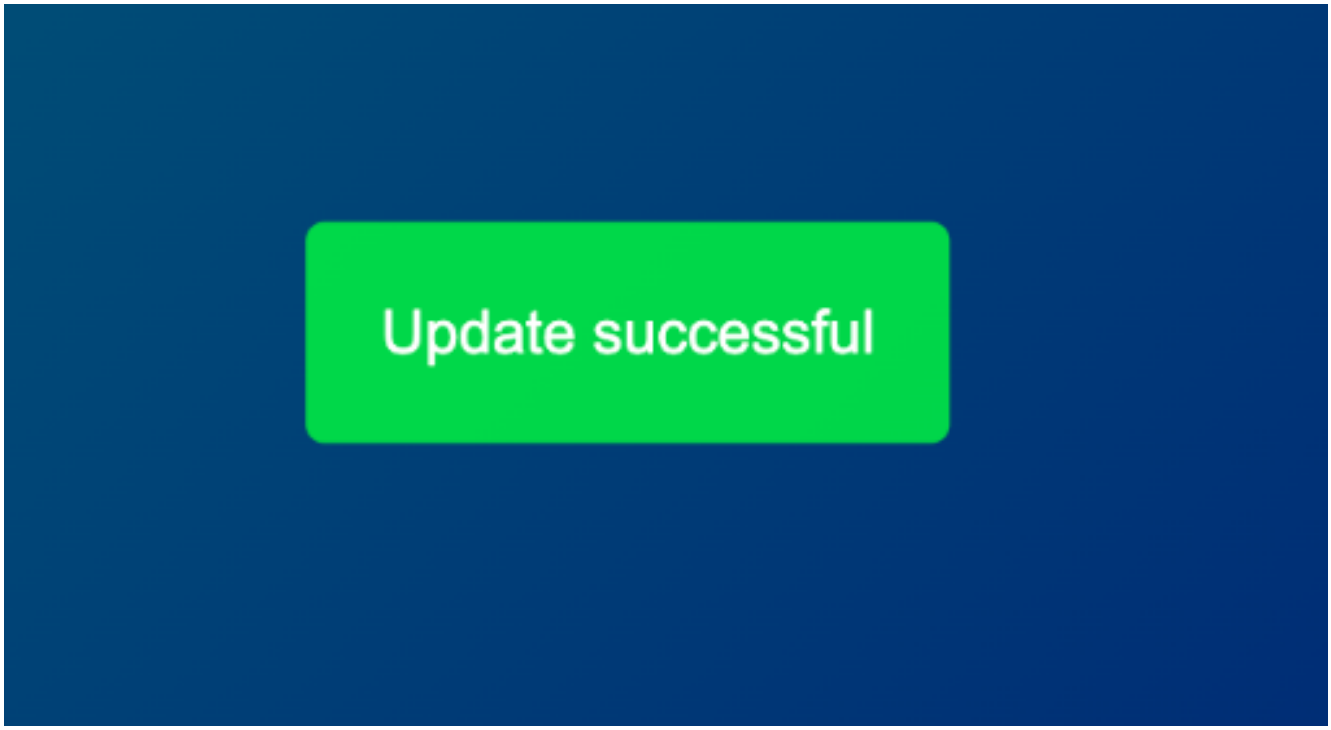
- **Wybór hosta** (*Przyciski z nazwami hostów*): Użytkownik może wybrać hosta, którego wersje serwisów chce sprawdzić, klikając na przycisk z nazwą hosta. Po kliknięciu na przycisk, aplikacja wyświetli tabele z nazwami serwisów oraz ich wersjami.

Rysunek 4.2: Wybór hosta.

- **Pobieranie wersji serwisów** (*Przycisk "Fetch"*): Użytkownik może pobrać wersje serwisów dla wybranego hosta, klikając na przycisk "Fetch". Po kliknięciu na przycisk, aplikacja sprawdzi czy wersje serwisów zmieniły się na danym hoście i w razie potrzeby je zaaktualizuje na stronie.

Rysunek 4.3: Przycisk Fetch.

- **Status pobierania informacji o wersjach serwisów** (*Komunikat "Update Successful"*): Po pobraniu wersji serwisów dla wybranego hosta, aplikacja wyświetli komunikat "Update Successful"informujący użytkownika o pomyślnym pobraniu wersji serwisów.



Rysunek 4.4: Komunikat Update Successful.

4.2 Opis interakcji z użytkownikiem

Aplikacja skupia się na minimalizmie i łatwości nawigacji, co pozwala na szybkie odnalezienie potrzebnych informacji i funkcji. Interfejs użytkownika jest intuicyjny i prosty w obsłudze, dlatego większość błędów jest logowana do pliku app.log.

Rozdział 5

Podsumowanie

W ramach projektu Aplikacja pozwalająca na szybkie sprawdzenie wersji serwisów na serwerach Linux szereg prac, w tym:

- Implementacja systemu logowania błędów do pliku app.log za pomocą biblioteki loguru.
- Uruchomienie prostego serwera www działającego na lokalnym hoście, dzięki bibliotece Flask.
- Przygotowanie interfejsu użytkownika na stronie www za pomocą templateów Jinja, kodu CSS oraz JavaScript.
- Wykorzystanie plików JSON do przechowywania danych o serwisach i wersjach, oraz docelowych hostach.
- Wykorzystanie biblioteki paramiko do obsługi połączeń SSH do docelowych hostów, za pomocą prywatnego klucza.

W dalszej kolejności planowane są następujące prace rozwojowe:

- Konteneryzacja aplikacji z wykorzystaniem Docker, co ułatwi wdrożenie i skalowalność.
- Rozbudowa obsługi wyjątków dla zwiększenia stabilności i niezawodności aplikacji.
- Wdrożenie obsługi requestów /POST w celu możliwości dodania pliku hosta bez potrzeby umieszczania go bezpośrednio w plikach.

Bibliografia

[1] Flask Documentation. *Pallets Projects*. Available at: https://flask.palletsprojects.com/en/3.0.x/.

[2] Paramiko Documentation. *Paramiko Development Team*. Available at: https://docs.paramiko.org/en/latest/.

[3] Jinja Documentation. *Pallets Projects*. Available at: https://jinja.palletsprojects.com/en/3.1.x/.

Spis rysunków

3.1 Diagram Gantta 9
3.2 Diagram Gantta 9
3.3 Diagram Gantta 10

4.1 Strona główna. 11
4.2 Wybór hosta. 12
4.3 Przycisk Fetch. 12
4.4 Komunikat Update Successful. 13