Politechnika Wrocławska

# Wyniki etapu III: Ocena architektury systemu: System e-learningowy

---

# **Sprawozdanie**

---

*Prowadzący:*
dr inż. Bogumiła Hnatkowska

*Wykonali:*
Jakub Staniszewski 266876
Kamil Wojcieszak 264487
Kasjan Kardaś 263505

Wrocław, 26 Styczeń 2026r.

# Spis treści

# 1) Przegląd podejść architektonicznych

Architektura ocenianego systemu e-learningowego została zaprojektowana z myślą o spełnieniu kluczowych wymagań jakościowych, takich jak wydajność, skalowalność, bezpieczeństwo, wysoka dostępność oraz zgodność z RODO. System wykorzystuje architekturę warstwową, wdrożoną w środowisku chmurowym.

Podstawowy podział architektury obejmuje:

- **Warstwę prezentacji (Frontend)** – aplikację typu SPA zrealizowaną w technologii React, serwowaną przez Nginx, odpowiedzialną za interakcję z użytkownikiem końcowym.
- **Warstwę logiki biznesowej (Backend)** – aplikację backendową (Python), udostępniającą API obsługujące użytkowników, kursy, testy, postępy oraz mechanizmy realizujące wymagania RODO.
- **Warstwę danych** – relacyjną bazę danych PostgreSQL w konfiguracji klastrowej lub replikacyjnej oraz magazyn obiektowy Amazon S3, przeznaczony do przechowywania plików multimedialnych.

Komunikacja pomiędzy warstwami realizowana jest wyłącznie poprzez zdefiniowane interfejsy API, co zapewnia luźne powiązania pomiędzy komponentami oraz umożliwia ich niezależny rozwój i skalowanie. System został wdrożony w architekturze wieloinstancyjnej, z wykorzystaniem mechanizmów autoskalowania i load balancingu, eliminujących pojedyncze punkty awarii.

W zakresie bezpieczeństwa zastosowano szyfrowanie całej komunikacji z użyciem protokołu TLS oraz bezpieczne przechowywanie haseł użytkowników w postaci skrótów bcrypt z losową solą. Wymagania RODO realizowane są poprzez wydzielony moduł logiczny backendu, odpowiedzialny za kontrolowane przetwarzanie danych osobowych, ich eksport oraz trwałe usuwanie.

# 2) Drzewo użyteczności

# 3) Analiza wybranych scenariuszy

# 4) Punkty wrażliwości i kompromisy

# 5) Ryzyka i nie-ryzyka

## 5.1) Ryzyka

- **Baza danych jako potencjalne wąskie gardło**

  Przy dużej liczbie użytkowników jednocześnie korzystających z systemu, na przykład podczas równoczesnego rozwiązywania testów, relacyjna baza danych może stać się elementem ograniczającym wydajność całego systemu.

- **Brak mechanizmu cache**

  W zaprojektowanej architekturze nie przewidziano dodatkowej warstwy pamięci podręcznej, co może prowadzić do częstych odwołań do bazy danych oraz zwiększonego obciążenia warstwy backendowej.

- **Złożoność infrastruktury**

  Wykorzystanie mechanizmów autoskalowania, load balancingu oraz klastrów baz danych zwiększa stopień skomplikowania infrastruktury, co może utrudniać jej utrzymanie, monitorowanie oraz diagnozowanie błędów.

- **Rzadkie pełne kopie zapasowe bazy danych**

  Pełne kopie zapasowe wykonywane są raz w miesiącu. W przypadku poważnej awarii lub błędu logicznego możliwa jest utrata danych z dłuższego okresu czasu, mimo wykonywania kopii przyrostowych.

- **Brak jawnie opisanych mechanizmów monitoringu**

  Brak szczegółowo opisanych mechanizmów monitoringu i alertowania może utrudniać szybkie wykrywanie problemów wydajnościowych i dostępnościowych.

### 5.2) Nie-ryzyka

- **Skalowalność aplikacji**

  Dzięki uruchomieniu systemu w architekturze wieloinstancyjnej oraz zastosowaniu mechanizmów autoskalowania, aplikacja jest przygotowana na wzrost liczby użytkowników.

- **Bezpieczeństwo haseł**

  Hasła użytkowników są przechowywane w postaci bezpiecznych skrótów z wykorzystaniem algorytmu bcrypt, co znacząco zmniejsza ryzyko ich przejęcia w przypadku naruszenia bezpieczeństwa.

- **Dostępność systemu**

  Zastosowanie load balancera umożliwia utrzymanie ciągłości działania systemu nawet w przypadku awarii pojedynczej instancji aplikacji.

- **Przechowywanie multimediów poza bazą danych**

  Pliki są w Amazon S3, nie w PostgreSQL. Wydzielenie plików multimedialnych do magazynu obiektowego ogranicza ryzyko problemów wydajnościowych bazy danych.

- **Jasna separacja warstw systemu**

  Zmiany w jednej warstwie nie wymuszają zmian w pozostałych. Architektura warstwowa zmniejsza ryzyko niekontrolowanych zależności pomiędzy komponentami systemu.

## 6) Wnioski