# WhiteListChecker wersja 2.0.0

**Aplikacja do weryfikacji białej listy podatników VAT**

## Spis treści

# 1. Ogólne informacje

WhiteListChecker jest aplikacją webową (WebAPI) służącą do weryfikacji podmiotów na białej liście podatników VAT prowadzonej przez Ministerstwo Finansów (Wykaz podatników VAT). Aplikacja udostępnia następujące funkcjonalności:

- automatyczne pobieranie dziennego pliku danych z serwera Ministerstwa Finansów (plikplaski.mf.gov.pl)
- weryfikacja numeru NIP i numeru konta bankowego na białej liście podatników VAT
- weryfikacja z użyciem algorytmu SHA-512 (zgodnie ze specyfikacją Ministerstwa Finansów)
- obsługa masek numerów kont bankowych (rachunki wirtualne)
- ręczne pobieranie pliku danych dla wskazanej daty
- automatyczne czyszczenie starych danych (konfigurowalny okres retencji)
- powiadomienia o statusie pobierania plików (Email, Microsoft Teams)
- interfejs Swagger do testowania i dokumentacji API

**Kluczowe informacje:**

- Aplikacja nasłuchuje na wewnętrznym porcie **8080**
- Domyślne mapowanie zewnętrzne: **5000**
- Interfejs Swagger: `/swagger`
- Endpoint wersji / health check: `/api/version`
- Źródło danych: `https://plikplaski.mf.gov.pl/pliki/{yyyyMMdd}.7z`

# 2. Wymagania

## 2.1. Wymagania sprzętowe

Do prawidłowego działania aplikacji wymagany jest następujący sprzęt:

- procesor 2-rdzeniowy
- 4GB pamięci operacyjnej (plik danych białej listy jest dużych rozmiarów - po rozpakowaniu zajmuje ponad 1GB)

- 2GB wolnej przestrzeni dyskowej (na pliki danych i archiwa dla kilku dni)

## 2.2. Wymagania oprogramowania

Aplikacja do prawidłowego działania wymaga:

- Docker Engine 20.10+ lub Podman 4.0+ (rekomendowane)
- Docker Compose / Podman Compose (opcjonalnie, rekomendowane)

Alternatywnie, do uruchomienia bez konteneryzacji:

- ASP.NET Core Runtime w wersji 10.0

## 2.3. Wymagania dodatkowe

Aplikacja do prawidłowego działania wymaga dostępu do zewnętrznego serwisu Ministerstwa Finansów:

- **https://plikplaski.mf.gov.pl** - źródło dziennego pliku danych białej listy podatników VAT (plik płaski w formacie 7z)

### Sprawdzenie dostępności adresów

Aby upewnić się, że serwer Ministerstwa Finansów jest dostępny z maszyny, na której zainstalowana jest aplikacja, można wykonać następujące testy:

**Windows (PowerShell):**

```
Test-NetConnection plikplaski.mf.gov.pl -Port 443
```

**Windows (CMD):**

```
curl -f -s -o nul -w "%{http_code}" https://plikplaski.mf.gov.pl
```

**Linux:**

```
curl -f -s -o /dev/null -w "%{http_code}\n" https://plikplaski.mf.gov.pl
```

W przypadku poprawnej dostępności:

- Windows PowerShell: wyświetli `TcpTestSucceeded : True`
- Windows CMD: wyświetli kod statusu HTTP (np. `200`, `301` lub `302`)
- Linux: wyświetli kod statusu HTTP (np. `200`, `301` lub `302`)

Jeśli serwer jest niedostępny, zostanie wyświetlony komunikat o błędzie połączenia.

---

# 3. Instalacja

## 3.1. Instalacja na systemach Linux

### Instalacja z Docker (rekomendowana)

**Wymagania:**

- Docker Engine 20.10+
- Docker Compose (opcjonalnie)

**Kroki instalacji:**

1. Zainstaluj Docker:

```
sudo apt-get install -y docker.io docker-compose
sudo systemctl start docker
sudo systemctl enable docker
sudo usermod -aG docker $USER
# WAŻNE: Wyloguj się i zaloguj ponownie aby zmiany grupy zadziałały
```

2. Utwórz katalog aplikacji:

```
mkdir -p ~/WhiteListChecker
```

```
cd ~/WhiteListChecker
```

3. Utwórz plik `docker-compose.yml`:

```
services:
  whitelistchecker-api:
    image: ghcr.io/ggiewon/whitelistchecker-webapi:develop
    container_name: whitelistchecker-api
    ports:
      - "5000:8080"
    volumes:
      - ./DataFiles:/app/DataFiles
      - ./Downloads:/app/Downloads
      - ./logs:/app/logs
    environment:
      - ASPNETCORE_ENVIRONMENT=Production
      - ASPNETCORE_URLS=http://+:8080
    restart: unless-stopped
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:8080/api/version"]
      interval: 30s
      timeout: 3s
      retries: 3
      start_period: 10s
    networks:
      - whitelistchecker-network

networks:
  whitelistchecker-network:
    driver: bridge
```

4. Utwórz katalogi danych:

```
mkdir -p DataFiles Downloads logs
chmod 755 DataFiles Downloads logs
```

5. Sprawdź strukturę katalogów:

```
ls -la
```

Oczekiwana struktura:

```
WhiteListChecker/
├── docker-compose.yml
├── DataFiles/
├── Downloads/
└── logs/
```

6. Zaloguj się do GitHub Container Registry i uruchom:

```
# Autentykacja (wymagana do pobrania obrazu)
echo "YOUR_GITHUB_TOKEN" | docker login ghcr.io -u YOUR_GITHUB_USERNAME --p

# Pobranie obrazu
docker-compose pull

# Uruchomienie
docker-compose up -d
```

7. Sprawdź działanie:

```
# Status kontenera
docker-compose ps

# Logi aplikacji
docker-compose logs -f whitelistchecker-api

# Test endpoint wersji
curl http://localhost:5000/api/version
```

**W przeglądarce:**

- Swagger UI: http://localhost:5000/swagger

Dokładna instrukcja instalacji z użyciem kontenerów (Docker i Podman, Linux i Windows) dostępna w pliku WhiteListChecker_Installation_Guide.pdf.

### Instalacja z Podman

```
# Instalacja Podman
sudo apt-get install -y podman
pip3 install podman-compose

# Uruchomienie (ta sama konfiguracja docker-compose.yml)
echo "YOUR_GITHUB_TOKEN" | podman login ghcr.io -u YOUR_GITHUB_USERNAME --p
podman-compose pull
podman-compose up -d
```

## 3.2. Instalacja na systemach Windows

### Instalacja z Docker Desktop

1. Zainstaluj Docker Desktop ze strony: https://www.docker.com/products/docker-desktop
2. Utwórz katalog `C:\WhiteListChecker`
3. Utwórz plik `docker-compose.yml` (taka sama zawartość jak w sekcji 3.1)
4. Utwórz katalogi: `DataFiles`, `Downloads`, `logs`:

```
New-Item -Path "C:\WhiteListChecker" -ItemType Directory
Set-Location -Path "C:\WhiteListChecker"
New-Item -Path ".\DataFiles" -ItemType Directory
New-Item -Path ".\Downloads" -ItemType Directory
New-Item -Path ".\logs" -ItemType Directory
```

5. Zaloguj się do GitHub Container Registry:

```
$env:GITHUB_TOKEN = "YOUR_GITHUB_TOKEN"
$env:GITHUB_TOKEN | docker login ghcr.io -u YOUR_GITHUB_USERNAME --password
Remove-Item Env:\GITHUB_TOKEN
```

6. Pobierz i uruchom:

```
docker-compose pull
docker-compose up -d
```

7. Sprawdź działanie:

```
# Status kontenera
docker-compose ps

# Test endpoint wersji
Invoke-WebRequest -Uri http://localhost:5000/api/version
```

**W przeglądarce:**

- Swagger UI: http://localhost:5000/swagger

Aby przetestować czy aplikacja została zainstalowana prawidłowo, można wywołać punkt końcowy **Version** (patrz sekcja 5.4).

> **UWAGA:** Alternatywnie można użyć **Podman Desktop** (darmowy, open source) zamiast Docker Desktop. Szczegóły w WhiteListChecker_Installation_Guide.pdf.

---

# 4. Konfiguracja

## 4.1. Pliki konfiguracyjne

Aplikacja korzysta z dwóch plików konfiguracyjnych:

1. `appsettings.json` - główny plik konfiguracyjny (wbudowany w obraz Docker)
2. `appsettings.additional.json` - opcjonalny plik nadpisujący ustawienia (może być zamontowany jako wolumen Docker)

Kolejność ładowania konfiguracji (od najniższego do najwyższego priorytetu):

1. `appsettings.json`

2. `appsettings.additional.json` (opcjonalny)
3. Zmienne środowiskowe

Konfigurację można również nadpisać za pomocą **zmiennych środowiskowych**.

## 4.2. Parametry konfiguracji

Poniżej przedstawiono pełną strukturę pliku `appsettings.json`:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Trace",
      "Microsoft": "Information"
    }
  },
  "AllowedHosts": "*",
  "HttpClient": {
    "TimeoutMinutes": 30
  },
  "Download": {
    "AlertAfterHour": 10
  },
  "Maintenance": {
    "RetentionDays": 4
  },
  "Notifications": {
    "Email": {
      "Enabled": false,
      "SmtpHost": "",
      "SmtpPort": 587,
      "UseSsl": true,
      "Username": "",
      "Password": "",
      "FromAddress": "",
      "FromName": "WhiteListChecker",
      "Recipients": []
    },
    "Teams": {
      "Enabled": false,
      "WebhookUrl": ""
    }
```

```
    }
  }
```

**Opis poszczególnych sekcji:**

### Sekcja `HttpClient`

Konfiguracja klienta HTTP używanego do pobierania plików danych z serwera Ministerstwa Finansów.

| Parametr | Opis | Domyślnie |
|---|---|---|
| `TimeoutMinutes` | Timeout żądania HTTP w minutach. Pliki danych są duże (>100MB), dlatego domyślna wartość jest ustawiona na 30 minut | `30` |

> **UWAGA:** W przypadku wolnego połączenia internetowego lub dużych plików danych, może być konieczne zwiększenie tej wartości.

### Sekcja `Download`

Konfiguracja automatycznego pobierania plików danych przez `DownloadService`.

| Parametr | Opis | Domyślnie |
|---|---|---|
| `AlertAfterHour` | Godzina (0-23), po której wysyłane jest powiadomienie o błędzie, jeśli plik danych dla bieżącego dnia nie został jeszcze pobrany. Ministerstwo Finansów zwykle publikuje pliki rano, dlatego domyślna wartość to 10 | `10` |

### Sekcja `Maintenance`

Konfiguracja automatycznego czyszczenia starych plików danych przez `MaintenanceService`.

| Parametr | Opis | Domyślnie |
|---|---|---|
| `RetentionDays` | Liczba dni przechowywania plików danych. Katalogi starsze niż ta wartość są automatycznie | `4` |

| | | |
|---|---|---|
| | usuwane. Wartość 4 oznacza, że przechowywane są dane z ostatnich 4 dni | |

> **UWAGA:** Każdy plik danych po rozpakowaniu zajmuje ponad 1GB. Przy domyślnym ustawieniu 4 dni, potrzebne jest minimum ~5GB przestrzeni dyskowej.

#### Sekcja `Logging`

Konfiguracja poziomów logowania. Aplikacja wykorzystuje **NLog** do zapisywania logów. Pliki logów są zapisywane w katalogu `logs/`.

| Parametr | Opis | Domyślnie |
|---|---|---|
| `LogLevel:Default` | Domyślny poziom logowania | `Trace` |
| `LogLevel:Microsoft` | Poziom logowania dla komponentów Microsoft | `Information` |

Dostępne poziomy logowania (od najmniej do najbardziej szczegółowego): `None`, `Critical`, `Error`, `Warning`, `Information`, `Debug`, `Trace`.

## 4.3. Konfiguracja powiadomień Email

Sekcja `Notifications.Email` odpowiada za konfigurację powiadomień wysyłanych pocztą elektroniczną. Powiadomienia są wysyłane przy pomyślnym pobraniu pliku danych oraz w przypadku błędu pobierania.

```
"Notifications": {
  "Email": {
    "Enabled": true,
    "SmtpHost": "smtp.example.com",
    "SmtpPort": 587,
    "UseSsl": true,
    "Username": "user@example.com",
    "Password": "haslo",
    "FromAddress": "whitelistchecker@example.com",
    "FromName": "WhiteListChecker",
    "Recipients": ["admin@example.com", "operator@example.com"]
  }
}
```

| Parametr | Opis | Typ | Domyślnie |
|---|---|---|---|
| `Enabled` | Włączenie/wyłączenie powiadomień email | boolean | `false` |
| `SmtpHost` | Nazwa hosta serwera SMTP | string | `""` |
| `SmtpPort` | Port serwera SMTP | integer | `587` |
| `UseSsl` | Użycie SSL/TLS dla połączenia SMTP | boolean | `true` |
| `Username` | Nazwa użytkownika do autentykacji SMTP | string | `""` |
| `Password` | Hasło do autentykacji SMTP | string | `""` |
| `FromAddress` | Adres email nadawcy | string | `""` |
| `FromName` | Nazwa wyświetlana nadawcy w nagłówku wiadomości | string | `"WhiteListChecker"` |
| `Recipients` | Lista adresów email odbiorców powiadomień | string[] | `[]` |

**Typowe konfiguracje SMTP:**

| Serwer | Host | Port | SSL |
|---|---|---|---|
| Microsoft 365 | `smtp.office365.com` | `587` | `true` |
| Gmail | `smtp.gmail.com` | `587` | `true` |
| Exchange on-premise | `mail.firma.pl` | `25` lub `587` | zależy od konfiguracji |

> **UWAGA:** W przypadku Gmail wymagane jest użycie hasła aplikacji (App Password) zamiast standardowego hasła konta.

## 4.4. Konfiguracja powiadomień Teams

Sekcja `Notifications.Teams` odpowiada za konfigurację powiadomień wysyłanych do Microsoft Teams za pomocą Incoming Webhook. Powiadomienia są wyświetlane jako

karty adaptacyjne (Adaptive Cards) z informacją o nazwie pliku, dacie i statusie.

```
"Notifications": {
  "Teams": {
    "Enabled": true,
    "WebhookUrl": "https://outlook.office.com/webhook/..."
  }
}
```

| Parametr | Opis | Typ | Domyślnie |
|---|---|---|---|
| `Enabled` | Włączenie/wyłączenie powiadomień Teams | boolean | `false` |
| `WebhookUrl` | URL webhooka Microsoft Teams Incoming Webhook | string | `""` |

**Tworzenie webhooka w Microsoft Teams**

1. Otwórz kanał w Microsoft Teams
2. Kliknij ikonę "..." → "**Connectors**" (lub "**Łączniki**")
3. Znajdź "**Incoming Webhook**" i kliknij "**Configure**" (lub "**Konfiguruj**")
4. Nadaj nazwę (np. `WhiteListChecker`) i opcjonalnie wgraj ikonę
5. Kliknij "**Create**" (lub "**Utwórz**")
6. Skopiuj wygenerowany URL webhooka
7. Wklej URL do konfiguracji `WebhookUrl`

## 4.5. Nadpisywanie konfiguracji

Istnieją trzy sposoby nadpisywania konfiguracji domyślnej:

### Sposób 1: Plik `appsettings.additional.json`

Utwórz plik `appsettings.additional.json` i zamontuj go jako wolumen Docker. Plik nadpisuje tylko te wartości, które są w nim zdefiniowane - pozostałe ustawienia pozostają domyślne.

**Przykład pliku:**

```
{
  "HttpClient": {
    "TimeoutMinutes": 60
  },
  "Download": {
    "AlertAfterHour": 12
  },
  "Maintenance": {
    "RetentionDays": 7
  },
  "Notifications": {
    "Email": {
      "Enabled": true,
      "SmtpHost": "smtp.office365.com",
      "SmtpPort": 587,
      "UseSsl": true,
      "Username": "user@firma.pl",
      "Password": "haslo123",
      "FromAddress": "whitelistchecker@firma.pl",
      "FromName": "WhiteListChecker",
      "Recipients": ["admin@firma.pl"]
    },
    "Teams": {
      "Enabled": true,
      "WebhookUrl": "https://outlook.office.com/webhook/..."
    }
  }
}
```

### Montowanie jako wolumen Docker:

```
volumes:
  - ./appsettings.additional.json:/app/appsettings.additional.json:ro
  - ./DataFiles:/app/DataFiles
  - ./Downloads:/app/Downloads
  - ./logs:/app/logs
```

Po zmianie pliku należy zrestartować kontener:

```
docker-compose restart
```

### Sposób 2: Zmienne środowiskowe

Użyj podwójnego podkreślenia ( `__` ) jako separatora dla zagnieżdżonych kluczy w zmiennych środowiskowych:

```
environment:
  - ASPNETCORE_ENVIRONMENT=Production
  - ASPNETCORE_URLS=http://+:8080
  - HttpClient__TimeoutMinutes=60
  - Download__AlertAfterHour=12
  - Maintenance__RetentionDays=7
  - Notifications__Email__Enabled=true
  - Notifications__Email__SmtpHost=smtp.office365.com
  - Notifications__Email__SmtpPort=587
  - Notifications__Email__Username=user@firma.pl
  - Notifications__Email__Password=haslo123
  - Notifications__Email__FromAddress=whitelistchecker@firma.pl
  - Notifications__Email__Recipients__0=admin@firma.pl
  - Notifications__Email__Recipients__1=operator@firma.pl
  - Notifications__Teams__Enabled=true
  - Notifications__Teams__WebhookUrl=https://outlook.office.com/webhook/...
```

### Sposób 3: Łączenie obu metod

Zmienne środowiskowe mają najwyższy priorytet, następnie `appsettings.additional.json`, a na końcu `appsettings.json`. Można łączyć obie metody.

---

# 5. Użytkowanie

Aplikacja umożliwia dostęp do wszystkich funkcji poprzez interfejs **Swagger**, gdzie wszystkie dostępne funkcje można wywołać w dowolnej przeglądarce internetowej.

## Dostęp do interfejsu Swagger

**Serwer lokalny:**

- Format adresu: `http://<adresIPSerwera>:<port>/swagger`
- Gdzie:
  - `adresIPSerwera` - IP lub nazwa komputera gdzie zainstalowana jest aplikacja
  - `port` - port na którym dostępna jest aplikacja (domyślnie 80 lub 8080 dla Docker)

**Docker:**

- `http://localhost:5000/swagger`

> **UWAGA:** Opis działania poszczególnych funkcji wraz z parametrami dla aktualnie zainstalowanej wersji dostępny jest poprzez interfejs Swagger i może się różnić od informacji zawartych w tej instrukcji (instrukcja zawiera zawsze opis najnowszej wersji aplikacji).

---

## 5.1. Check

**Ścieżka bazowa:** `/api/check`

Kontroler odpowiedzialny za weryfikację numeru NIP i konta bankowego na białej liście podatników VAT.

### 5.1.1. Weryfikacja NIP i konta bankowego

**Endpoint:** `POST /api/check`

**Opis:** Weryfikuje jedno lub więcej par NIP+konto bankowe na białej liście podatników VAT dla wskazanej daty. Metoda ładuje plik danych dla podanej daty, oblicza hash SHA-512 dla każdej pary NIP+konto (zgodnie ze specyfikacją MF), a następnie porównuje go z listą skrótów podatników czynnych i zwolnionych.

**Content-Type:** `multipart/form-data`

**Parametry (form data):**

- `Date` (string, wymagany) - Data dla której sprawdzana jest biała lista, format: `yyyyMMdd` (np. `20260208`)
- `NipAccount` (array of string, wymagany) - Lista par NIP+konto do sprawdzenia. Każdy element ma format: `NIP|NumerKontaBankowego`

**Format NipAccount:**

Format każdego elementu tablicy: `NNNNNNNNNN|PPKKBBBBBBBBCCCCCCCCCCCC`

Gdzie:

- `NNNNNNNNNN` - 10-cyfrowy numer NIP
- `|` - separator
- `PP` - kod kraju (np. `PL`)
- `KK` - cyfry kontrolne IBAN
- `BBBBBBBB` - numer rozliczeniowy banku (8 cyfr)
- `CCCCCCCCCCCC` - numer rachunku klienta

**Przykład:** `1234567890|PL12345678901234567890123456`

**Odpowiedź:**

- Status: `200 OK`
- Format: JSON

```
{
  "checks": [
    {
      "nipAccount": "1234567890|PL12345678901234567890123456",
      "existOnList": true,
      "fileCrc": "a1b2c3d4e5f6...",
      "date": "20260208",
      "sha512": "f7e6d5c4b3a2..."
    }
  ]
}
```

**Struktura odpowiedzi:**

| Pole | Typ | Opis |
|---|---|---|
| `checks` | array | Tablica wyników weryfikacji |

**Struktura `SingleCheckEntry` (element tablicy `checks`):**

| Pole | Typ | Opis |
|---|---|---|
| `nipAccount` | string | Sprawdzana para NIP+konto (echo parametru wejściowego) |
| `existOnList` | boolean | `true` - para NIP+konto znajduje się na białej liście podatników VAT (czynnych lub zwolnionych); `false` - para nie została znaleziona na liście |
| `fileCrc` | string | Suma kontrolna SHA pliku danych użytego do weryfikacji. Służy do potwierdzenia integralności pliku |
| `date` | string | Data pliku danych użytego do weryfikacji (format `yyyyMMdd`) |
| `sha512` | string | Obliczony hash SHA-512 dla podanej pary NIP+konto. Hash jest obliczany zgodnie ze specyfikacją MF (wielokrotna transformacja SHA-512) |

**Mechanizm działania:**

1. Aplikacja ładuje plik danych białej listy dla podanej daty
2. Dla każdej pary NIP+konto: a. Łączy datę i parę NIP+konto w jeden ciąg: `{date}{nipAccount}` b. Oblicza wielokrotny hash SHA-512 (liczba transformacji określona w pliku danych) c. Porównuje obliczony hash z listą skrótów podatników czynnych (`skrotyPodatnikowCzynnych`) d. Porównuje obliczony hash z listą skrótów podatników zwolnionych (`skrotyPodatnikowZwolnionych`) e. Jeśli nie znaleziono - próbuje z maskami kont bankowych (obsługa rachunków wirtualnych)
3. Zwraca wynik weryfikacji dla każdej pary

**Obsługa masek kont bankowych:**

Ministerstwo Finansów udostępnia w pliku danych listę masek kont bankowych. Maski służą do obsługi rachunków wirtualnych, gdzie część numeru konta jest zamaskowana znakiem `X`. Jeśli standardowe sprawdzenie nie daje wyniku pozytywnego, aplikacja automatycznie stosuje odpowiednie maski i powtarza weryfikację.

**Kody błędów:**

- `500 Internal Server Error` - Błąd serwera (np. brak pliku danych dla wskazanej daty, błąd odczytu pliku)

### Przykład wywołania (curl):

```
curl -X POST http://localhost:5000/api/check \
  -F "Date=20260208" \
  -F "NipAccount=1234567890|PL12345678901234567890123456"
```

### Przykład z wieloma parami:

```
curl -X POST http://localhost:5000/api/check \
  -F "Date=20260208" \
  -F "NipAccount=1234567890|PL12345678901234567890123456" \
  -F "NipAccount=9876543210|PL65432109876543210987654321"
```

### Windows PowerShell:

```
$body = @{
    Date = "20260208"
    NipAccount = @(
        "1234567890|PL12345678901234567890123456",
        "9876543210|PL65432109876543210987654321"
    )
}
Invoke-RestMethod -Uri "http://localhost:5000/api/check" -Method Post -Form
```

### Przykład odpowiedzi z wieloma wynikami:

```
{
  "checks": [
    {
      "nipAccount": "1234567890|PL12345678901234567890123456",
      "existOnList": true,
      "fileCrc": "a1b2c3d4e5f6789...",
      "date": "20260208",
      "sha512": "f7e6d5c4b3a29..."
```

```
      },
      {
        "nipAccount": "9876543210|PL65432109876543210987654321",
        "existOnList": false,
        "fileCrc": "a1b2c3d4e5f6789...",
        "date": "20260208",
        "sha512": "1a2b3c4d5e6f7..."
      }
    ]
  }
```

> **UWAGA:** Aby weryfikacja mogła się powieść, plik danych dla podanej daty musi być wcześniej pobrany. Pliki są pobierane automatycznie przez `DownloadService` (patrz sekcja 6.1) lub ręcznie za pomocą endpointu Download (patrz sekcja 5.2).

---

## 5.2. Download

**Ścieżka bazowa:** `/api/download`

Kontroler odpowiedzialny za ręczne pobieranie pliku danych białej listy.

### 5.2.1. Pobierz plik danych

**Endpoint:** `GET /api/download?date={yyyyMMdd}`

**Opis:** Ręcznie inicjuje pobieranie i rozpakowywanie pliku danych białej listy z serwera Ministerstwa Finansów ( `https://plikplaski.mf.gov.pl/pliki/{date}.7z` ). Plik jest pobierany, zapisywany w katalogu `Downloads` , rozpakowywany do `DataFiles/{date}/` , a następnie archiwum jest usuwane.

**Parametry:**

- `date` (query, opcjonalny) - Data pliku do pobrania w formacie `yyyyMMdd` (np. `20260208` ). Jeśli nie podano, używana jest data bieżąca
- `force` (query, opcjonalny, boolean) - Wymuszenie ponownego pobrania nawet jeśli plik dla danej daty już istnieje. Domyślnie: `false`

**Odpowiedź:**

- Status: `200 OK`
- Format: JSON

```
{
  "requestedDate": "20260208",
  "downloadStatus": "Ok",
  "errorMessage": null
}
```

**Struktura odpowiedzi:**

| Pole | Typ | Opis |
|---|---|---|
| `requestedDate` | string | Data pliku, który został pobrany (format `yyyyMMdd`) |
| `downloadStatus` | enum | Status operacji pobierania (patrz tabela poniżej) |
| `errorMessage` | string/null | Komunikat błędu (null jeśli operacja zakończona pomyślnie) |

**Wartości `downloadStatus`:**

| Wartość | Opis |
|---|---|
| `Unknown` | Nieznany status (stan początkowy) |
| `Ok` | Plik pobrany i rozpakowany pomyślnie |
| `Skipped` | Plik dla danej daty już istnieje na dysku. Użyj parametru `force=true` aby wymusić ponowne pobranie |
| `Error` | Wystąpił błąd podczas pobierania lub rozpakowywania. Szczegóły w polu `errorMessage` |

**Przykłady wywołania:**

```
# Pobierz plik dla bieżącej daty
curl http://localhost:5000/api/download
```

```
# Pobierz plik dla konkretnej daty
curl "http://localhost:5000/api/download?date=20260208"
```

```
# Wymuś ponowne pobranie (nawet jeśli plik już istnieje)
curl "http://localhost:5000/api/download?date=20260208&force=true"
```

### Windows PowerShell:

```
# Pobierz plik dla bieżącej daty
Invoke-RestMethod -Uri "http://localhost:5000/api/download"

# Wymuś ponowne pobranie
Invoke-RestMethod -Uri "http://localhost:5000/api/download?date=20260208&fo
```

### Przykład odpowiedzi - plik już istnieje:

```
{
  "requestedDate": "20260208",
  "downloadStatus": "Skipped",
  "errorMessage": null
}
```

### Przykład odpowiedzi - błąd:

```
{
  "requestedDate": "20260208",
  "downloadStatus": "Error",
  "errorMessage": "Response status code does not indicate success: 404 (Not
}
```

> **UWAGA:** Pobieranie dużych plików może trwać kilka minut w zależności od szybkości łącza internetowego. Timeout żądania HTTP jest konfigurowalny przez

> parametr `HttpClient:TimeoutMinutes` (domyślnie 30 minut).

---

## 5.3. Clear

**Ścieżka bazowa:** `/api/clear`

Kontroler odpowiedzialny za ręczne czyszczenie starych plików danych.

### 5.3.1. Wyczyść stare dane

**Endpoint:** `GET /api/clear?date={yyyyMMdd}`

**Opis:** Usuwa katalogi z danymi starsze niż data odcięcia wyliczona na podstawie daty referencyjnej i konfiguracji `Maintenance:RetentionDays`. Endpoint służy do ręcznego czyszczenia starych plików danych. Automatyczne czyszczenie jest realizowane przez `MaintenanceService` (patrz sekcja 6.2).

**Parametry:**

- `date` (query, opcjonalny) - Data referencyjna w formacie `yyyyMMdd` (np. `20260208`). Jeśli nie podano, używana jest data bieżąca

**Mechanizm działania:**

Dla podanej daty, endpoint:

- oblicza datę odcięcia: `date - RetentionDays`
- usuwa wszystkie katalogi z datą starszą niż data odcięcia
- np. dla `date=20260208` i `RetentionDays=4`, data odcięcia to `20260204`, więc usuwane są katalogi do `20260203` włącznie

**Odpowiedź:**

- Status: `200 OK`
- Format: JSON

```
{
  "startDate": "2026-02-08T00:00:00",
  "deletedDataFolders": 3,
```

```
  "errorMessage": null
}
```

### Struktura odpowiedzi:

| Pole | Typ | Opis |
|---|---|---|
| `startDate` | DateTime | Data referencyjna (parsowana z parametru `date`) |
| `deletedDataFolders` | integer | Liczba faktycznie usuniętych katalogów z danymi |
| `errorMessage` | string/null | Komunikat błędu (null jeśli operacja zakończona pomyślnie) |

### Przykłady wywołania:

```
# Wyczyść stare dane względem bieżącej daty
curl http://localhost:5000/api/clear
```

```
# Wyczyść stare dane względem konkretnej daty
curl "http://localhost:5000/api/clear?date=20260208"
```

### Windows PowerShell:

```
Invoke-RestMethod -Uri "http://localhost:5000/api/clear"
```

### Przykład odpowiedzi:

```
{
  "startDate": "2026-02-08T00:00:00",
  "deletedDataFolders": 5,
```

```
  "errorMessage": null
}
```

---

## 5.4. Version

**Ścieżka bazowa:** `/api/version`

Kontroler odpowiedzialny za informacje o wersji aplikacji.

### 5.4.1. Pobierz wersję aplikacji

**Endpoint:** `GET /api/version`

**Opis:** Zwraca informacje o wersji aplikacji oraz dacie kompilacji. Może być używany jako **Health Check** do sprawdzania czy aplikacja działa i jest dostępna. Endpoint jest również używany przez mechanizm Docker HEALTHCHECK do monitorowania stanu kontenera.

**Parametry:** Brak

**Odpowiedź:**

- Status: `200 OK`
- Format: JSON

```
{
  "versionId": "2.0.0+<git-commit-sha>",
  "buildDate": "2026-02-08T12:30:23.705232"
}
```

**Struktura odpowiedzi:**

| Pole | Typ | Opis |
|---|---|---|
| `versionId` | string | Wersja aplikacji z `AssemblyInformationalVersion`; może zawierać metadane builda z hashem commita (np. `2.0.0+<git-commit-sha>`) |

| | | |
|---|---|---|
| `buildDate` | DateTime | Data i czas kompilacji aplikacji (format ISO 8601) |

**Przykład wywołania:**

```
GET /api/version
```

```
curl http://localhost:5000/api/version
```

**Windows PowerShell:**

```
Invoke-RestMethod -Uri "http://localhost:5000/api/version"
```

**Użycie jako Health Check:** Metoda może zostać wywołana z dowolnego klienta HTTP. Otrzymanie statusu `200 OK` oznacza, że aplikacja działa poprawnie.

**Konfiguracja Docker HEALTHCHECK:**

```
healthcheck:
  test: ["CMD", "curl", "-f", "http://localhost:8080/api/version"]
  interval: 30s
  timeout: 3s
  retries: 3
  start_period: 10s
```

**Sprawdzenie stanu zdrowia kontenera:**

```
# Docker
docker inspect --format='{{.State.Health.Status}}' whitelistchecker-api

# Podman
podman inspect --format='{{.State.Health.Status}}' whitelistchecker-api
```

# 6. Serwisy w tle

Aplikacja uruchamia automatycznie dwa serwisy działające w tle (Hosted Services) po starcie. Serwisy te działają przez cały czas życia aplikacji i nie wymagają interakcji użytkownika.

## 6.1. DownloadService

**Opis:** Serwis automatycznie pobierający dzienny plik danych białej listy podatników VAT z serwera Ministerstwa Finansów.

**Harmonogram:** Uruchamiany co **10 minut** (pierwsza próba natychmiast po starcie aplikacji).

**Źródło danych:** `https://plikplaski.mf.gov.pl/pliki/{yyyyMMdd}.7z`

**Szczegółowy opis działania:**

1. Sprawdza czy plik danych dla bieżącej daty już istnieje w katalogu `DataFiles`
2. Jeśli plik istnieje - operacja jest pomijana (bez logowania)
3. Jeśli plik nie istnieje: a. Generuje URL do pobrania: `https://plikplaski.mf.gov.pl/pliki/{yyyyMMdd}.7z` b. Pobiera plik archiwum 7z (streaming do pliku w katalogu `Downloads`) c. Rozpakowuje archiwum do katalogu `DataFiles/{yyyyMMdd}/` d. Usuwa pobrany plik archiwum z katalogu `Downloads` e. Wysyła powiadomienie o pomyślnym pobraniu (Email i/lub Teams, jeśli skonfigurowane)

**Obsługa błędów:**

- W przypadku błędu pobierania (`HttpRequestException`) lub timeout (`TaskCanceledException`) wykonywane są do **3 prób** z wykładniczym opóźnieniem:
  - Próba 1 → czekanie 2 sekundy
  - Próba 2 → czekanie 4 sekundy
  - Próba 3 → brak ponowienia
- Jeśli wszystkie 3 próby zakończą się niepowodzeniem:
  - Jeśli aktualna godzina jest >= `AlertAfterHour` (domyślnie 10:00): wysyłane jest powiadomienie o błędzie

- Powiadomienie o błędzie wysyłane jest **maksymalnie raz dziennie** (aby nie zaśmiecać skrzynki)
- Serwis kontynuuje próby pobierania co 10 minut

**Bezpieczeństwo wątków:** Serwis używa semafora (`SemaphoreSlim`) aby zapobiec równoległemu uruchamianiu wielu operacji pobierania.

**Powiązana konfiguracja:**

- `HttpClient:TimeoutMinutes` - timeout pobierania (domyślnie 30 min)
- `Download:AlertAfterHour` - godzina po której wysyłane jest powiadomienie o błędzie (domyślnie 10)

## 6.2. MaintenanceService

**Opis:** Serwis automatycznie czyszczący stare pliki danych, aby zapobiegać nadmiernemu zużyciu dysku.

**Harmonogram:** Uruchamiany co **12 godzin** (pierwsza próba natychmiast po starcie aplikacji).

**Szczegółowy opis działania:**

1. Oblicza datę odcięcia: `data bieżąca - RetentionDays` (domyślnie 4 dni)
2. Skanuje wszystkie podkatalogi w folderze `DataFiles`
3. Dla każdego podkatalogu: a. Parsuje nazwę katalogu jako datę (format `yyyyMMdd`) b. Jeśli data katalogu jest starsza niż data odcięcia - katalog jest usuwany c. Loguje informację o usuniętych katalogach
4. Błędy są logowane ale nie przerywają pracy serwisu

**Powiązana konfiguracja:**

- `Maintenance:RetentionDays` - okres retencji w dniach (domyślnie 4)

**Przykład:** Przy domyślnej konfiguracji `RetentionDays=4` i dacie bieżącej 2026-02-08:

- Zachowane zostaną katalogi: `20260208`, `20260207`, `20260206`, `20260205`
- Usunięte zostaną katalogi: `20260204` i starsze

---

# 7. Powiadomienia

Aplikacja obsługuje dwa niezależne kanały powiadomień. Oba kanały mogą działać jednocześnie.

## Email

Powiadomienia email są wysyłane jako wiadomości HTML z informacjami o zdarzeniu. Wysyłane są w następujących sytuacjach:

| Zdarzenie | Opis | Przykład treści |
|---|---|---|
| **Pobranie pliku** | Informacja o pomyślnym pobraniu nowego pliku danych | "Pobrano nowy plik danych: 20260208.7z" |
| **Błąd pobierania** | Informacja o braku pliku danych po skonfigurowanej godzinie | "Brak pliku danych 20260208.7z po godzinie 10:00. Błąd: ..." |

Powiadomienia email zawierają:

- Temat wiadomości z informacją o zdarzeniu
- Treść z nazwą pliku, datą i czasem zdarzenia
- Treść błędu (w przypadku powiadomień o błędzie)
- Stopkę z nazwą aplikacji

Konfiguracja: patrz sekcja 4.3.

## Microsoft Teams

Powiadomienia Teams są wysyłane jako karty adaptacyjne (Adaptive Cards) poprzez Incoming Webhook. Karty zawierają:

| Element | Opis |
|---|---|
| **Tytuł** | Nazwa zdarzenia |
| **Plik** | Nazwa pliku danych |
| **Data/czas** | Znacznik czasu UTC zdarzenia |
| **Komunikat** | Szczegóły zdarzenia lub komunikat błędu |

Konfiguracja: patrz sekcja 4.4.

> **UWAGA:** Powiadomienia o błędzie pobierania są wysyłane maksymalnie **raz dziennie** per kanał, niezależnie od liczby nieudanych prób. Powiadomienia o pomyślnym pobraniu są wysyłane przy każdym pobraniu.

---

# 8. Wolumeny danych

Aplikacja wykorzystuje trzy katalogi do przechowywania danych:

| Wolumen | Ścieżka w kontenerze | Opis | Rozmiar |
|---|---|---|---|
| DataFiles | `/app/DataFiles` | Rozpakowane pliki danych białej listy | ~1GB per dzień |
| Downloads | `/app/Downloads` | Tymczasowe archiwum 7z (usuwane po rozpakowaniu) | ~100MB tymczasowo |
| logs | `/app/logs` | Pliki logów aplikacji (NLog) | Zależy od poziomu logowania |

**Struktura katalogów danych:**

```
DataFiles/
├── 20260205/
│   └── (pliki danych białej listy)
├── 20260206/
│   └── (pliki danych białej listy)
├── 20260207/
│   └── (pliki danych białej listy)
└── 20260208/
    └── (pliki danych białej listy)
```

**Docker Compose - montowanie wolumenów:**

```
volumes:
  - ./DataFiles:/app/DataFiles
  - ./Downloads:/app/Downloads
  - ./logs:/app/logs
```

> **UWAGA:** Katalog `DataFiles` rośnie codziennie o ~1GB nowych plików danych. Stare pliki są automatycznie czyszczone przez `MaintenanceService` zgodnie z ustawieniem `RetentionDays` (domyślnie: 4 dni). Przy domyślnym ustawieniu potrzeba minimum ~5GB wolnej przestrzeni dyskowej.

---

# 9. Wersje aplikacji

## Historia wersji

### Wersja 2.0.0 (aktualna)

- Migracja do IHost i nowoczesnej konfiguracji .NET
- Wyodrębnienie konfiguracji do pliku `appsettings.json`
- Wsparcie dla pliku `appsettings.additional.json` (nadpisywanie konfiguracji bez zmiany obrazu Docker)
- Powiadomienia Email - konfigurowalny serwer SMTP, wielu odbiorców, wiadomości HTML
- Powiadomienia Microsoft Teams - karty adaptacyjne (Adaptive Cards) przez Incoming Webhook
- `DownloadService` - automatyczne pobieranie plików z obsługą ponawiania (retry z exponential backoff)
- `MaintenanceService` - automatyczne czyszczenie starych danych co 12 godzin
- Powiadomienia o błędach pobierania plików (raz dziennie, po skonfigurowanej godzinie)
- Optymalizacja `FrozenSet` dla wyszukiwania O(1) na liście skrótów
- Upgrade do .NET 10.0
- Konteneryzacja z Docker/Podman (obraz dostępny na GitHub Container Registry)
- Obsługa lokalizacji (komunikaty w PL i EN)
- Docker HEALTHCHECK na endpoint `/api/version`

- Uruchamianie jako non-root user w kontenerze

**Wersja 1.x**

- Pierwsza wersja produkcyjna
- Podstawowa weryfikacja NIP i konta bankowego na białej liście
- Ręczne pobieranie plików danych
- Weryfikacja SHA-512 z obsługą masek kont bankowych

---

## 10. Licencja

Aplikacja jest oprogramowaniem własnościowym. Wszelkie prawa zastrzeżone.

---

**Data ostatniej aktualizacji dokumentacji:** 2026-02-08

**Wersja dokumentacji:** 1.0

**Zgodność z wersją aplikacji:** 2.0.0+