

SPCloud — System przechowywania plików w chmurze

Dokumentacja techniczna (projekt)

Imię Nazwisko 1 Imię Nazwisko 2 Imię Nazwisko 3 Imię Nazwisko 4
000000 000000 000000 000000

14 stycznia 2026

Frontend	SvelteKit 2.x (Svelte 5), Node.js 22, TypeScript
Backend	FastAPI, Python 3.13, SQLAlchemy (async)
Baza danych	PostgreSQL 17
Object Storage	MinIO (S3-compatible)
Infrastruktura	Docker Compose, NGINX, Raspberry Pi 5 (docelowo)

Spis treści

1	Wprowadzenie	3
2	Zakres funkcjonalny	3
2.1	Funkcje użytkownika	3
2.2	Funkcje administratora	3
3	Architektura systemu	3
3.1	Diagram architektury	4
3.2	Warstwa frontendowa	4
3.3	Warstwa backendowa	5
3.4	Baza danych	5
3.5	Object Storage	5
4	Autentykacja i autoryzacja	6
4.1	Hasła i bezpieczeństwo	6
4.2	Rodzaje tokenów	6
4.3	Przykładowe payloady JWT	6
4.4	Flow rejestracji i logowania (TOTP)	6
5	Wersjonowanie plików	7
5.1	Wersjonowanie: zasady działania	7
5.2	Model danych plików	7
6	Endpointy API	8
6.1	Użytkownicy (users)	8
6.2	TOTP	8
6.3	Pliki (files)	8
6.4	Logi (admin)	9
7	Logowanie zdarzeń	9
8	Instalacja i uruchomienie	9
8.1	Wymagania	9
8.2	Konfiguracja	9
8.3	Uruchomienie	10
8.4	Porty i routing	10
8.5	Infrastruktura produkcyjna	10
9	Podsumowanie	10

1 Wprowadzenie

SPCloud to system przechowywania plików w chmurze umożliwiający użytkownikom przechowywanie plików w magazynie obiektowym, zarządzanie wersjami oraz pobieranie plików z poziomu aplikacji webowej. Projekt został zrealizowany jako zadanie akademickie.

Główne założenia systemu obejmują:

- autentykację użytkowników oraz wymuszenie 2FA (TOTP) przy logowaniu
- przechowywanie plików w MinIO (S3-compatible) w modelu *bucket per user*
- wersjonowanie plików (upload istniejącego pliku tworzy kolejną wersję)
- przywracanie wybranej wersji bez kopiowania obiektu w storage (zmiana `current_version` w bazie)
- limit przestrzeni dyskowej per użytkownik (domyślnie 100 MiB)
- możliwość oznaczania plików jako ulubione oraz pobieranie wielu plików jako ZIP
- logowanie zdarzeń (logowanie, operacje na plikach, pobranie logów)

Dokument opisuje architekturę systemu, kluczowe elementy implementacji oraz instrukcję uruchomienia.

2 Zakres funkcjonalny

2.1 Funkcje użytkownika

- rejestracja konta i logowanie (z TOTP)
- konfiguracja TOTP na podstawie tokenu konfiguracyjnego (QR code)
- upload pliku (nowy plik lub nowa wersja)
- lista plików, pobieranie pliku, pobieranie wielu plików jako ZIP
- lista wersji pliku, pobieranie konkretnej wersji
- przywracanie wersji (ustawienie jako aktualnej)
- usuwanie pliku oraz usuwanie wersji (z wyjątkiem wersji aktualnej)
- oznaczanie/odznaczanie pliku jako ulubionego
- podgląd informacji o wykorzystaniu storage

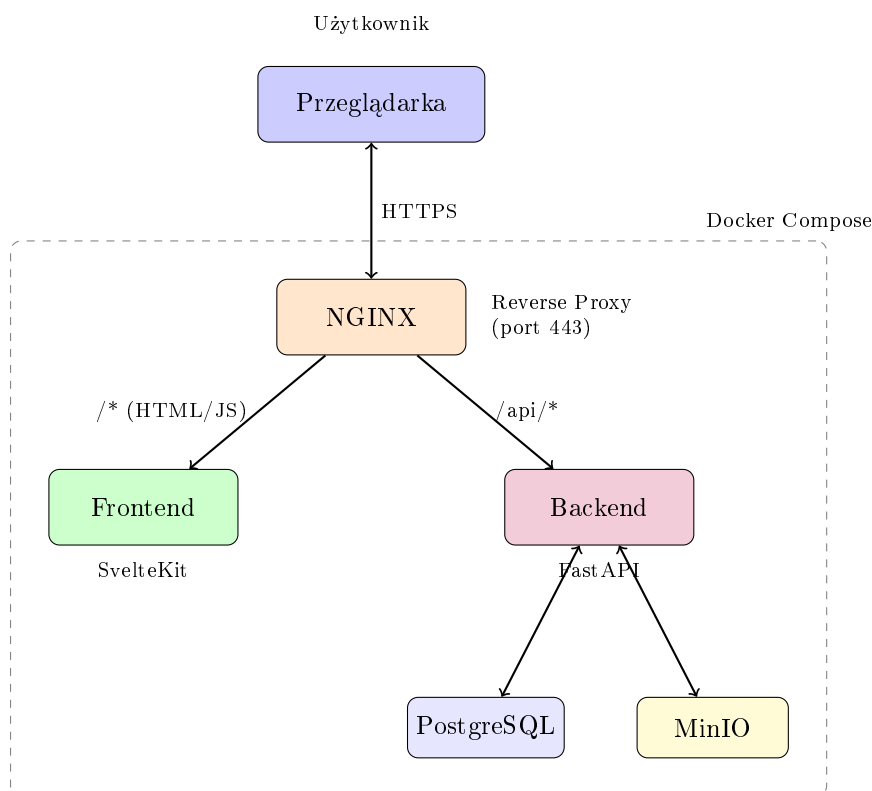
2.2 Funkcje administratora

- pobranie logów systemowych przez endpoint administracyjny

3 Architektura systemu

System jest uruchamiany kontenerowo (Docker Compose) i składa się z odseparowanych komponentów komunikujących się poprzez HTTP oraz bazę danych.

3.1 Diagram architektury



Rysunek 1: Architektura systemu SPCloud

Przepływ żądań przedstawia się następująco:

1. Użytkownik otwiera przeglądarkę i wchodzi na adres aplikacji (HTTPS)
2. NGINX (reverse proxy) kieruje żądania ścieżek **/*** do kontenera Frontend
3. Frontend (SvelteKit) zwraca statyczne pliki HTML, CSS i JavaScript do przeglądarki
4. Kod JavaScript wykonywany w przeglądarce wysyła żądania API (**/api/***) przez NGINX
5. NGINX przekazuje żądania **/api/*** do kontenera Backend (FastAPI)
6. Backend komunikuje się z PostgreSQL (metadane, użytkownicy) oraz MinIO (pliki)

Uwaga: Frontend i Backend nie komunikują się bezpośrednio między sobą. Cała komunikacja z API odbywa się z poziomu przeglądarki użytkownika przez NGINX.

3.2 Warstwa frontendowa

Frontend został zbudowany z wykorzystaniem SvelteKit 2.x (Svelte 5), Node.js 22 oraz TypeScript. Główne elementy:

- SvelteKit jako framework do budowy aplikacji
- TypeScript dla bezpieczeństwa typów
- komunikacja z API po HTTPS przez NGINX (żądania **/api/***)

Komponenty frontendowe:

- widoki rejestracji/logowania oraz konfiguracji TOTP
- przeglądarka plików (lista, ulubione, informacje o storage)
- upload plików oraz obsługa wersji pliku
- pobieranie plików (pojedynczo lub ZIP)

3.3 Warstwa backendowa

Backend został zaimplementowany w Pythonie z wykorzystaniem FastAPI. Wykorzystane technologie:

- FastAPI jako framework webowy
- SQLAlchemy (async) z asyncpg dla asynchronicznej komunikacji z bazą
- Pydantic do walidacji danych
- JWT dla autentykacji
- Argon2id (Passlib) do hashowania haseł
- Python-multipart do obsługi uploadu plików
- Boto3 do komunikacji z MinIO (S3 API)
- PyOTP + qrcode do obsługi TOTP

Główne endpointy API:

- `/api/v1/users/*` - rejestracja, logowanie, tokeny, dane użytkownika
- `/api/v1/totp/*` - konfiguracja i weryfikacja TOTP
- `/api/v1/files/*` - operacje na plikach oraz wersjach
- `/api/v1/logs/*` - pobieranie logów (admin)

3.4 Baza danych

System wykorzystuje PostgreSQL 17 jako relacyjną bazę danych. Schemat bazy obejmuje następujące tabele:

- **users** - dane użytkowników (w tym status TOTP oraz limity storage)
- **files** - metadane plików (aktualna wersja, rozmiar, ulubione)
- **file_versions** - historia wersji plików wraz ze ścieżką `s3://...`
- **refresh_tokens** - refresh tokeny przechowywane po stronie serwera
- **logs** - logi zdarzeń (kto/co/kiedy)

Relacje między encjami: **users** —> **files** —> **file_versions** oraz **users** —> **refresh_tokens/logs**.

3.5 Object Storage

MinIO pełni rolę kompatybilnego z S3 magazynu obiektów. Wszystkie pliki użytkowników są przechowywane w osobnych bucketach, zorganizowanych według identyfikatorów użytkowników.

Konwencje przyjęte w projekcie:

- **bucket per user:** `user-{username}`
- **nazwa obiektu:** `filename_v{version}.ext` (np. `dokument_v3.txt`)
- w bazie danych przechowywana jest ścieżka w postaci `s3://<bucket>/<key>`

Zalety wykorzystania MinIO:

- Kompatybilność z API S3
- Wysoka wydajność przy dużych plikach
- Możliwość lokalnego uruchomienia
- Łatwa skalowalność

4 Autentykacja i autoryzacja

System wykorzystuje JWT (JSON Web Tokens) do autentykacji. Logowanie wymaga skonfigurowanego TOTP (2FA).

4.1 Hasła i bezpieczeństwo

- hasła są hashowane algorytmem **Argon2id** (time_cost=3, memory_cost=64 MB, parallelism=2)
- refresh tokeny są przechowywane w bazie danych (tabela `refresh_tokens`); wylogowanie usuwa wszystkie refresh tokeny użytkownika

4.2 Rodzaje tokenów

Token	Ważność	Przeznaczenie
Access Token	15 minut	Autoryzacja requestów do API
Refresh Token	1 dzień	Odświeżanie access tokena
Setup Token (TOTP)	15 minut	Konfiguracja TOTP po rejestracji/logowaniu

4.3 Przykładowe payloady JWT

```
1 {
2   "sub": "username",
3   "iss": "SPCloud",
4   "iat": 1736851200,
5   "nbf": 1736851200,
6   "exp": 1736852100,
7   "jti": "abc123..."
8 }
```

Kod 1: Przykładowy payload Access Token (JWT)

```
1 {
2   "sub": "username",
3   "iss": "SPCloud",
4   "iat": 1736851200,
5   "nbf": 1736851200,
6   "exp": 1736937600,
7   "jti": "xyz789...",
8   "type": "refresh"
9 }
```

Kod 2: Przykładowy payload Refresh Token (JWT)

```
1 {
2   "sub": "username",
3   "exp": 1736852100,
4   "type": "totp_setup"
5 }
```

Kod 3: Przykładowy payload Setup Token (TOTP)

4.4 Flow rejestracji i logowania (TOTP)

Po rejestracji oraz przy pierwszym logowaniu użytkownik otrzymuje **Setup Token** do konfiguracji TOTP. Po skonfigurowaniu aplikacji uwierzytelniającej (np. Google Authenticator) użytkownik loguje się już zawsze z kodem TOTP.

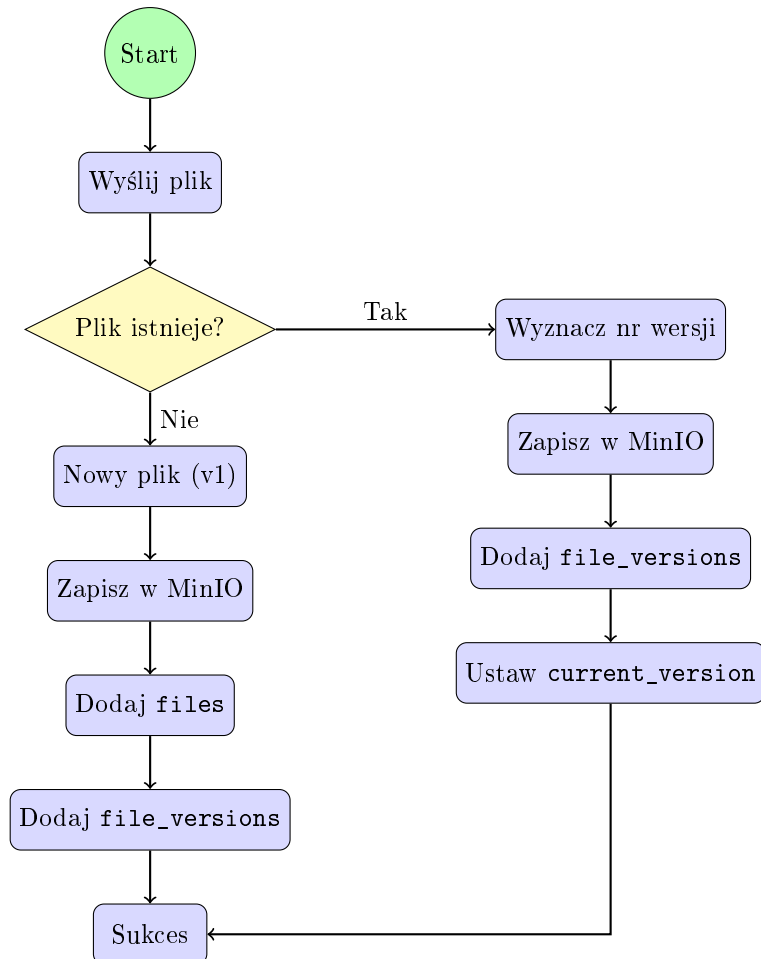
```
1 POST /api/v1/users/register -> Setup Token (TOTP, 15 min)
2 POST /api/v1/totp/setup -> QR Code + secret (wymaga Setup Token)
3 POST /api/v1/totp/verify -> Access + Refresh Token (wymaga Setup Token)
4
5 POST /api/v1/users/login -> jeśli TOTP nie skonfigurowany: Setup Token
6                           jeśli TOTP skonfigurowany: 403 "TOTP verification required"
7 POST /api/v1/users/login/totp -> Access + Refresh Token
```

Kod 4: Schemat rejestracji/logowania (uproszczony)

5 Wersjonowanie plików

System automatycznie tworzy wersje plików przy każdej zmianie, umożliwiając przywracanie wcześniejszych stanów.

5.1 Wersjonowanie: zasady działania



Rysunek 2: Diagram BPMN - Wersjonowanie plików

Mechanizm wersjonowania działa następująco:

1. Przesłany plik jest weryfikowany pod kątem istnienia
2. Jeśli plik istnieje (ta sama nazwa w obrębie użytkownika), tworzona jest nowa wersja $vN+1$
3. Obiekt w MinIO jest zapisywany pod nazwą `filename_v{n}.ext`
4. W bazie danych aktualizowane są metadane i pole `current_version`

Przywracanie wersji nie kopiuje danych w MinIO — system jedynie zmienia `current_version` w tabeli `files`.

5.2 Model danych plików

```
1 files (  
2   id UUID PK,  
3   name TEXT,  
4   path TEXT,  
5   owner TEXT FK -> users.username,
```

```

6   current_version INT,
7   size INT,
8   is_favorite BOOL,
9   created_at TIMESTAMPTZ,
10  updated_at TIMESTAMPTZ,
11  UNIQUE(owner, name)
12 )
13
14 file_versions (
15   id UUID PK,
16   file_id UUID FK -> files.id,
17   version_number INT,
18   path TEXT,
19   size INT,
20   created_at TIMESTAMPTZ,
21   created_by TEXT FK -> users.username
22 )

```

Kod 5: Fragment schematu bazy danych (plik + wersje)

6 Endpointy API

Wszystkie endpointy FastAPI są wystawione pod prefiksem `/api/v1`. Autoryzacja odbywa się przez nagłówek `Authorization: Bearer <token>`.

6.1 Użytkownicy (users)

Metoda	Endpoint	Opis
POST	<code>/users/register</code>	Rejestracja (zwraca Setup Token do TOTP)
POST	<code>/users/login</code>	Logowanie (Setup Token jeśli TOTP nie skonfigurowany, w przeciwnym razie 403)
POST	<code>/users/login/totp</code>	Logowanie z kodem TOTP (zwraca Access+Refresh)
POST	<code>/users/refresh</code>	Odświeżenie Access Tokenu na podstawie Refresh Tokenu
POST	<code>/users/logout</code>	Wylogowanie (usuwa refresh tokeny użytkownika)
GET	<code>/users/me</code>	Dane użytkownika + limity storage + status TOTP
GET	<code>/users/isadmin</code>	Informacja czy użytkownik ma rolę <code>admin</code>

6.2 TOTP

Endpointy `/totp/setup` oraz `/totp/verify` wymagają **Setup Token** (token typu `totp_setup`).

Metoda	Endpoint	Opis
POST	<code>/totp/setup</code>	Generuje sekret oraz QR Code do konfiguracji aplikacji TOTP
POST	<code>/totp/verify</code>	Weryfikuje kod TOTP i zwraca Access+Refresh Token
GET	<code>/totp/status</code>	Zwraca status konfiguracji TOTP dla użytkownika

6.3 Pliki (files)

Metoda	Endpoint	Opis
POST	/files/upload	Upload pliku (nowy plik lub nowa wersja)
GET	/files/	Lista plików użytkownika
GET	/files/me	Informacje o wykorzystaniu storage i statystyki
GET	/files/download/{id}	Pobranie aktualnej wersji pliku
POST	/files/download	Pobranie wielu plików jako ZIP
DELETE	/files/{id}	Usunięcie pliku (wszystkie wersje)
POST	/files/change-is-favorite	Oznaczenie/odznaczenie pliku jako ulubiony
GET	/files/{id}/versions	Lista wersji pliku
GET	/files/{id}/versions/{n}	Pobranie konkretnej wersji n
POST	/files/{id}/restore/{n}	Przywrócenie wersji n (zmiana <code>current_version</code>)
DELETE	/files/{id}/versions/{n}	Usunięcie wersji n (nie można usunąć aktualnej)

6.4 Logi (admin)

Metoda	Endpoint	Opis
GET	/logs/download/{limit}	Pobranie ostatnich limit logów (wymaga roli admin)

7 Logowanie zdarzeń

Każda istotna akcja jest logowana do tabeli `logs`. Log zawiera m.in. timestamp (UTC), nazwę użytkownika, akcję, status (SUCCESS/FAILED), opcjonalnie `file_id` oraz pole `details` (JSON jako string, np. IP klienta).

```

1 LOGIN, LOGOUT, REGISTER,
2 FILE_UPLOAD, FILE_DOWNLOAD, FILE_MANY_DOWNLOAD, FILE_DELETE,
3 FILE_FAVORITE, FILE_UNFAVORITE,
4 FILE_VERSION_CREATE, FILE_VERSION_RESTORE, FILE_VERSION_DELETE,
5 LOG_DOWNLOAD

```

Kod 6: Lista kluczowych typów akcji logowanych w systemie

8 Instalacja i uruchomienie

System może być uruchomiony w środowisku lokalnym lub na serwerze produkcyjnym.

8.1 Wymagania

- Docker Engine 24.x
- Docker Compose V2
- Minimum 4GB RAM
- 20GB wolnego miejsca na dysku

8.2 Konfiguracja

Zmienne środowiskowe konfigurowane są poprzez plik `.env`. Najważniejsze zmienne backendu:

Zmienna	Opis
DB_URL	Connection string do PostgreSQL (SQLAlchemy async)
JWT_SECRET	Sekret do podpisu JWT
JWT_EXPIRE_MIN	Czas życia Access Tokenu (minuty)
JWT_REFRESH_EXPIRE_DAYS	Czas życia Refresh Tokenu (dni)
JWT_ISSUER	Issuer w JWT
MINIO_ENDPOINT	Adres serwera MinIO (np. minio:9000)
MINIO_ACCESS_KEY	Klucz dostępu MinIO
MINIO_SECRET_KEY	Sekret dostępu MinIO
MINIO_SECURE	Czy używać SSL do MinIO (true/false)

8.3 Uruchomienie

1. Sklonuj repozytorium
2. Skonfiguruj plik `.env`
3. Uruchom kontenery: `docker compose up -d -build`
4. Backend inicjalizuje strukturę bazy danych przy starcie aplikacji
5. Otwórz przeglądarkę pod adresem `https://localhost` (frontend)

8.4 Porty i routing

- **NGINX:** 80 (redirect do HTTPS) oraz 443 (reverse proxy)
- **Frontend:** domyślnie 3000 (kontener) oraz port hosta `$FRONTEND_PORT` (domyślnie 3000)
- **Backend:** 8000
- **MinIO:** 9000 (S3 API) oraz 9001 (konsola)
- **PostgreSQL:** 5432 w sieci kontenerów oraz port hosta `$POSTGRES_PORT`

NGINX przekazuje ruch do backendu dla ścieżek `/api/*` oraz do frontendu dla pozostałych ścieżek.

8.5 Infrastruktura produkcyjna

Środowisko produkcyjne zostało uruchomione na Raspberry Pi 5 z następującą konfiguracją:

- System: Ubuntu Server 24.04 LTS
- Docker z wtyczką Compose
- NGINX jako reverse proxy
- Certyfikaty Let's Encrypt dla HTTPS

9 Podsumowanie

SPCloud to system przechowywania plików w chmurze z autentykacją dwuskładnikową (TOTP), wersjonowaniem plików oraz separacją danych w magazynie obiektowym (bucket per user). System został uruchomiony w architekturze kontenerowej (Docker Compose) z reverse proxy NGINX.

Potencjalne kierunki rozwoju:

- dodanie szyfrowania plików po stronie klienta
- rozbudowa o role/uprawnienia oraz panel administracyjny
- dodanie mechanizmu udostępniania plików (linki czasowe)
- usprawnienie obsługi bardzo dużych plików (np. multipart upload)