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, 27 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 zostanie 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

Atrybut jakości	Udoskonalenie atrybutu	Scenariusze
Wydajność i skalowalność	Czas odpowiedzi na zapytanie	Utrzymanie czasu odpowiedzi API na poziomie średnio 300ms (95 percentyl)
	Liczba użytkowników	Wsparcie dla co najmniej 1 000 równoczesnych użytkowników bez spadku wydajności
Bezpieczeństwo	Ochrona danych	Wartości haseł wszystkich użytkowników nie mogą być możliwe do przeczytania
	Ochrona komunikacji	Brak możliwości przechwycenia przez użytkownika postronnego komunikacji między klientem a serwerem
Dostępność	Czas dostępności	Zapewnienie dostępności do aplikacji przez co najmniej 99% czasu
	skalowalność	System musi obsłużyć wzmożony ruch użytkowników w okresach szczytowych (np. sesja egzaminacyjna)
RODO	Pozyskiwanie danych	Mechanizm umożliwiający pobranie danych osobowych (JSON/CSV)
	Pozbywanie się danych	Możliwość całkowitego usunięcia konta wraz z danymi w ciągu maksymalnie 30 dni

Tabela 1: Drzewo użyteczności - atrybuty jakości i scenariusze

3) Analiza wybranych scenariuszy

W tej sekcji dokonano analizy czterech kluczowych scenariuszy jakościowych wybranych z drzewa użyteczności.

3.1) Scenariusz 1: Utrzymanie czasu odpowiedzi API na poziomie średnio 300ms (95 percentyl)

Atrybut: Wydajność i skalowalność

Bodziec: Użytkownik wykonuje operację w systemie generującą zapytanie HTTP do API.

Odpowiedź: System zwraca odpowiedź w czasie ≤300ms dla 95% zapytań.

Realizacja:

- **AWS Lambda** - automatyczne skalowanie, niezależne serwisy (Auth, Shop, Showroom and Service)
- **API Gateway** - routing, throttling, zarządzanie priorytetami żądań
- **CloudFront CDN** - cachowanie, geograficzna dystrybucja, minimalizacja opóźnień
- **Rozdzielone bazy danych** - brak blokad między obszarami funkcjonalnymi, optymalizacja per baza

3.2) Scenariusz 2: Wsparcie dla co najmniej 1000 równoczesnych użytkowników bez spadku wydajności

Atrybut: Wydajność i skalowalność

Bodziec: 1000+ użytkowników jednocześnie wykonuje operacje (przeglądanie, zakupy, rezerwacje).

Odpowiedź: System obsługuje wszystkich użytkowników utrzymując czas odpowiedzi <300ms.

Realizacja:

- **Automatyczne skalowanie Lambda** - tworzenie nowych instancji, niezależne skalowanie mikroserwisów
- **API Gateway** - throttling, limity per użytkownik, queue management
- **CloudFront** - serwowanie statycznych zasobów z cache, redukcja żądań do Lambda
- **Rozdzielone bazy** - dedykowany obszar per baza, niezależne skalowanie

3.3) Scenariusz 3: Zapewnienie szyfrowania wszystkich haseł użytkowników

Atrybut: Bezpieczeństwo

Bodziec: Rejestracja, zmiana hasła, logowanie, potencjalny wyciek bazy danych.

Odpowiedź: Hasła zahaszowane (bcrypt/Argon2), brak możliwości odzyskania w postaci jawnej.

Realizacja:

- **Auth Lambda** - implementacja bcrypt/Argon2, haszowanie przed zapisem
- **Mechanizm salt** - unikalny salt per hasło, ochrona przed rainbow table
- **Work factor/memory cost** - regulowana złożoność, odporność na brute force
- **Izolacja Account Database** - dedykowana baza, ograniczony dostęp tylko dla Auth Lambda

3.4) Scenariusz 4: Zapewnienie szyfrowania całej komunikacji między klientem a serwerem

Atrybut: Bezpieczeństwo

Bodziec: Przesyłanie danych wrażliwych (logowanie, płatności, dane osobowe), potencjalny atak MITM.

Odpowiedź: Komunikacja przez HTTPS/TLS, szyfrowanie danych, autentyczność serwera, integralność danych.

Realizacja:
- **CloudFront HTTPS** - terminacja HTTPS, AWS Certificate Manager, wymuszenie HTTPS
- **API Gateway HTTPS** - szyfrowanie end-to-end, konfiguracja minimalnej wersji TLS 1.2+
- **Szyfrowane połączenia DB** - komunikacja Lambda-DB szyfrowana, AWS VPC, encryption at rest
- **Integracja tpay** - HTTPS, weryfikacja certyfikatu SSL/TLS, mechanizmy autoryzacji

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