# Zagadnienia bezpieczeństwa

* 1. Dostęp do danych (baza danych, uwierzytelnienie itd)

Dostęp do danych znajdujących się w bazach danych systemu powinni otrzymać użytkownicy do nich uprawnieni, w związku z czym, w systemie jest stosowana identyfikacja użytkowników. Użytkownicy mogą uzyskać dostęp do swoich danych osobowych tylko i wyłącznie poprzez uwierzytelnienie w serwisie (login i hasło).

Aby zminimalizować ryzyko naruszenia ochrony danych osobowych dostęp do nich zostanie przydzielony jak najmniejszej liczbie członków zespołu. Wszystkie próby odczytu/modyfikacji danych wrażliwych muszą być jasno udokumentowane. Bazy danych zostaną umieszczone w kontenerach, więc port usługi bazy danych nie jest dostępny z sieci publicznej, co zapewnia nam większe bezpieczeństwo.

Usługi internetowe RESTful muszą być zabezpieczone przed wyciekami danych uwierzytelniających. Hasła, tokeny bezpieczeństwa i klucze API nie powinny pojawiać się w adresie URL, ponieważ może to prowadzić do wycieku poufnych informacji.

## JWT

JWT – Json Web Token jest otwartym standardem (RFC 7519) tworzenia tokenów dostępowych, które zawierają w sobie pewną porcję danych (payload). Siłą tokenów jest to, że mogą przenosić dowolną porcję danych i informacji o użytkowniku. Token składa się z trzech części:

Nagłówka – określa jaki algorytm szyfrowania został wykorzystany do utworzenia podpisu tokena.

Payloadu – tu znajdują się dekodowane w base64 uprawnienia/informacje przenoszone w tokenie.

Podpisu cyfrowego – podpis payloadu token-a. Podpisany kluczem prywatnym serwera (dostawcy token-a). Zabezpiecza token przed modyfikacją.

Należy zaznaczyć iż domyślnie tokeny JWT nie posiadają funkcji pozwalających na wykluczenie przechwycenia tokena. Często w tokenach umieszcza się takie informacje jak ip, rodzaj przeglądarki (user-agent). Tokeny mają różne czasy życia, w przypadku aplikacji o wysokich wymaganiach bezpieczeństwa tokeny z pozwoleniami np. przelewu mogą być jednorazowe.

Uwaga! Użycie tokenów nie eliminuje konieczności użycia rozwiązań pozwalających zachować poufność przesyłanych informacji. Nie należy rozumieć tokenów JWT jako rozwiązań tej klasy co np. SSL.

Token przesyłany jest w nagłówku „Authorization” zapytania HTTP.

Rolą JWT w aplikacji Pubweb jest wykluczenie konieczności każdorazowego logowania loginem i hasłem oraz możliwość przekazywania informacji pomiędzy serwisami takich jak role i przywileje w spójnej formie. Użycie tokenów JWT pozwala również na ograniczenie konieczności odpytywania punktu autoryzacyjnego o dane użytkownika za każdym razem.

Standardowy schemat autoryzacji przy pomocy tokena JWT wygląda następująco:

1. Użytkownik wysyła hasło i login do serwera autoryzacyjnego

2. Serwer autoryzacyjny zwraca token JWT wraz z informacjami o tym jakie uprawnienia posiada użytkownik w aplikacji, do podpisu JWT użyty jest klucz prywatny serwera autoryzacyjnego

3. Użytkownik używa tokenu wygenerowanego przez centrum autoryzacji do wykonania operacji, w którymś z mikroserwisów.

4. Mikroserwis udziela lub nie udziela zgody na wykonanie operacji na podstawie:

4.1. Sprawdzenia za pomocą klucza publicznego serwera autoryzacji autentyczności JWT (porównanie podpisów).

4.2. W przypadku, gdy weryfikacja z 4.1. przebiegła pomyślnie serwis sprawdza czy w JWT zawarte są uprawnienia (np. Role/przywileje) wymagane do wykonania danej operacji

4.3. Mikroserwis wykonuje inne walidacje biznesowe charakterystyczne dla danej operacji

Powyższy schemat ma poniższe implikacje:

- Jeżeli dojdzie do rozszerzenia uprawnień użytkownika to po stronie użytkownika leży wygenerowanie i używanie nowego tokena o adekwatnych uprawnieniach

- Jeżeli dojdzie do ograniczenia uprawnień użytkownika to z punktu widzenia autoryzacji za pomocą JWT wejdzie ono w życie dopiero po wygaśnięciu poprzedniego tokena. Dla operacji o wysokim stopniu bezpieczeństwa potrzebny jest dodatkowy mechanizm autoryzacji (np. każdorazowe odpytywanie serwera autoryzacyjnego o uprawnienia użytkownika, którego dotyczy JWT).

## Przesyłanie danych ( https, szyfrowanie)

Komunikacja z serwerem odbywa się poprzez zastosowanie protokołu HTTPS. HTTPS zapewnia szyfrowanie asymetrycznie między klientem a serwerem, co pozwoli to na uzyskanie żądanej poufności oraz integralności transmisji danych.

Bezpieczne usługi REST powinny jedynie udostępniać punkty końcowe HTTPS. Pozwala to na ochronę poświadczeń uwierzytelniających podczas przesyłania, takich jak na przykład hasła, klucze API lub tokeny sieciowe JSON. Umożliwia to również klientom uwierzytelnianie usług oraz gwarantuje integralność przesyłanych danych.

Aby chronić przesyłane dane, należy stosować tzw. “dobre praktyki” TLS/SSL, takie jak zweryfikowane certyfikaty, odpowiednio chronione klucze prywatne, stosowanie wyłącznie bezpiecznych szyfrów. Dane prywatne muszą być szyfrowane w pamięci przy użyciu kluczy o odpowiedniej długości i w ściśle określonych warunkach dostępu, zarówno technicznych, jak i proceduralnych. Dane uwierzytelniające użytkownika muszą być hashowane niezależnie od tego, czy są one szyfrowane czy nie.

## Serwery

Dostęp do serwerów jest możliwy poprzez protokół komunikacyjny SSH. Zalogowanie do SSH jest możliwe tylko poprzez zastosowanie SSH Key. Logowanie przy pomocy hasła może zostać złamane atakiem brute force, a rozszyfrowanie SSH Key jest aktualnie niemal niemożliwe.

Całe oprogramowanie serwerów jest aktualizowane na bieżąco w celu zmniejszenia liczby podatności.

Serwery są odpowiednio skonfigurowane pod względem bezpieczeństwa poprzez zastosowanie zapór sieciowych (firewalls).

## Przechowywanie haseł

Hasła w systemie przechowywane są tylko i wyłącznie w postaci ich skrótów wykorzystując bezpieczną funkcję skrótu.

Informacje niejawne dotyczące użytkowników, takie jak hasła, muszą być również chronione przy użyciu silnych, odpornych na kolizje funkcji skrótów, w celu znacznego ograniczenia ryzyka ujawnienia danych uwierzytelniających, jak również zapewnienia właściwej kontroli integralności.

## Kodowanie wyjść

Należy zapewnić, poprzez usługi internetowe, takie kodowanie informacji wysyłanych do klientów aby mogły być one wykorzystywane jako dane, a nie jako skrypty. Jest to szczególnie istotne, gdy klienci usług internetowych wykorzystują dane wyjściowe do renderowania stron HTML bezpośrednio lub pośrednio przy użyciu obiektów AJAX.

## Rozmiar oraz przepustowość komunikatów

Aplikacje internetowe, mogą być obiektem ataków DOS wykonywanych poprzez automatyczne wysyłanie do serwisów internetowych tysięcy obszernych komunikatów z protokołu SOAP. Może to doprowadzić do sparaliżowania atakowanej aplikacji, uniemożliwiając jej odpowiadanie na prawdziwe wiadomości. Aby zapobiec takiej sytuacji, rozmiar komunikatów SOAP powinien być ograniczony. Większy rozmiar (lub brak limitu w ogóle) zwiększa szanse na udany atak DoS.

Przepustowość komunikatów oznacza liczbę zapytań serwisów internetowych obsługiwanych w zadanym czasie. Należy dokonać konfiguracji zoptymalizowanej pod kątem maksymalnej przepustowości wiadomości, co pozwoli na uniknięcie sytuacji podobnych do DoS.

## Prepared query

W celu zapobiegania atakowi SQL injection zostanie wykorzystany mechanizm prepared query opierający się na wysyłaniu zapytania oraz danych do serwera bazy danych oddzielnie.

# Zagadnienia niezawodności

## Odporność na awarie

Systemy rozproszone charakteryzują się wyższą odpornością na uszkodzenia, ponieważ zasoby mogą być zwielokrotnione oraz awarie zazwyczaj są częściowe. Awarie częściowe nie powodują zatrzymania całego systemu tylko pojedynczej składowej. Odporność na awarie zostanie zapewniona przez m.in. automatyczne restartowanie kontenerów Docker. Funkcja autorestart zapewnia automatyczne resetowanie kontenerów w przypadku ich zatrzymania bądź awarii. Zapewni nam to przywrócenie stanu części systemu sprzed awarii.

## Replikacja danych

Awaria niektórych systemów może wpływać na dostępność danych. Oprócz zastosowania auto restartowania kontenerów zastosowana również zostanie replikacja danych, czyli przechowywanie danych w kilku kopiach. Replikacja danych jest realizowana poprzez kopiowanie i przesyłanie danych między serwerami, i ich synchronizacji w celu zapewnienia spójności. Pozwala na skrócenie czasu dostępu do danych, oraz uniezależnienie się od czasowej niedostępności serwerów. Wadą jest konieczność aktualizowania repliki w przypadku zmiany danych źródłowych.

W systemie zastosowana jest baza danych PostgreSQL, która oferuję łatwą konfigurację replikacji danych. PostgreSQL zapewnia mechanizmy niezawodności opisane w https://www.postgresql.org/docs/9.3/wal-reliability.html.

## Usterki

Niezawodny system powinien posiadać mechanizm diagnozowania usterek i podejmowania odpowiednich działań. Działania te będą obejmować maskowanie usterek za pomocą całkowitego ukrycia i kontynuowania dalszej pracy; chwilowe lub całkowite przerwanie funkcjonowania w przypadku, gdy maskowanie jest niemożliwe do wykonania. Całkowite przerwanie jest wykonywane w ściśle określony sposób pozwalający na uniknięcie utraty lub spójności danych.