SIGMA

Elektroniczny Dziennik Szkolny

Dokumentacja Projektu

Wersja 1.0 — Marzec 2026

Zespół BazantCorp

Backend: Spring Boot 3.4 · Java 23 · PostgreSQL
Mobile: Android · Kotlin · Jetpack Compose

1. Cel projektu

SIGMA to elektroniczny dziennik szkolny tworzony przez zespół BazantCorp. System docelowo zastępuje papierową dokumentację szkolną, umożliwiając zarządzanie ocenami, frekwencją i komunikacją między uczniami, nauczycielami a administracją szkoły.

Aplikacja składa się z dwóch głównych komponentów:

- **Backend:** REST API (backend) — udostępnia dane aplikacji mobilnej przez zabezpieczone endpointy.
- **Mobile:** Aplikacja Android (Kotlin + Jetpack Compose) — mobilny klient dla uczniów, nauczycieli i administratorów.

> *Projekt jest rozwijany iteracyjnie. Niniejsza dokumentacja opisuje aktualny stan wersji 1.0, obejmując moduł autentykacji, konfigurację środowiska oraz proces migracji bazy danych.*

2. Stack technologiczny

2.1 Backend

Technologia	Wersja	Zastosowanie
Spring Boot	3.4.2	Główny framework aplikacji
Java	23 (LTS)	Język programowania backendu
Spring Security	6.x	Autentykacja i autoryzacja
Hibernate / JPA	6.x	ORM, zarządzanie schematem bazy danych
jjwt	0.12.6	Generowanie i walidacja tokenów JWT
PostgreSQL	16	Relacyjna baza danych (środowisko Docker)
Flyway	Community	Wersjonowanie i migracje schematu bazy
springdoc-openapi	2.7.0	Swagger UI / dokumentacja REST API
Gradle	—	System budowania projektu
Lombok	latest	Redukcja boilerplate w klasach Java

2.2 Mobile (Android)

Technologia	Zastosowanie
Kotlin	Język programowania aplikacji mobilnej
Jetpack Compose	Deklaratywny UI toolkit dla Androida
MVVM	Wzorzec architektoniczny (ViewModel + StateFlow / LiveData)
Retrofit / OkHttp	Komunikacja z REST API backendu
Android Keystore	Bezpieczne przechowywanie kluczy i tokenów

2.3 Infrastruktura i narzędzia

Narzędzie	Zastosowanie
Docker / Docker Compose	Lokalne uruchamianie bazy PostgreSQL i backendu
GitHub Actions	CI/CD — testy, build i deploy na serwer
Git / GitHub	Repozytorium kodu i zarządzanie Pull Requestami
Jira	Zarządzanie zadaniami i backlogiem projektu
IntelliJ IDEA	Główne IDE dla backendu (obsługa pliku .env)

3. Architektura systemu

3.1 Warstwy backendu

Backend zbudowany jest zgodnie z warstwową konwencją Spring Boot:

Warstwa	Odpowiedzialność
Controller	Obsługa requestów HTTP, walidacja DTO (AuthController, UserController)
Service	Logika biznesowa (AuthService, JwtService, RefreshTokenService)
Repository	Dostęp do bazy danych przez Spring Data JPA
Entity	Encje Hibernate: User, RefreshToken, UserRole
Filter	JwtAuthenticationFilter — weryfikacja tokenów przy każdym żądaniu
Config	SecurityConfig, JwtProperties, OpenApiConfig, obsługa błędów 401/403
DTO	Obiekty transferu danych: request/response (RegisterRequest, AuthResponse…)
Exception	GlobalExceptionHandler, dedykowane wyjątki biznesowe

3.2 Baza danych

PostgreSQL 16 uruchamiany jest w kontenerze Docker. Schemat zarządzany jest przez Flyway — migracje wersjonowane plikami SQL w ścieżce:

> *sigma-backend/src/main/resources/db/migration/ (migracja inicjalna: V1__init_schema.sql)*

Flyway uruchamia się automatycznie przy starcie Spring Boot. Aplikacja skonfigurowana jest z spring.jpa.hibernate.ddl-auto=validate — brak zgodności schematu blokuje start aplikacji, co chroni przed uruchomieniem na niezmigrowanej bazie.

4. Moduł autentykacji

4.1 Strategia tokenów

System używa dwupoziomowego mechanizmu tokenów:

Token	Charakterystyka
Access Token (JWT)	Krótkotrwały (15 min), algorytm HS512, stateless. Przesyłany w nagłówku Authorization: Bearer.
Refresh Token (Opaque)	Długotrwały (30 dni), UUID przechowywany w tabeli refresh_tokens w PostgreSQL. Przesyłany w ciele żądania.

4.2 Token Rotation i Reuse Detection

Każde wywołanie /auth/refresh unieważnia stary token (revokeReason=ROTATED) i tworzy nowy w tej samej rodzinie (family_id). Jeśli klient spróbuje użyć już unieważnionego tokenu, cała rodzina tokenów jest natychmiast unieważniana — skutecznie blokując atakującego posiadającego skradziony token.

4.3 Role użytkowników

Rola	Uprawnienia
ADMIN	Pełny dostęp do wszystkich endpointów, w tym /api/v1/admin/**
TEACHER	Dostęp do endpointów /api/v1/teacher/** i niżej
STUDENT	Dostęp do endpointów publicznych i własnego profilu (/api/v1/me)

4.4 Endpointy autentykacji

Endpoint	Opis
POST /api/v1/auth/register	Rejestracja nowego konta (domyślna rola: STUDENT)
POST /api/v1/auth/login	Logowanie — zwraca access token + refresh token
POST /api/v1/auth/refresh	Wymiana refresh tokenu na nową parę tokenów (rotation)
POST /api/v1/auth/logout	Wylogowanie z bieżącego urządzenia (revoke sesji)
POST /api/v1/auth/logout-all	Wylogowanie ze wszystkich urządzeń (revoke wszystkich tokenów)
GET /api/v1/me	Profil zalogowanego użytkownika (wymaga access tokenu)

Swagger UI dostępny lokalnie pod adresem: http://localhost:8080/swagger-ui.html

5. Konfiguracja środowiska

5.1 Plik .env

Projekt używa pliku .env do konfiguracji wrażliwych danych (baza danych, klucze JWT). Plik jest dodany do .gitignore i nie jest commitowany do repozytorium — każdy członek zespołu konfiguruje go lokalnie.

> ⚠ *Plik .env zawiera dane wrażliwe. Nigdy nie wgrywaj go do repozytorium Git, nie wysyłaj mailem ani nie udostępniaj publicznie.*

5.2 Wymagane zmienne środowiskowe

Zmienna	Opis
POSTGRES_DB	Nazwa bazy danych (domyślnie: sigma_db)
POSTGRES_USER	Użytkownik PostgreSQL (domyślnie: sigma)
POSTGRES_PASSWORD	Hasło do bazy danych
DB_HOST / DB_PORT	Host i port bazy danych (Docker: localhost:5433)
JWT_SECRET_KEY	Min. 512-bitowy klucz do podpisywania tokenów HS512
JWT_EXPIRATION	Czas życia access tokenu w ms (domyślnie: 900000 = 15 min)
JWT_REFRESH_EXPIRATION	Czas życia refresh tokenu w ms (domyślnie: 2592000000 = 30 dni)
JWT_ISSUER	Issuer w claims JWT (domyślnie: sigma-backend)

5.3 Konfiguracja w IntelliJ IDEA

Aby IntelliJ automatycznie wczytywał zmienne z pliku .env przy uruchomieniu:

- Otwórz konfigurację uruchomienia (Run → Modify Run Configuration…)
- Zaznacz checkbox Enable EnvFile
- Kliknij + → .env file i wskaż plik .env w katalogu głównym projektu
- Zapisz konfigurację i uruchom aplikację normalnie (Shift+F10)

> *Funkcja EnvFile jest wbudowana w IntelliJ IDEA 2020.3 i nowsze. Starsze wersje wymagają wtyczki EnvFile.*

6. Migracje bazy danych (Flyway)

6.1 Jak działa migracja

Flyway uruchamia się automatycznie wraz ze startem aplikacji Spring Boot. Historia wykonanych migracji przechowywana jest w tabeli flyway_schema_history. Aplikacja skonfigurowana z ddl-auto=validate — nie startuje, jeśli schemat nie jest zgodny z aktualną migracją.

6.2 Uruchomienie lokalne

Krok	Działanie
1. Przejdź do katalogu	cd sigma-backend
2. Utwórz/uzupełnij .env	Ustaw realne wartości zmiennych środowiskowych
3. Uruchom testy	./gradlew clean test

4. Uruchom aplikację	./gradlew bootRun
5. Sprawdź logi	Szukaj: "Successfully applied ... migration" lub "Schema is up to date"

6.3 Deploy (GitHub Actions)

Branch deploy: wyłącznie master. Pipeline w .github/workflows/deploy.yml wykonuje kolejno:

- Uruchamia testy backendu
- Łączy się z serwerem przez SSH
- Tworzy plik .env na serwerze z GitHub Secrets
- Uruchamia docker compose up -d --build backend
- Weryfikuje logi Flyway — brak potwierdzenia migracji przerywa deploy

6.4 Rollback — procedura

Flyway Community nie wspiera automatycznego cofania migracji. Stosowana procedura manualna:

Krok	Opis
1. Backup przed deployem	pg_dump -Fc -f backup_before_release.dump
2. Migracja się wysypała	Aplikacja nie startuje (oczekiwane). Sprawdź logi: docker logs sigma-backend
3. Przywrócenie bazy	dropdb → createdb → pg_restore backup_before_release.dump
4. Ponowny deploy	Popraw migrację i uruchom pipeline ponownie

7. Zasady współpracy zespołu

7.1 Strategia gałęzi (Git)

Główny branch to master — chroniony przed bezpośrednimi commitami. Wszelkie zmiany trafiają wyłącznie przez zatwierdzony Pull Request.

Konwencja nazw branchy:

> *<czasownik>-<opis-zadania> np. implement-login-screen · fix-grade-calculation-bug*

Dozwolone przedrostki: implement, fix, refactor, add, remove, update. Nazwy pisane małymi literami, słowa oddzielone myślnikami.

7.2 Commit messages

Commit messages pisane po angielsku, format Conventional Commits, max 72 znaki. Przykłady:

- Implement JWT login flow
- Fix grade calculation for weighted average
- Refactor student repository layer

7.3 Proces Pull Request

Krok	Opis
1. Otwórz PR	base: master, compare: twój branch. Uzupełnij opis wg szablonu.
2. Assign reviewer	Poproś minimum 1 osobę z zespołu o code review.
3. Zbierz Approve	Wymagany minimum 1 Approve od innego członka zespołu.
4. Merge	AUTOR PR-a wykonuje merge po zebraniu wymaganych approve'ów.
5. Cleanup	AUTOR PR-a zamyka PR i usuwa branch zadania.

> *Kluczowa zasada: osoba, która otworzyła PR, zbiera approve'y i sama robi merge. Nikt inny nie merguje cudzego PR-a bez wyraźnej prośby autora.*

7.4 Zarządzanie zadaniami (Jira)

Status	Znaczenie
To Do	Zaplanowane, nierozpoczęte
In Progress	Aktywnie realizowane — branch otwarty
In Review	PR otwarty, czeka na code review
Done	PR zmergowany, branch usunięty

8. Dobre praktyki

8.1 Bezpieczeństwo

- Nigdy nie commituj sekretów, kluczy API ani pliku .env do repozytorium
- Hasła użytkowników hashowane BCrypt z cost factor 12
- Tokeny JWT podpisywane HS512 (min. 512-bitowy klucz)
- Android: tokeny przechowywane w EncryptedSharedPreferences / Android Keystore
- Jeden PR = jedno zadanie. Unikaj "workhorse PR-ów" z dziesiątkami zmian.

8.2 Backend — Spring Boot

- Architektura warstwowa: Controller → Service → Repository
- Endpointy REST dokumentowane adnotacjami Swagger/OpenAPI
- Testy jednostkowe dla warstwy serwisowej obowiązkowe dla krytycznych funkcji
- Endpointy REST w kebab-case: /api/v1/student-grades

8.3 Mobile — Android Kotlin

- Wzorzec MVVM: ViewModel + StateFlow / LiveData

- Composables bezstanowe — stan zarządzany wyłącznie w ViewModel
- Brak logiki biznesowej w Activity/Fragment
- Zasoby (stringi, kolory, wymiary) definiowane w plikach XML, nie na sztywno

9. Kontakt i zasoby

Zasób	Lokalizacja
Repozytorium	GitHub — projekt sigma-dziennik
Zarządzanie zadaniami	Jira — projekt SIGMA
Dokumentacja API	http://localhost:8080/swagger-ui.html (lokalnie)
Środowisko produkcyjne	GitHub Actions / deploy branch: master