Bezpieczeństwo serwerów i aplikacji web

Kacper Zawadzki 279416
Filip Wójcik 279531
Wojciech Nachaj 279521

Laboratorium 1 - 13.10.2025

Koncepcja i Cel Strony
Głównym celem strony jest codzienne dostarczanie ciekawostek użytkownikom w przystępnej formie. Ma ona służyć zarówno rozrywce, jak i poszerzaniu wiedzy, jednocześnie tworząc zaangażowaną społeczność poprzez funkcje interaktywne.

Funkcjonalności

Wyświetlanie Faktów

- Strona Główna: Codziennie losowo wybierany fakt (tzw. *Fakt Dnia*) jest wyświetlany prominentnie. Może to być duży nagłówek z krótkim opisem i ewentualnie powiązanym obrazem/grafiką.
- Archiwum Faktów: Użytkownicy mogą przeglądać archiwum wszystkich opublikowanych faktów. Możliwe filtry/sortowanie według daty, popularności, kategorii.
- Losowy Fakt (Na Żądanie): Przycisk "Wylosuj inny fakt" dla natychmiastowego dostępu do kolejnej ciekawostki, niezależnie od Faktu Dnia.
- Kategorie: Fakty powinny być opatrzone tagami/kategoriami (np. *Historia, Nauka, Kosmos, Natura, Technologia*), co ułatwi ich późniejsze wyszukiwanie.

Logowanie: Pełne Funkcje dla Zalogowanych

- Rejestracja/Logowanie: Standardowy system z użyciem e-mail/hasło lub integracją z kontami społecznościowymi (np. Google, Facebook).
- Anonimowość (Ograniczona):
    - Przeglądanie: Wszyscy użytkownicy (w tym niezalogowani) mają dostęp do przeglądania faktów i archiwum.
    - Komentowanie: Dostępna jest opcja komentowania jako "Gość" (anonimowo), lecz komentarze te mogą podlegać ściślejszej moderacji lub mieć ograniczoną widoczność, np. bez możliwości dodawania "polubień" czy odpowiadania na nie.
- Profil Użytkownika: Zalogowani użytkownicy mają swój profil, gdzie mogą zobaczyć:

    - Swoje dodane fakty (oczekujące i opublikowane).
    - Swoje komentarze.
    - Ulubione fakty (jeśli wprowadzona zostanie funkcja "ulubionych").
    - Statystyki aktywności (np. ile faktów zatwierdzono, ile komentarzy dodano).

Komentarze

- System Komentarzy: Pod każdym faktem (zarówno Faktem Dnia, jak i w archiwum) musi znaleźć się system komentarzy.
- Funkcjonalność: Odpowiadanie na komentarze (tworzenie wątków), dodawanie reakcji/polubień, zgłaszanie nieodpowiednich treści.
- Moderacja: Wprowadzenie prostych zasad i systemu moderacji (automatycznej dla wulgaryzmów + manualnej dla zgłoszonych treści). Zalogowani użytkownicy mogą mieć większe uprawnienia, np. automatycznie wyświetlane komentarze.

Dodawanie Własnych Faktów

- Formularz: Zalogowani użytkownicy mają dostęp do formularza, umożliwiającego proponowanie nowych faktów (tytuł, treść, źródło, kategoria, opcjonalnie obrazek).
- Proces Weryfikacji:
    - Poczekalnia/Weryfikacja: Wszystkie zgłoszone fakty trafiają do poczekalni i muszą zostać zatwierdzone przez moderatora (lub administratora) przed publikacją, aby zachować jakość i wiarygodność treści.
    - Informacja Zwrotna: Użytkownik, który zgłosił fakt, otrzymuje powiadomienie o jego statusie (zatwierdzony/odrzucony).

Technologie
Front-end: HTML/PHP
Back-End: Flask(Python)
Baza Danych: MySQL

Tabele - podstawowa struktura:

1. Użytkownicy (users): id, username, email, password_hash, rola (np. user, moderator, admin), data_rejestracji.
2. Fakty (facts): id, tytul, tresc, zrodlo (link), kategoria, data_dodania, data_publikacji, status (np. roboczy, oczekujący, opublikowany), user_id_autora.
3. Komentarze (comments): id, fact_id, user_id (może być NULL dla anonimowych/gości), pseudonim_goscia (dla gości), tresc, data_dodania, parent_comment_id (do wątkowania).
4. Reakcje (reactions): id, comment_id lub fact_id, user_id, typ_reakcji (np. 'like', 'love').
5. Kategorie (categories): id, nazwa.

Laboratorium 2 - 20.10.2025

L2. Implementacja podstawowej funkcjonalności aplikacji
- Implementacja funkcjonalności w środowisku DEV
- Krótka prezentacja działania aplikacji

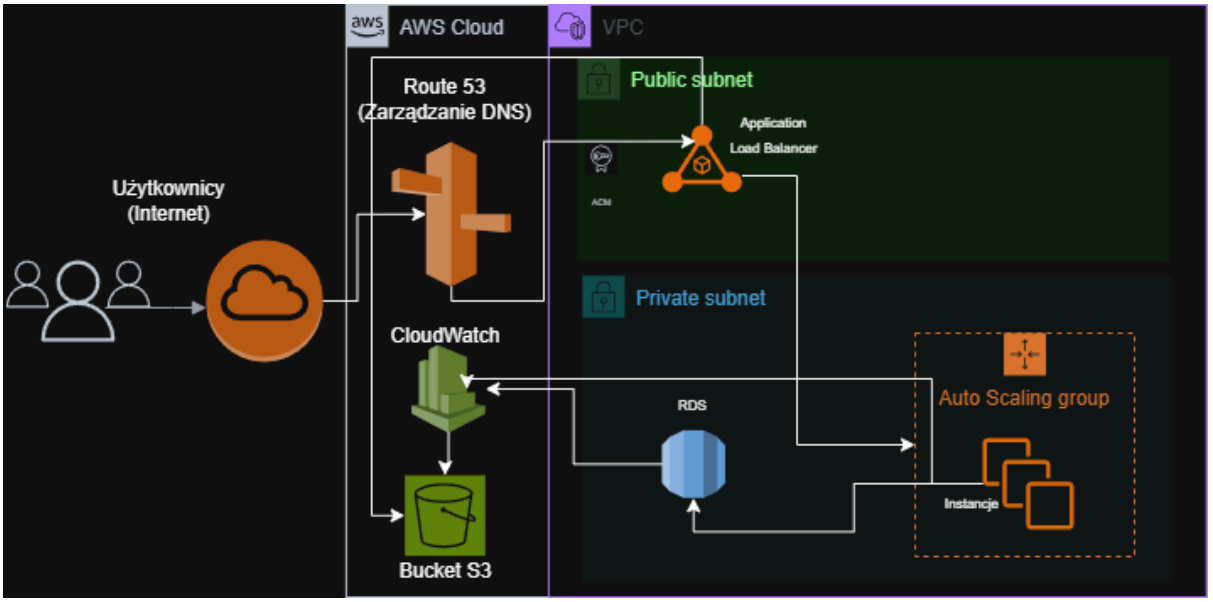
Zaimplementowano podstawowe funkcjonalności w środowisku DEV.

Laboratorium 3 - 27.10.2025

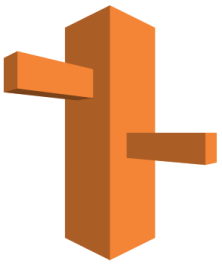
L3. Projekt architektury infrastruktury aplikacji - środowisko cloud
- Load balancer (*ze skalowaniem)
- Baza danych gotowa do skalowania środowiska
- Domena publiczna (www.factshub.pl)
- Certyfikat SSL/TLS (nie na serwerze aplikacyjnym)

Środowiskiem cloud będzie AWS.
Nazwa domeny publicznej - *www.factshub.pl*

Schemat projektu architektury infrastruktury aplikacji:



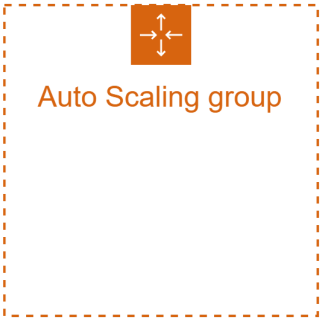
1. **Route 53** — zarządza domeną i rekordami DNS (publiczny Hosted Zone).



2. **ACM (AWS Certificate Manager)** — certyfikat TLS wydany i przypięty do ALB (certyfikat nie jest instalowany na serwerach aplikacyjnych).

3. **Application Load Balancer (ALB)** — w publicznych subnetach, odbiera ruch HTTPS i rozdziela go do grupy targetów.

4. **Auto Scaling Group (ASG)** z Launch Template / Launch Configuration — instancje aplikacyjne w prywatnych subnetach (minimum 2 instancje, rozproszone po Availability Zones).



5. **Amazon RDS (MySQL)** — multi-AZ, z możliwością skalowania (read replicas).

6. **S3** — przechowywanie assetów, logów ALB (access logs) oraz kopii zapasowych eksportowanych danych.

7. **CloudWatch** — monitoring, alarmy, wykrywanie zagrożeń, audyt.

8. **Security Groups & NACLs** — restrykcyjne reguły dostępu: ALB -> ASG -> RDS. (nie uwzględnione na schemacie w celu zwiększenia czytelności)

Wybór usług i krótkie uzasadnienie:

1. **Route 53** - Zarządzana przez AWS usługa DNS (Domain Name System), która pozwala kierować ruch użytkowników do odpowiednich zasobów w chmurze.
2. **ALB** — obsługuje warstwę aplikacji (HTTP/HTTPS), routing na poziomie ścieżek/hostów, integruje się z Target Groups.
3. **ACM** — bezpłatne certyfikaty TLS, zarządzane, odnawiane automatycznie; można je przypiąć do ALB.
4. **ASG + Launch Template** — automatyczne skalowanie instancji aplikacyjnych w oparciu o load/CPU/throughput.

5. **RDS MySQL (Multi-AZ)** — Zarządzana relacyjna baza danych MySQL, działająca w trybie Multi-AZ (Multi-Availability Zone). AWS automatycznie tworzy replikę bazy w drugiej strefie dostępności (AZ) i replikuje dane synchronicznie. RDS upraszcza backup/patching.
6. **Amazon S3** - Wysoce trwała i skalowalna usługa przechowywania obiektów (plików, logów, kopii zapasowych, statycznych treści). W naszym przypadku przechowujemy logi i kopie zapasowe z RDS, ALB oraz Instancji.
7. **Amazon CloudWatch** - System monitoringu i logowania w AWS, który zbiera metryki, logi oraz zdarzenia z usług takich jak EC2, ALB, RDS czy Lambda. W naszym przypadku przekazuje on odpowiednie dane do S3.
8. **Security Groups i Network ACL**:

    Mechanizmy kontroli ruchu sieciowego w ramach VPC:

    - **Security Groups (SG):** działają na poziomie instancji i są stanowe (*stateful*) — automatycznie pozwalają na ruch powrotny.

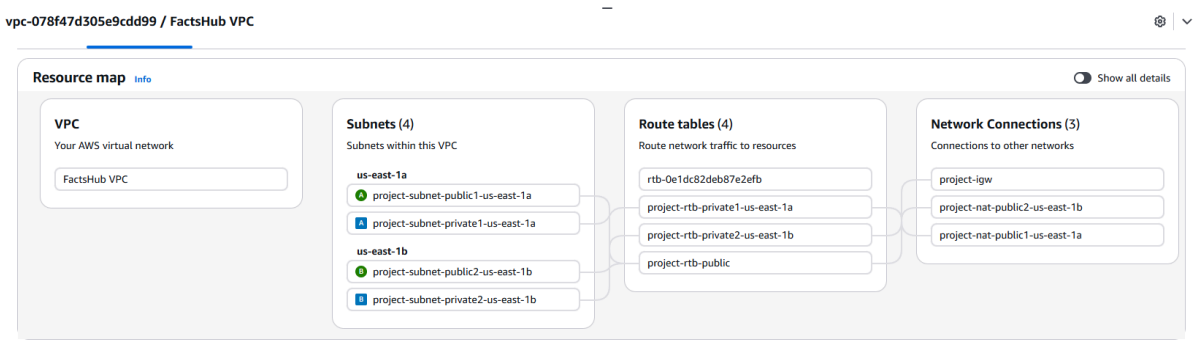
    - **Network ACLs (NACL):** działają na poziomie subnetu i są bezstanowe (*stateless*) — wymagają osobnych reguł dla ruchu wychodzącego i przychodzącego.

Laboratorium 4 - 03.11.2025

L4. Implementacja architektury

- Implementacja architektury w środowisku Cloud Publicznego
- Krótka prezentacja budowy architektury

Etap I: Fundament – Budowa Sieci VPC



- **Liczba Stref Dostępności (AZs): 2**.
- **Liczba podsieci publicznych i prywatnych:** 2 Publiczne i 2 Prywatne.
- **NAT Gateways: 1** NAT Gateway (w Publicznej Podsieci). *Pozwoli to serwerom w Prywatnej Podsieci (EC2, RDS) pobierać aktualizacje z Internetu, ale sam Internet nie będzie miał do nich dostępu.*

Etap II: Zaplecze – Wdrożenie Bazy Danych (RDS)

Security Groups (4) Info

Actions | Export security groups to CSV | Create security group

Find security groups by attribute or tag

Name	Security group ID	Security group name	VPC ID	Description	Owner
–	sg-0d8f620ea1b79c273	FactsHub-RDS-SG	vpc-078f47d305e9cdd99	Security group for RDS MySql in FactsH...	374508014546
–	sg-042d6316f9b165aec	default	vpc-078f47d305e9cdd99	default VPC security group	374508014546
–	sg-0ffc264caad41e1d8	default	vpc-0951058ee5387f9f7	default VPC security group	374508014546
–	sg-0cd1ddb5ff3a6a443	FactsHub-EC2-SG	vpc-078f47d305e9cdd99	Security group for EC2 instances in Fact...	374508014546

Utworzono nowe Security Groups - dla RDS (otwarty port **3306** dla MySQL) oraz EC2. Security Group dla RDS przepuszcza tylko source z SG EC2.

Utworzono instancję RDS - z silnikiem MySQL, typem db.t3.micro, z single AZ deployment oraz VPC utworzonym wcześniej i SG z RDS.

Etap III: Aplikacja – Serwery i Skalowanie (EC2 & ASG)

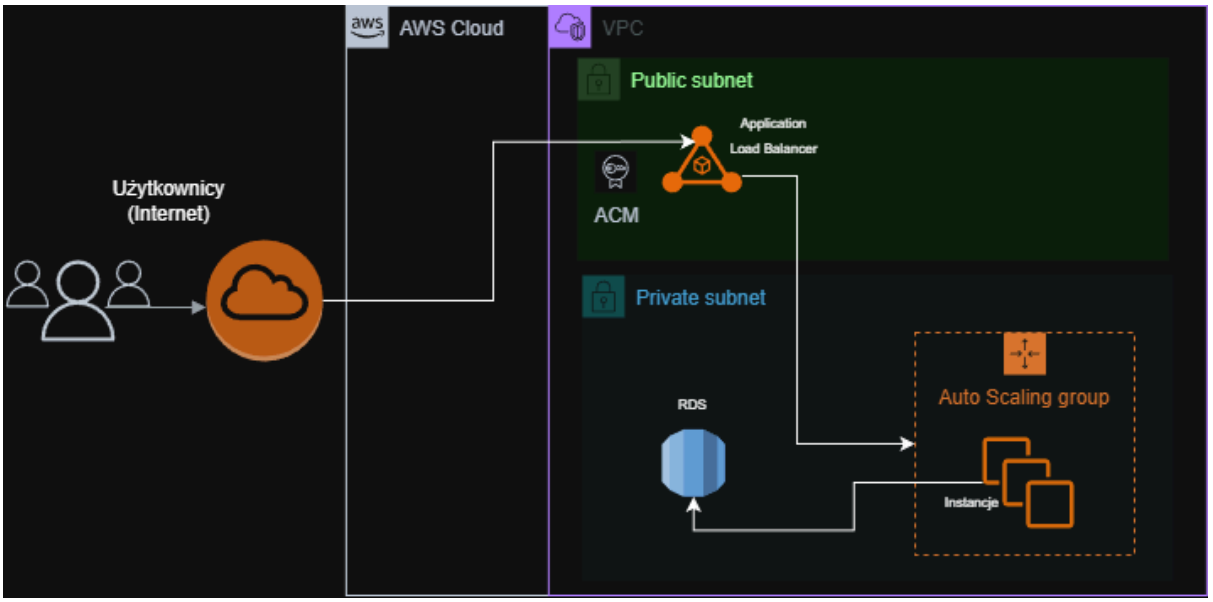
- Utworzono SG dla load balancera. Zmienione zostały reguły inbound dla SG EC2, pozwalające na wejście HTTP od SG ALB.
- Utworzono Launch Template z gotowym skryptem, który inicjuje instalacje podstawowych programów dla działających instancji. Gwarantuje połączenie z RDS i używa zmiennych środowiskowych, co zostało też zmienione w skryptach .py. link do repo GIT: https://github.com/Kapo-cybersec/FactsHubProject
- Utworzono Auto Scaling Group w istniejącym VPC, dwoma prywatnymi podsieciami, z planowaną i minimalną ilością instancji 1, a maksymalną 2.

Etap IV: Front – Dostęp Publiczny (ALB & Route 53)

- Utworzono nowy target group który kieruje ruch do ASG na porcie HTTP (80).
- Target group przesyła ruch na port **5000** - port na którym działa Flask.

ACM zostało wdrożone. Domena została postawiona za pomocą darmowej strony do tworzenia własnych domen (https://dash.infinityfree.com/login). Route 53 jest płatną usługą, niemożliwą do nabycia poprzez darmowy plan AWS.

Obecna architektura nie uwzględnia jeszcze bucketa **S3** i **CloudWatcha** (skonfigurowanego) więc okrojona jest do następującej postaci:



Laboratorium 5 - 17.11.2025

Brak możliwości realizacji etapu za pomocą CodeDeploy i S3 - free tier account na to nie pozwala, dlatego zdecydowaliśmy się na mnie wydajne podejście w postaci **"Rolling Update" (Instance Refresh)** sterowane z GitHuba

Implementacja CI/CD (Continuous Integration / Continuous Deployment)

W celu automatyzacji procesu wdrażania zmian w aplikacji **FactsHub**, zaimplementowano bezobsługowy potok (pipeline) wdrażania oparty na **GitHub Actions** oraz natywnych mechanizmach AWS.

Architektura rozwiązania:

- **Repozytorium:** GitHub (Przechowywanie kodu źródłowego).
- **Orkiestrator:** GitHub Actions (Zarządzanie procesem).
- **Metoda wdrażania:** Rolling Update via **AWS Auto Scaling Instance Refresh**.
- **Infrastruktura:** EC2 (Amazon Linux) w Auto Scaling Group (ASG) za Application Load Balancerem (ALB).

Przebieg procesu wdrożenia:

1. **Push do gałęzi `main`:** Każde wysłanie zmian do repozytorium automatycznie uruchamia zdefiniowany workflow (`.github/workflows/refresh.yml`).
2. **Inicjacja odświeżania:** GitHub Actions, autoryzując się za pomocą `GitHub Secrets` (Access Key/Secret Key), wysyła do AWS polecenie `start-instance-refresh` dla grupy Auto Scaling.
3. **Wymiana instancji:**
    - ASG powołuje nową instancję EC2.
    - Podczas startu instancja wykonuje skrypt **User Data** (zdefiniowany w Launch Template).
    - Skrypt automatycznie instaluje zależności systemowe, klonuje najnowszy kod z repozytorium GitHub i uruchamia aplikację Flask jako usługę systemową (`systemd`).
4. **Weryfikacja i czyszczenie:** Po poprawnym uruchomieniu nowej instancji i przejściu Health Checków, stara instancja z poprzednią wersją kodu jest automatycznie terminowana.

W trakcie realizacji etapu napotkano na dwie trudności:

- W przypadku usunięcia i ponownego dodania usług na AWS - należy pamiętać, aby zaktualizować tablice routingu dla NAT. Wchodzimy **VPC** -> **Route tables** tam najpewniej zobaczymy wpis “Blackhole” -> zmieniamy starego NAT’a na nowego. Jest to spowodowane oszczędnością na zasobach AWS

- W trakcie testów pipeline'a przeoczono edycję repozytorium GitHub w templacie - efekt byl taki, że mimo edycji plików html i faktu, że aws wdrażał "refresh" wygląd strony pozostawał taki sam

Update:

Zmieniono repozytorium GitHub na prywatne - w ten sposób zapewniono dodatkową warstwę ochrony, która uniemożliwia użytkownikom z zewnątrz odczytanie kodu źródłowego aplikacji. Sposób w jaki zostało to zaimplementowane opisany jest poniżej.

Rozwiązanie opiera się na autoryzacji z użyciem kluczy SSH, gdzie klucz publiczny jest znany GitHubowi, a klucz prywatny jest bezpiecznie przechowywany w AWS.

Wykorzystane komponenty:

- **GitHub Deploy Key (Klucz Wdrożeniowy):** Para kluczy SSH (publiczny i prywatny).
- **AWS Secrets Manager:** Usługa AWS do bezpiecznego przechowywania klucza prywatnego.
- **AWS IAM Role (Profil Instancji EC2):** Mechanizm nadawania uprawnień instancji EC2 do odczytu sekretu.

Przebieg Implementacji

Proces implementacji składał się z czterech głównych kroków:

Krok 1: Generowanie i Konfiguracja Klucza Wdrożeniowego (Deploy Key)

1. Lokalnie wygenerowano nową, dedykowaną parę kluczy SSH (`ssh-keygen`).
2. **Klucz publiczny** (`.pub`) został dodany do prywatnego repozytorium GitHub w sekcji **Settings -> Deploy keys**. Klucz ten otrzymał uprawnienia **tylko do odczytu (Read-only)**.

Krok 2: Bezpieczne Przechowywanie Klucza Prywatnego

1. Ze względów bezpieczeństwa, wygenerowany **klucz prywatny** (zawartość pliku `id_rsa`) został zapisany jako nowy sekret w usłudze **AWS Secrets Manager**.
2. Takie podejście eliminuje ryzyko wycieku klucza, ponieważ nie jest on nigdzie "na stałe" zapisany w kodzie ani konfiguracji.

Krok 3: Nadanie Uprawnień Instancjom EC2 (Rola IAM)

1. Stworzono nową **Rolę IAM** (`ec2-factshub-role-ssh`) z zaufaniem dla usługi EC2.
2. Do roli tej dołączono politykę uprawnień **Inline Policy (JSON)**, która zezwalała **tylko i wyłącznie** na jedną akcję: `secretsmanager:GetSecretValue` i tylko dla jednego, konkretnego zasobu (ARN naszego sekretu).

3. W konfiguracji **Launch Template** (Szablon Uruchamiania), w sekcji "Advanced details", wskazano tę rolę jako **"IAM instance profile"**. Od teraz każda nowa instancja EC2 uruchamiała się z tą rolą i przypisanymi jej uprawnieniami.

Krok 4: Modyfikacja Skryptu `User Data`

Skrypt `User Data` został rozbudowany o logikę, która (działając już z uprawnieniami Roli IAM) wykonuje następujące czynności podczas startu instancji:

1. Instaluje klienta AWS CLI (`yum install aws-cli`).
2. Używa polecenia `aws secretsmanager get-secret-value` (podając region i nazwę sekretu), aby pobrać klucz prywatny z AWS Secrets Manager.
3. Zapisuje pobrany klucz do odpowiedniego katalogu na serwerze (`/home/ec2-user/.ssh/id_rsa`).
4. Natychmiast ustawia restrykcyjne uprawnienia dla klucza (`chmod 600`), aby system SSH go zaakceptował.
5. Dodaje `github.com` do listy znanych hostów (`ssh-keyscan`), aby uniknąć prośby o potwierdzenie hosta.
6. Wykonuje polecenie `git clone` używając **protokołu SSH** (`sudo -u ec2-user git clone git@github.com:Kapo-cybersec/FactsHubProject.git .`), które teraz automatycznie używa pobranego klucza do autoryzacji.
7. Dopiero po pomyślnym sklonowaniu repozytorium, skrypt usuwa plik z kluczem prywatnym, przechodzi do instalacji zależności (`pip install`) i uruchomienia aplikacji jako usługi `systemd`.

Laboratorium 6 - 24.11.2025

Elementy bezpieczeństwa CI/CD

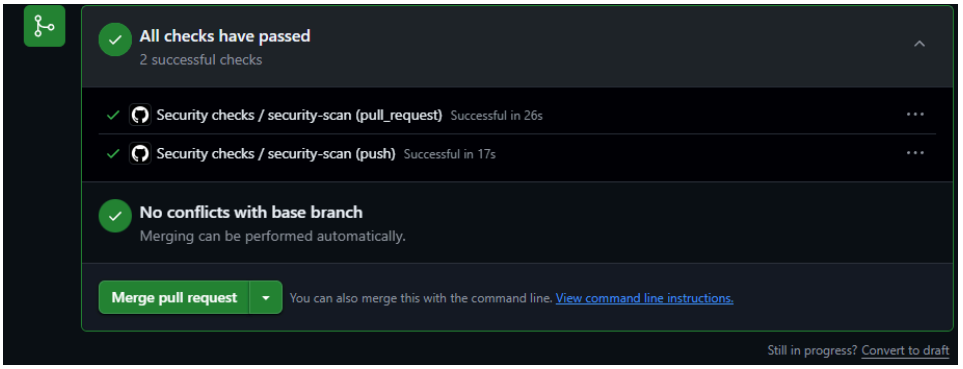
Dostęp do repozytorium – SSH i prywatne repozytorium

Repozytorium projektu jest skonfigurowane jako **prywatne**, a dostęp do niego odbywa się z wykorzystaniem kluczy SSH. Dodano do repozytorium **deploy key** z uprawnieniami read/write.

Łączenie z SSH:

```
Wojtek@DESKTOP-EJEJ1L4 MINGW64 ~
$ git clone git@github.com:Kapo-cybersec/FactsHubProject.git
Cloning into 'FactsHubProject'...
git@github.com: Permission denied (publickey).
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.

Wojtek@DESKTOP-EJEJ1L4 MINGW64 ~
$ eval "$(ssh-agent -s)"
Agent pid 874

Wojtek@DESKTOP-EJEJ1L4 MINGW64 ~
$ ssh-add ~/.ssh/github-deploy-group
Identity added: /c/Users/Wojtek/.ssh/github-deploy-group (Deploy-group-ke

Wojtek@DESKTOP-EJEJ1L4 MINGW64 ~
$ git clone git@github.com:Kapo-cybersec/FactsHubProject.git
Cloning into 'FactsHubProject'...
remote: Enumerating objects: 61, done.
remote: Counting objects: 100% (61/61), done.
remote: Compressing objects: 100% (44/44), done.
remote: Total 61 (delta 18), reused 12 (delta 2), pack-reused 0 (from 0)
Receiving objects: 100% (61/61), 24.43 KiB | 287.00 KiB/s, done.
Resolving deltas: 100% (18/18), done.
```

Osobny branch do testowania zmian w CI/CD
Stworzono osobnego brancha do testowania zmian o nazwie security-test. Po zaakceptowaniu zmian można je zmergować do głownego brancha.

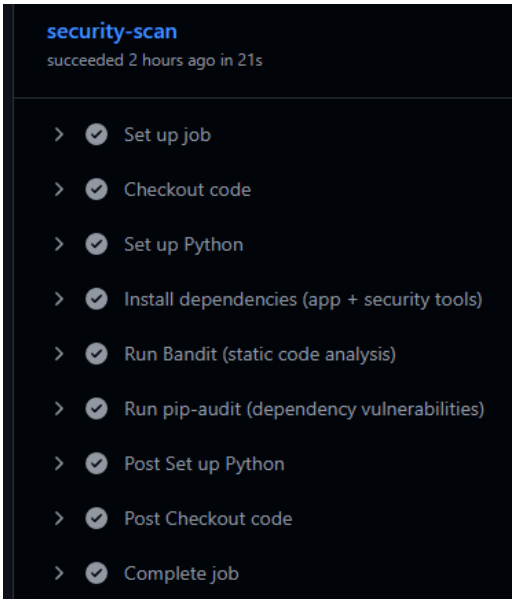


Dodano narzędzia bezpieczeństwa: Bandit i pip-audit
W celu rozdzielenia zależności aplikacji i narzędzi developersko–bezpieczeństwowych utworzono plik requirements-dev.txt, który zawiera:

-odwołanie do podstawowych zależności aplikacji z requirements.txt,

-Bandit – narzędzie do statycznej analizy bezpieczeństwa kodu Pythona (SAST),
-pip-audit – narzędzie do skanowania podatności w zależnościach Pythona.

Następnie dodano workflow, który uruchamia się automatycznie przy **push** i **pull request.**
Workflow wykonuje security scan:

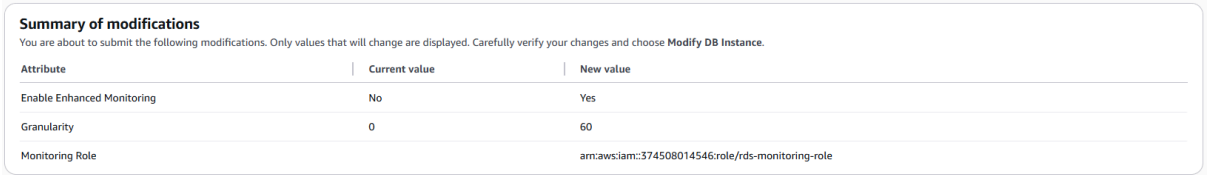


Podsumowanie:
-Repozytorium GitHub aplikacji ustawione na prywatne.
-Autoryzacja z użyciem kluczy SSH. Klucz publiczny jest znany dla GitHub'a, a klucz prywatny jest na AWS
- Wykorzystanie AWS Secrets Manager dla przechowywania klucza prywatnego
- Wykorzystanie AWS IAM Role do nadania ról instancjom EC2
- Osobny branch do testowania
- Workflow dla security-scan z użyciem narzędzi bandit i pip-audit

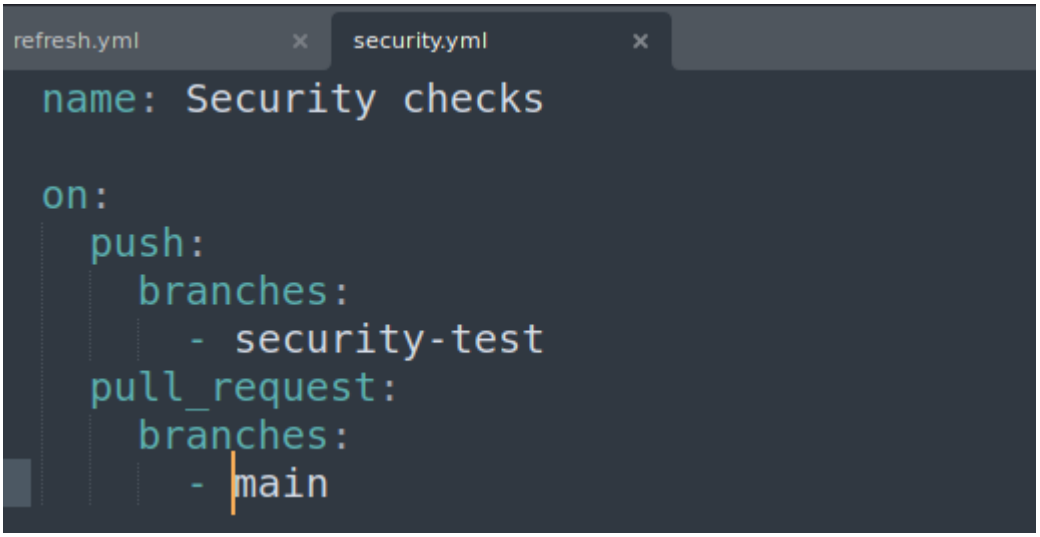
Laboratorium 7 - 01.12.2025

Modyfikacje bezpieczeństwa które zaszły w aplikacji FactsHub:

1. Włączono Enhanced monitoring, który daje dane z systemu operacyjnego co 60 sekund:



2. Zmieniono kod security-test.yml. W tym momencie skan robiony jest tylko w momencie pusha na testowy branch oraz przy próbie pull-requestu na main branch.



3. Poprawiono kod w oparciu o statyczny skan programem Bandit:
   - Błąd Krytyczny: `debug=True`
     Bandit: `A Flask app appears to be run with debug=True... allows the execution of arbitrary code.`

     To jest **najgorsza możliwa dziura**. Jeśli zostawimy debug=True na produkcji, a aplikacja wyrzuci błąd, haker zobaczy w przeglądarce interaktywną konsolę Pythona. Może wpisać import os; os.system('rm -rf /') i wyczyścić serwer. Teraz tryb debugowania opisany jest w zmiennej środowiskowej.

     Zmiana w kodzie:

```python
if __name__ == '__main__':
    debug_mode = os.environ.get('FLASK_DEBUG', 'False') == 'True'
    app.run(debug=debug_mode, host='0.0.0.0', port=5000)
```

   - Błąd Poważny: Hardcoded Secret Key (B105)
     Bandit znalazł: `Possible hardcoded password: 'your-secret-key-change-in-production-12345'`

     Jeśli to zostawimy, każdy kto zna ten ciąg znaków (a jest on teraz w historii Gita), może podrobić swoje ciasteczko sesyjne i zalogować się jako admin bez hasła. Klucz musi pochodzić ze zmiennych środowiskowych (tak jak hasło do bazy).

Rozwiązanie:

```
app = Flask(__name__)
app.secret_key = 'your-secret-key-change-in-production-12345'
app.secret_key = os.environ.get('SECRET_KEY')
```

- Błąd "False Positive": Bind 0.0.0.0 (B104)
  Bandit ostrzega: `Possible binding to all interfaces.`
  To oznacza, że aplikacja nasłuchuje na wszystkich kartach sieciowych (`0.0.0.0`), a nie tylko lokalnie (`127.0.0.1`).

  W normalnym komputerze to ryzyko bo wystawiam serwer na świat. Jednak w naszym środowisku AWS, z ALB oraz SG zostawiamy tak, aby ALB mógł połączyć się z instancjami. SG już blokuje cały ruch z internetu poza ALB.

4. Inne zmiany w plikach
   a) Plik aa.py:
   SQL Injection Fix: W endpointcie `/archive` usunięto sklejanie stringów (`+=`) i `f-stringi`. Teraz używamy bezpiecznej parametryzacji `%s`.
   Rate Limiting: Dodano limity na logowanie (5/min) i rejestrację (3/h) oraz dodawanie komentarzy.
   Audit Logging: Dodano logi `[SECURITY]` przy logowaniu i błędach.
   Talisman: Wymusza nagłówki bezpieczeństwa (HSTS, XSS Protection).
   Secure Cookies: Ciasteczka sesji są teraz oznaczone jako `HttpOnly` i `Secure`.

   b) Plik base.html

   Funkcja `viewFactDetails` w JavaScripcie używała `innerHTML` do wstawiania danych z API (tytuł, treść komentarza) prosto do strony. Gdyby ktoś wpisał w tytule faktu `<img src=x onerror=alert(1)>`, kod by się wykonał u każdego (tzw. DOM-based XSS).

   Naprawa: Dodano funkcję `escapeHtml()`, która zamienia znaki specjalne (, , `<`, `>`) na bezpieczne odpowiedniki.

A. Ochrona przed SQL Injection (Parametryzacja)

- **Problem:** W endpointcie `/archive` parametry filtrowania (`category`) oraz paginacji (`limit`, `offset`) były doklejane bezpośrednio do zapytania SQL jako ciągi znaków (string concatenation). Stwarzało to ryzyko ataku SQL Injection, w którym atakujący mógłby zmodyfikować zapytanie, aby uzyskać nieautoryzowany dostęp do danych.

- **Rozwiązanie:** Zastąpiono konkatenację ciągów znaków mechanizmem **Prepared Statements** (parametryzacja). Zmienne przekazywane są teraz jako lista parametrów (placeholders `%s`) do sterownika bazy danych, który dba o ich bezpieczne escapowanie.

B. Ochrona przed Brute-Force (Rate Limiting)

- **Problem:** Endpointy logowania (`/login`) i rejestracji (`/register`) nie posiadały ograniczeń częstotliwości zapytań. Pozwalało to na masowe próby odgadywania haseł (ataki słownikowe) oraz spamowanie bazy nowymi kontami.
- **Rozwiązanie:** Wdrożono bibliotekę **Flask-Limiter**, która monitoruje liczbę zapytań z poszczególnych adresów IP. Skonfigurowano limity: 5 prób logowania na minutę oraz 3 rejestracje na godzinę. Przekroczenie limitu skutkuje błędem HTTP 429.

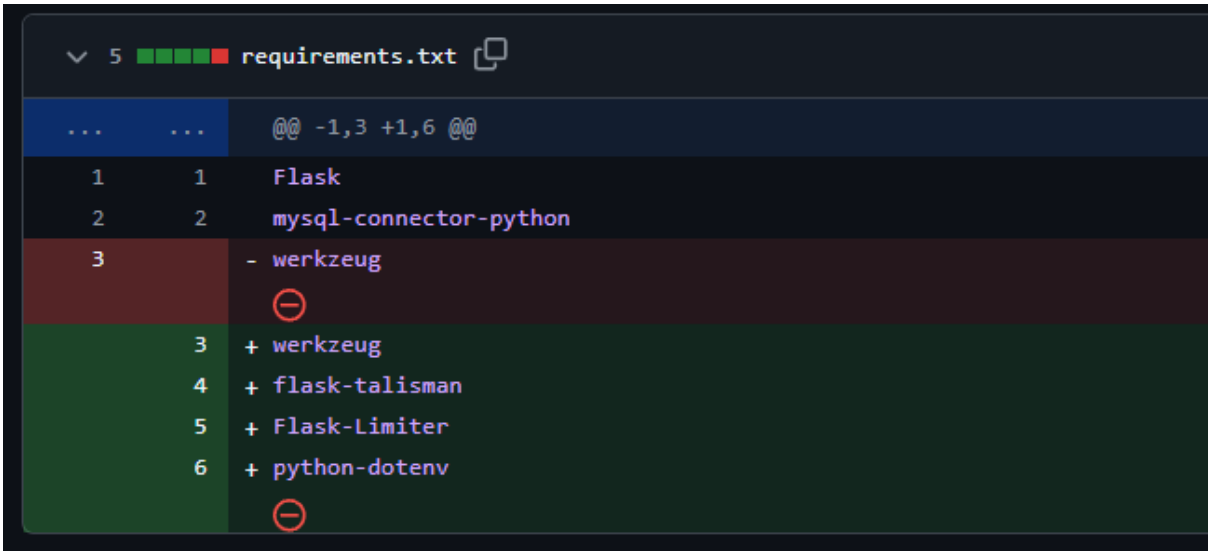
C. Audyt Bezpieczeństwa (Security Logging)

- **Problem:** Brak systematycznego rejestrowania zdarzeń krytycznych (udane/nieudane logowania, zmiany uprawnień) utrudniał detekcję incydentów i analizę powłamaniową.
- **Rozwiązanie:** Zaimplementowano scentralizowany system logowania zdarzeń przy użyciu biblioteki `logging`. Logi zawierają tag `[SECURITY]`, nazwę użytkownika oraz adres IP źródła, co pozwala na łatwe filtrowanie w AWS CloudWatch.

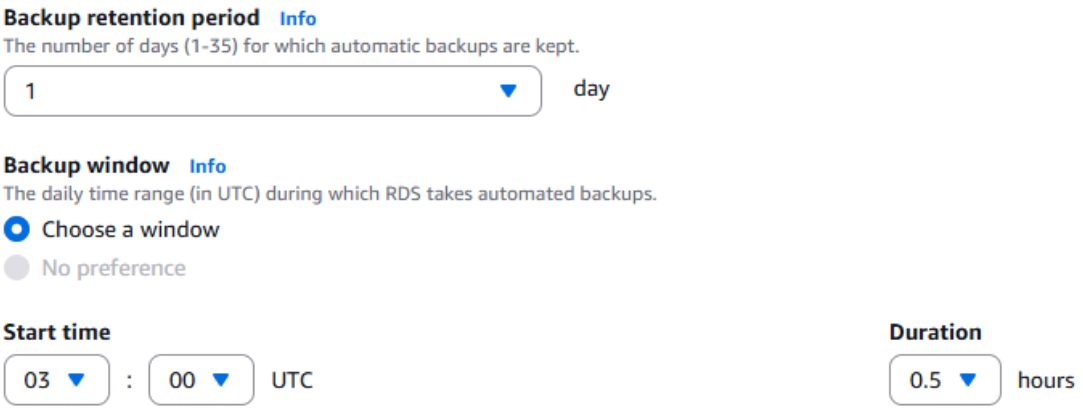
D. Ochrona przed XSS (Cross-Site Scripting)

- **Problem:** Aplikacja renderowała dane pobrane z API (tytuły faktów, treść komentarzy) bezpośrednio do drzewa DOM przy użyciu właściwości `innerHTML` w JavaScript. Pozwalało to na wykonanie złośliwego kodu JavaScript, jeśli został on umieszczony w bazie danych przez atakującego (Stored XSS).
- **Rozwiązanie:** Zaimplementowano funkcję sanityzującą `escapeHtml()` po stronie klienta, która zamienia znaki specjalne HTML na bezpieczne encje przed wstawieniem ich do strony.

Zaktualizowano również plik *requirements.txt*:



Zostały również włączone **backupy** bazy danych.



Backupy trzymane są 1 dzień (ograniczenie dla free tieru), a robią się codziennie między **3:00** a **3:30** UTC. Jest to godzina o zwykle bardzo małym ruchu sieciowym.

Konfiguracyjne elementy bezpieczeństwa aplikacji ()

1. **Bezpieczne połączenie z bazą danych**
   -Dane dostępu do bazy (`host`, `user`, `password`, `database`) pobierane są ze zmiennych środowiskowych, a nie są wpisane na stałe w kodzie.

   -Połączenie jest centralnie konfigurowane przez słownik `DB_CONFIG` i funkcję `get_db_connection()`.

2. **Wymuszenie bezpiecznego połączenia HTTPS**
   -W konfiguracji Flask-Talisman dodano `force_https=True`, co wymusza używanie protokołu HTTPS. **Na moment obecny jest to wyłączone, ze względu na to że połączenie szyfrowane jest jedynie między cloudflare a ALB.**

3. **Konfiguracja sesji użytkownika**
   -Sesje przechowywane są w bezpiecznych ciasteczkach
   `SESSION_COOKIE_SECURE=True`,
   `SESSION_COOKIE_HTTPONLY=True`,
   `SESSION_COOKIE_SAMESITE='Lax'`.
   -Ustawiono maksymalny czas życia sesji wynoszący jedną godzine.:
   `PERMANENT_SESSION_LIFETIME=timedelta(hours=1)`.

   Dodatkowe nagłówki bezpieczeństwa dla każdej odpowiedzi
   Dodano funkcję `@app.after_request add_security_headers`, która uzupełnia nagłówki HTTP:
   -`X-Content-Type-Options: nosniff` – blokada MIME sniffing,
   -`X-Frame-Options: SAMEORIGIN` – ochrona przed clickjackingiem,
   -`X-XSS-Protection: 1; mode=block` – dodatkowa ochrona przed XSS

-`Permissions-Policy: geolocation microphone=(),camera=()` – blokada dostępu do geolokalizacji, mikrofonu kamery,
-`Referrer-Policy: strict-origin-when-cross-origin` ograniczenie przekazywania pełnego URL w nagłówku `Referer`.
-Usuwany jest nagłówek `Server`, aby nie ujawniać informacji o używanym serwerze.

WAF

Wdrożenie Web Application Firewall (WAF) w modelu hybrydowym (Cloudflare + AWS)

Uzasadnienie wyboru technologii

Ze względu na ograniczenia środowiska laboratoryjnego (konto AWS Academy/Student), które blokuje dostęp do natywnej usługi AWS WAF, zdecydowano się na wdrożenie rozwiązania zewnętrznego. Wybrano **Cloudflare** działające w modelu **Reverse Proxy**.

Takie podejście pozwoliło na:

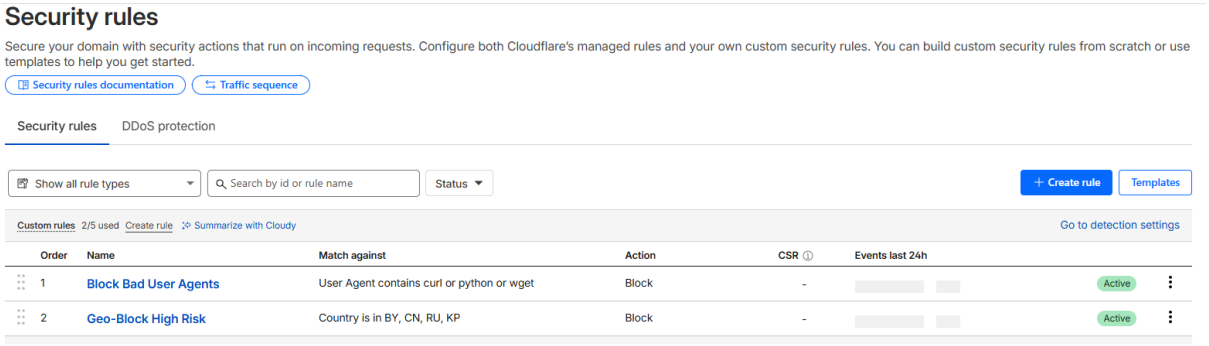
- Ochronę aplikacji przed atakami w warstwie aplikacji (L7 modelu OSI), takimi jak SQL Injection czy XSS.
- Zapewnienie ochrony przed atakami DDoS.
- Ukrycie rzeczywistej infrastruktury AWS (Application Load Balancer) przed bezpośrednim dostępem z Internetu.

Architektura logiczna

Ruch sieciowy nie trafia bezpośrednio do infrastruktury AWS. Cloudflare staje się "tarczą" (Edge Layer) pośredniczącą w komunikacji.

W ramach projektu zaprojektowano następujące warstwy ochrony:

- **Geo-Blocking (Geolokalizacja):**
  - Skonfigurowano regułę WAF blokującą ruch z krajów o wysokim ryzyku cyberzagrożeń (np. Rosja, Chiny), z których nie spodziewamy się legalnych użytkowników. Pozwala to na drastyczne zredukowanie "szumu" w logach i prób automatycznych skanów portów.
- **Ochrona przed skryptami i prostymi botami:**
  - Skonfigurowano regułę WAF blokującą ruch pochodzący z skryptów napisanych w Pythonie lub narzędzia konsolowego (curl, wget)



Match against	Action
User Agent contains curl or python or wget	Block
Country is in BY, CN, RU, KP	Block

ANTYDDOS

Zastosowane rozwiązanie hybrydowe (Cloudflare + AWS) jest standardem przemysłowym. Pozwala na oddzielenie warstwy bezpieczeństwa od warstwy aplikacyjnej. W przypadku ataku DDoS, to globalna sieć Cloudflare przyjmuje uderzenie, chroniąc zasoby AWS (EC2) przed przeciążeniem i skalowaniem kosztów (Auto Scaling nie uruchomi dodatkowych maszyn, ponieważ zły ruch zostanie wycięty wcześniej).

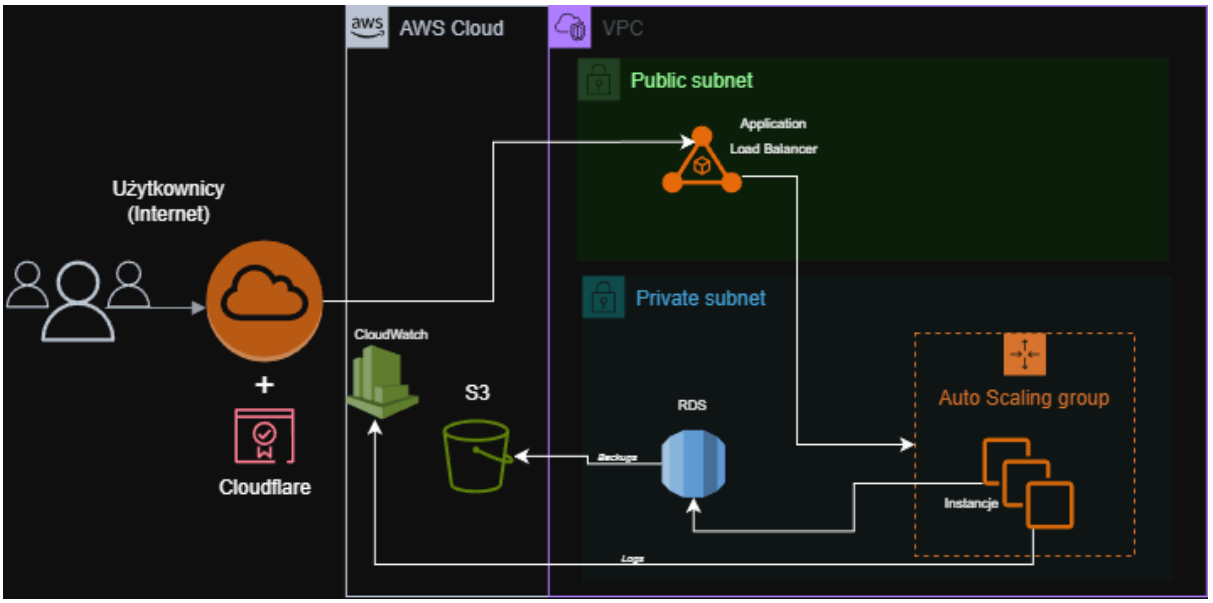
CDN

Treści statyczne (obrazy, skrypty) są cache'owane na serwerach brzegowych (Edge Servers), co redukuje opóźnienia (latency) dla użytkownika końcowego i zmniejsza koszty transferu danych wychodzących (Data Egress) z chmury AWS.

CDN oraz AntyDDoS działają automatycznie na cloudflare, dodatkowo warto zauważyć, że CloudFlare oferuje nam Origin Cloaking w wyniku czego adres naszego serwera jest nie widoczny dla klientów.

Cloudflare wystawia również certyfikat SSL, dzieki czemu nie musimy już korzystać z usługi AWS ACM.

Obecna topologia naszej aplikacji wygląda następująco:



Laboratorium 8 - 08.12.2025

Implementacja automatycznego utwardzania systemu operacyjnego (OS Hardening) zgodnie ze standardami CIS/STIG.

W celu zminimalizowania powierzchni ataku (Attack Surface Reduction) oraz spełnienia wymogów audytowych, wdrożono zautomatyzowany proces utwardzania systemu (Hardening) realizowany na etapie inicjalizacji instancji (User Data). Konfiguracja opiera się na wytycznych **CIS Benchmark** (Center for Internet Security) oraz standardach wojskowych **STIG**.

Proces obejmuje trzy kluczowe obszary bezpieczeństwa:

1. **Zabezpieczenie demona SSH (Access Control):**
    - Całkowite zablokowanie logowania dla konta `root` (wymuszenie eskalacji uprawnień przez `sudo`).
    - Blokada pustych haseł oraz ograniczenie liczby prób autoryzacji (ochrona przed atakami Brute-Force).
    - Implementacja "Legal Banner" (zgodnie z wymogami prawnymi) informującego o monitorowaniu aktywności i konsekwencjach nieautoryzowanego dostępu.
2. **Utwardzanie Kernela (Network Stack Hardening):**
    - Modyfikacja parametrów `sysctl` w celu ochrony przed atakami sieciowymi.
    - Wyłączenie odpowiedzi na ICMP Echo (Ping), co czyni serwer "niewidzialnym" dla podstawowych skanerów sieciowych.

- Ochrona przed atakami typu IP Spoofing oraz blokada nieautoryzowanego przekierowania pakietów (ICMP Redirects).

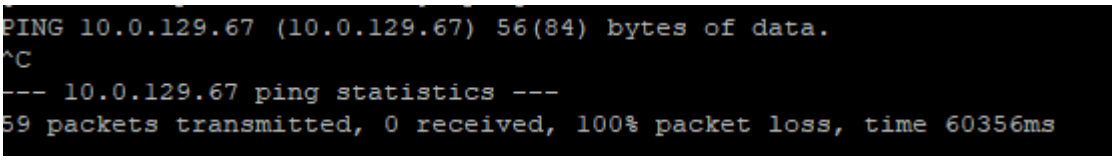
3. **Patch Management:**
   - Automatyczna instalacja krytycznych poprawek bezpieczeństwa (`--security`) przy każdym starcie nowej instancji w grupie Auto Scaling.

```
# ================================================
# HARDENING SYSTEMU (CIS/STIG COMPLIANCE)
# ================================================
dnf upgrade -y --security
echo "[SECURITY] Rozpoczynam procedurę utwardzania systemu"

# ---- Utwardzanie SSH (Ochrona przed włamaniami) ----

CONFIG_SSH="/etc/ssh/sshd_config"

# Zabroń logowania jako root (wymusza użycie ec2-user)
sed -i 's/^#PermitRootLogin yes/PermitRootLogin no/' $CONFIG_SSH
sed -i 's/^PermitRootLogin yes/PermitRootLogin no/' $CONFIG_SSH

# Zabroń pustych haseł
sed -i 's/^#PermitEmptyPasswords no/PermitEmptyPasswords no/'
$CONFIG_SSH

# Max 3 próby hasła (Stop Brute Force)
sed -i 's/#MaxAuthTries.*/MaxAuthTries 3/' $CONFIG_SSH

# Rozłącz leniwych adminów po 5 minutach bezczynności
sed -i 's/#ClientAliveInterval.*/ClientAliveInterval 300/' $CONFIG_SSH
sed -i 's/#ClientAliveCountMax.*/ClientAliveCountMax 0/' $CONFIG_SSH

# ---- Legal Banner (Wymóg audytowy) ----

# Tworzymy plik z groźnym ostrzeżeniem
cat <<EOF > /etc/issue.net
**********************************************************************
****
WARNING: RESTRICTED AREA

This system is for the use of authorized users only. Activities on
this system may be monitored and recorded by system personnel.

Unauthorized access or use of this system may result in civil and
criminal penalties.
**********************************************************************
****
EOF

# Włączamy baner w configu SSH
echo "Banner /etc/issue.net" >> $CONFIG_SSH
```

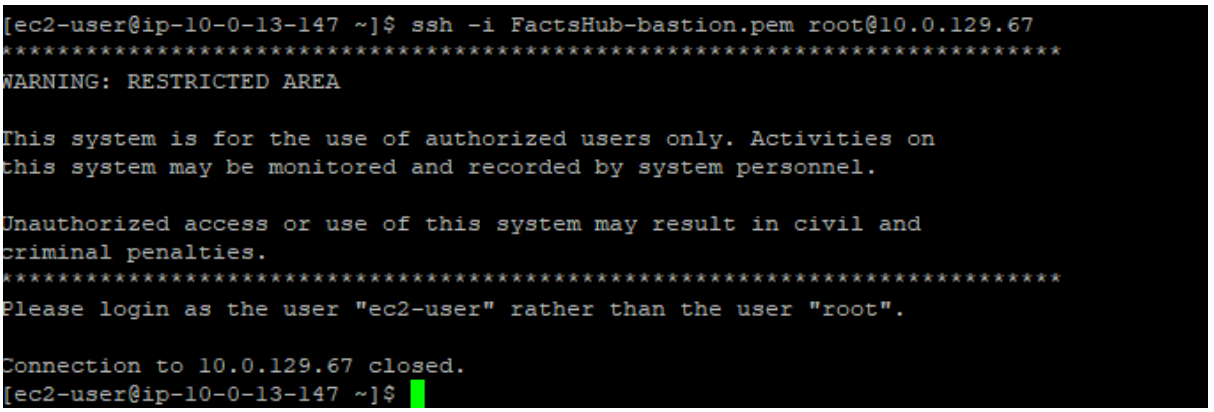
```
# Restart SSH, żeby zmiany weszły w życie
systemctl restart sshd

# ---- Utwardzanie Kernela (Network Hardening) ----

# Wyłączamy Ping i zabezpieczamy przed spoofingiem
cat <<EOF > /etc/sysctl.d/99-security-hardening.conf
# Ignoruj pingi (ICMP Echo)
net.ipv4.icmp_echo_ignore_all = 1
# Ochrona przed IP Spoofing (weryfikacja źródła)
net.ipv4.conf.all.rp_filter = 1
net.ipv4.conf.default.rp_filter = 1
# Nie akceptuj przekierowań (Redirects) - ochrona przed MITM
net.ipv4.conf.all.accept_redirects = 0
net.ipv4.conf.default.accept_redirects = 0
EOF

# Załaduj nowe ustawienia kernela
sysctl -p /etc/sysctl.d/99-security-hardening.conf

# ---- Zabezpieczenie plików systemowych (File Permissions) ----

# Tylko root może dotykać harmonogramu zadań
chown root:root /etc/crontab
chmod 600 /etc/crontab

echo "[SECURITY] Hardening zakończony pomyślnie."

# ================================================
```

Przykłady sprawdzenia zabezpieczeń:

- Ping na serwer:



- logowanie SSH jako root oraz banner:



Laboratorium 9 - 15.12.2025

Część 1

Implementacja Dynamicznych Testów Bezpieczeństwa (DAST)

Cel: Rozszerzenie procedur CI/CD o weryfikację bezpieczeństwa uruchomionej aplikacji w czasie rzeczywistym (Dynamic Application Security Testing) przy użyciu narzędzia OWASP ZAP.

1. Koncepcja i różnice względem SAST

W poprzednich etapach (Laboratorium 6) wdrożono statyczną analizę kodu (SAST) przy użyciu narzędzia Bandit. SAST analizuje kod źródłowy "na sucho". Obecne wdrożenie DAST (OWASP ZAP) ma na celu przetestowanie aplikacji **z zewnątrz**, symulując zachowanie potencjalnego atakującego. Pozwala to na wykrycie błędów, które nie są widoczne w samym kodzie Pythona, takich jak:

- Skuteczność konfiguracji nagłówków bezpieczeństwa na poziomie Load Balancera i Cloudflare.
- Błędy w renderowaniu front-endu (HTML/JS).
- Dostępność zasobów, które powinny być ukryte.

2. Implementacja w GitHub Actions

Utworzono nowy workflow `.github/workflows/dast.yml`, który uruchamia skanowanie na żądanie (`workflow_dispatch`). Takie podejście zapobiega nadmiernemu obciążaniu środowiska produkcyjnego przy każdym commitcie.

Kluczowe elementy konfiguracji:

- **Target:** `https://factshub.pl` – skanowanie odbywa się na domenie publicznej, co pozwala zweryfikować całą ścieżkę sieciową (Cloudflare -> ALB -> EC2).
- **Uprawnienia:** Skonfigurowano uprawnienia `issues: write`, co pozwala automatowi OWASP ZAP na automatyczne tworzenie zgłoszenia (Issue) w repozytorium w przypadku wykrycia podatności.

Fragment konfiguracji (`dast.yml`):

YAML

```
None
name: DAST Security Scan (OWASP ZAP)
on:
  workflow_dispatch:
permissions:
  contents: read
  issues: write
jobs:
  zap_scan:
    runs-on: ubuntu-latest
    steps:
      - name: OWASP ZAP Baseline Scan
        uses: zaproxy/action-baseline@v0.14.0
        with:
          target: 'https://factshub.pl'
          token: ${{ secrets.GITHUB_TOKEN }}
```

3. Przebieg testu i integracja

Zmiany zostały najpierw przetestowane na gałęzi `security-test`, a następnie zmergowane do gałęzi `main`. Uruchomienie workflowu potwierdziło dostępność aplikacji (status HTTP 200 OK) oraz poprawne działanie mechanizmu raportowania.

4. Analiza wyników (Raport OWASP ZAP)

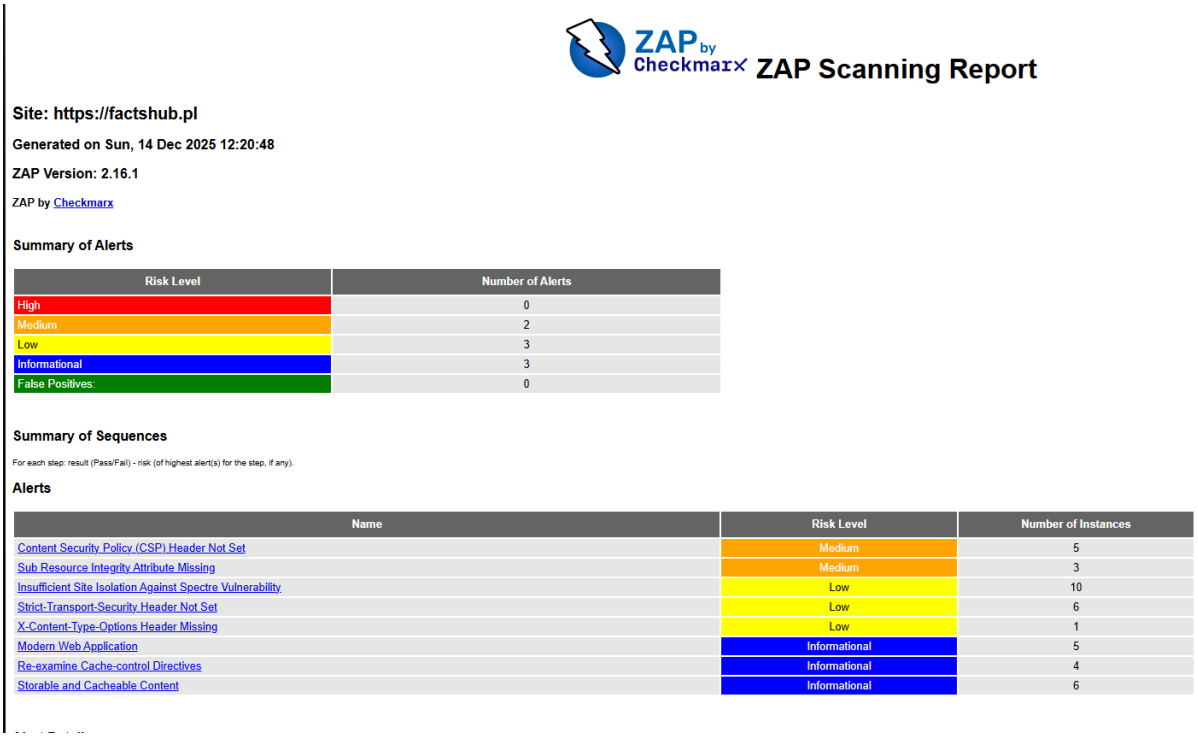
Statystyki Podatności

Skaner nie wykrył podatności o poziomie krytycznym (High), co świadczy o dobrym podstawowym poziomie bezpieczeństwa. Zidentyfikowano jednak obszary wymagające poprawy:

- **Medium Risk (Średnie):** 2 alerty (dotyczące głównie nagłówków CSP i SRI).
- **Low Risk (Niskie):** 3 alerty (HSTS, izolacja witryny).
- **Informational:** 3 alerty (w tym potwierdzenie, że jest to "Modern Web Application").

Wykryte Problemy i Wnioski

1. **Niezamierzone ujawnienie błędu w kodzie Front-end (XSS Prevention)** Analiza przeskanowanych adresów URL w raporcie ujawniła błąd implementacji w kodzie JavaScript. Skaner próbował uzyskać dostęp do adresu: `https://factshub.pl/$%7BescapeHtml(c.image_url)%7D`.
    - **Wniosek:** Funkcja `escapeHtml()`, która miała chronić przed XSS (wdrażana w Lab 7), nie została wykonana przez silnik JavaScript, lecz potraktowana jako zwykły tekst (string). Wskazuje to na użycie nieprawidłowej składni w pliku szablonu (prawdopodobnie brak użycia *template literals* z grawisami ` `). Dzięki DAST wykryliśmy błąd funkcjonalny, który testy statyczne (SAST) pominęły.
2. **Brak nagłówka Content Security Policy (CSP) [Medium Risk]** Alert wystąpił w 5 instancjach.
    - **Opis:** Serwer nie definiuje, z jakich źródeł przeglądarka może ładować zasoby (skrypty, obrazy).
    - **Wniosek:** Jest to typowe dla wczesnej fazy rozwoju aplikacji, ponieważ źle skonfigurowane CSP może zablokować działanie strony. Należy zaplanować wdrożenie CSP w przyszłych sprintach, aby mitygować ataki XSS.
3. **Problem z nagłówkiem HSTS (Strict-Transport-Security) [Low Risk]** Mimo konfiguracji `flask-talisman` w kodzie Python, raport wskazuje brak tego nagłówka w odpowiedziach serwera (6 instancji).
    - **Wniosek:** Nagłówek jest prawdopodobnie wycinany lub nadpisywany przez warstwę pośrednią (Cloudflare lub AWS ALB). Rekomendowanym rozwiązaniem jest włączenie HSTS bezpośrednio w panelu Cloudflare ("Edge Certificates"), zamiast polegać wyłącznie na aplikacji backendowej.
4. **Brak atrybutu "Integrity" dla zasobów zewnętrznych (SRI)** Zgłoszono brak sumy kontrolnej (Sub Resource Integrity) dla czcionek Google Fonts: `<link href="https://fonts.googleapis.com/css2?family=Outfit..."`.
    - **Ryzyko:** Jeśli serwery Google zostałyby skompromitowane, atakujący mógłby wstrzyknąć złośliwy kod CSS/JS do naszej aplikacji. Dodanie atrybutu `integrity` zabezpieczy przed taką ewentualnością.



ZAP by Checkmarx ZAP Scanning Report

Site: https://factshub.pl

Generated on Sun, 14 Dec 2025 12:20:48

ZAP Version: 2.16.1

ZAP by Checkmarx

Summary of Alerts

Risk Level	Number of Alerts
High	0
Medium	2
Low	3
Informational	3
False Positives:	0

Summary of Sequences

For each step: result (Pass/Fail) - risk (of highest alert(s) for the step, if any).

Alerts

Name	Risk Level	Number of Instances
Content Security Policy (CSP) Header Not Set	Medium	5
Sub Resource Integrity Attribute Missing	Medium	3
Insufficient Site Isolation Against Spectre Vulnerability	Low	10
Strict-Transport-Security Header Not Set	Low	6
X-Content-Type-Options Header Missing	Low	1
Modern Web Application	Informational	5
Re-examine Cache-control Directives	Informational	4
Storable and Cacheable Content	Informational	6

Część 2

Testy Bezpieczeństwa Infrastruktury

Przeprowadzono skan portów przy pomocy narzędzie nmap

```
┌──(cali㉿kali)-[~]
└─$ nmap -F -sV factshub.pl
Starting Nmap 7.95 ( https://nmap.org ) at 2025-12-14 11:44 EST
Nmap scan report for factshub.pl (188.114.96.11)
Host is up (0.012s latency).
Other addresses for factshub.pl (not scanned): 188.114.97.11 2a06:98c1:3120::b 2a06:98c1:3121::b
Not shown: 96 filtered tcp ports (no-response)
PORT     STATE SERVICE  VERSION
80/tcp   open  http     Cloudflare http proxy
443/tcp  open  ssl/http Cloudflare http proxy
8080/tcp open  http     Cloudflare http proxy
8443/tcp open  ssl/http Cloudflare http proxy

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 35.63 seconds
```

Test potwierdza, że WAF działa poprawnie, Serwer na AWS (EC2 oraz load balancer) nie jest widoczny. Jedyne co widać to serwery brzegowe cloudflare.

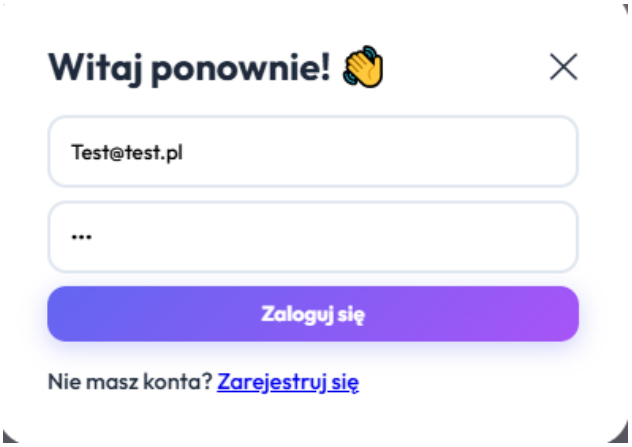
Porty 80 oraz 443 to Standardowe porty dla strony internetowej,
Porty 8080 i 8443 to alternatywne porty HTTP/HTTPS, które Cloudflare domyślnie obsługuje (Caching/Proxy). Nie stanowią zagrożenia, ponieważ ruch i tak przechodzi przez filtry WAF.

Przeprowadzono skan przy użycia narzędzia Nikto
Uruchomiono skaner w celu weryfikacji nagłówków HTTP i konfiguracji serwera webowego.

```
┌──(cali㉿kali)-[~]
└─$ nikto -h factshub.pl
- Nikto v2.5.0
---------------------------------------------------------------------------
+ Multiple IPs found: 188.114.96.11, 188.114.97.11, 2a06:98c1:3120::b, 2a06:98c1:3121::b
+ Target IP:          188.114.96.11
+ Target Hostname:    factshub.pl
+ Target Port:        80
+ Start Time:         2025-12-14 11:53:17 (GMT-5)
---------------------------------------------------------------------------
+ Server: cloudflare
+ /: The anti-clickjacking X-Frame-Options header is not present. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/
Headers/X-Frame-Options
+ /: An alt-svc header was found which is advertising HTTP/3. The endpoint is: ':443'. Nikto cannot test HTTP/3 over QUIC
. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/alt-svc
+ /: The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a
 different fashion to the MIME type. See: https://www.netsparker.com/web-vulnerability-scanner/vulnerabilities/missing-co
ntent-type-header/
+ Root page / redirects to: https://factshub.pl/
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ /robots.txt: contains 9 entries which should be manually viewed. See: https://developer.mozilla.org/en-US/docs/Glossary
/Robots.txt
+ /: Uncommon header 'proxy-status' found, with contents: Cloudflare-Proxy;error=http_request_error.
+ /cdn-cgi/trace: Retrieved access-control-allow-origin header: *.
+ /cdn-cgi/trace: Cloudflare trace CGI found, which may leak some system information.
+ 7962 requests: 0 error(s) and 7 item(s) reported on remote host
+ End Time:           2025-12-14 11:55:35 (GMT-5) (138 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
```

1. Skaner zidentyfikował serwer jako cloudflare i nie wykryto nagłówków zdradzających wersje serwera backendowego.
2. Skaner zaraportował brak nagłówków na porcie 80, lecz wynika to z automatycznego skierowania na szyfrowany protokołów HTTPS gdzie mechanizmy są aktywne.
3. Skaner potwierdza poprzez wykrycie ścieżki /cdn-cgi/trace , że ruch sieciowy jest procesowany przez Reverse proxy i zapewnia ochrone przed atakami DDoS(WAF działa)

Test brute-force



Po pięciu próbach logowania użytkownik zostaje zablokowany na 5 minut i strona nie pozwala mu się dalej zalogować

Stress-test

```
┌──(cali㉿kali)-[~]
└─$ ab -n 100 -c 10 https://factshub.pl/
This is ApacheBench, Version 2.3 <$Revision: 1923142 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking factshub.pl (be patient).....done


Server Software:        cloudflare
Server Hostname:        factshub.pl
Server Port:            443
SSL/TLS Protocol:       TLSv1.3,TLS_AES_256_GCM_SHA384,256,256
TLS Server Name:        factshub.pl

Document Path:          /
Document Length:        20392 bytes

Concurrency Level:      10
Time taken for tests:   3.037 seconds
Complete requests:      100
Failed requests:        80
   (Connect: 0, Receive: 0, Length: 80, Exceptions: 0)
Total transferred:      2125839 bytes
HTML transferred:       2054909 bytes
Requests per second:    32.93 [#/sec] (mean)
Time per request:       303.658 [ms] (mean)
Time per request:       30.366 [ms] (mean, across all concurrent requests)
Transfer rate:          683.67 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       23   28   2.2     28      34
Processing:   194  242  83.5    206     461
Waiting:      193  224  45.3    205     353
Total:        218  271  84.3    235     491

Percentage of the requests served within a certain time (ms)
  50%    235
  66%    242
  75%    250
  80%    255
  90%    457
  95%    466
  98%    478
  99%    491
 100%    491 (longest request)
```

Przy pomocy narzędzia ab wysłano 100 żądań przez 10 symulowanych użytkowników. Serwer odrzucił 80 ze 100 żądań, oznacza to, że Rate Limiter działa poprawnie i broni serwer przed przeciążeniem.

Test ze strony sslabs (SSL Server Test)

SSL Report: factshub.pl

Assessed on: Sun, 14 Dec 2025 18:05:04 UTC | Hide | Clear cache

Scan Another >>

	Server	Test time	Grade
1	2606:4700:3035:0:0:0:ac43:98ab Ready	Sun, 14 Dec 2025 18:00:50 UTC Duration: 64.127 sec	B
2	172.67.152.171 Ready	Sun, 14 Dec 2025 18:01:54 UTC Duration: 63.564 sec	B
3	2606:4700:3035:0:0:0:6815:5228 Ready	Sun, 14 Dec 2025 18:02:58 UTC Duration: 63.189 sec	B
4	104.21.82.40 Ready	Sun, 14 Dec 2025 18:04:01 UTC Duration: 62.848 sec	B

SSL Report v2.4.1

Przeprowadzono weryfikację poprawności konfiguracji warstwy szyfrującej (SSL/TLS) przy użyciu narzędzia *Qualys SSL Labs*. Test weryfikował poprawność certyfikatu oraz siłę obsługiwanych protokołów. Uzyskano ocenę B, co oznacza, że szyfrowanie działa i jest bezpieczne, lecz nie jest idealne według najnowszych standardów.

Prezentacja - 12.01.2026

Link do prezentacji:
https://docs.google.com/presentation/d/1XZXuYQiIKqtDusJNI55AYRl45GhCQgdnoKTnlwJy6bw/edit?usp=sharing