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

2.0.1) Interfejs 2: Klient - System do zarządzania działalnością salonu samochodowego

Atrybut jakości	Udoskonalenie atrybutu	Scenariusze
Wydajność i skalowalność	Czas odpowiedzi na zapytanie ——————————- Liczba użytkowników	• utrzymanie czasu odpowiedzi API na poziomie średnio 300ms (95 percentyl) • wsparcie dla co najmniej 1 000 równoczesnych użytkowników bez spadku wydajności
Bezpieczeństwo	Przechowywanie danych ——————————- Komunikacja	• zapewnienie szyfrowania wszystkich haseł użytkowników, • zapewnienie szyfrowania całej komunikacji między klientem a serwerem
Dostępność	Czas dostępności	• zapewnienie drugiej klasy dostępności (dostępność usługi przez co najmniej 99% czasu)
RODO	Pozyskiwanie danych, ———-, Pozbywanie się danych	• mechanizm umożliwiający pobranie danych osobowych (JSON/CSV), • możliwość całkowitego usunięcia konta wraz z danymi w ciągu maksymalnie 30 dni

Tabela 1:

3) Analiza wybranych scenariuszy

W tej sekcji dokonano analizy czterech kluczowych scenariuszy jakościowych wybranych z drzewa użyteczności. Dla każdego scenariusza określono sposób jego realizacji w obecnej architekturze oraz zidentyfikowano punkty wrażliwości i kompromisy.

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

3.1.1) Atrybut jakości

Wydajność i skalowalność

3.1.2) Kontekst

System zarządzania działalnością salonu samochodowego obsługuje równoczesne zapytania od wielu użytkowników wykonujących różne operacje (przeglądanie katalogu, składanie zamówień, rezerwacje wizyt). Użytkownik oczekuje płynnego działania aplikacji bez zauważalnych opóźnień.

3.1.3) Bodziec
Użytkownik wykonuje operację w systemie (np. wyszukiwanie części w sklepie, rezerwacja wizyty w serwisie), która generuje zapytanie HTTP do API.

3.1.4) Odpowiedź
System przetwarza żądanie i zwraca odpowiedź w czasie nie przekraczającym 300ms dla 95% wszystkich zapytań.

3.1.5) Realizacja w architekturze
Decyzje architektoniczne wspierające scenariusz:

1. **Architektura mikroserwisowa na AWS Lambda**
   - Funkcje Lambda umożliwiają automatyczne skalowanie w odpowiedzi na wzrost liczby żądań
   - Niezależne serwisy (Auth, Shop, Showroom and Service) mogą być skalowane osobno w zależności od obciążenia
2. **API Gateway jako punkt wejściowy**
   - Zarządza routingiem żądań do odpowiednich funkcji Lambda
   - Umożliwia implementację throttlingu i zarządzania priorytetami żądań
3. **CloudFront CDN dla API i Frontend**
   - Cachowanie odpowiedzi API redukuje obciążenie backendu dla częstych zapytań
   - Geograficzna dystrybucja węzłów CDN minimalizuje opóźnienia sieciowe
   - Statyczne zasoby frontend dostępne z najbliższego punktu dystrybucji
4. **Rozdzielone bazy danych**
   - Account Database, Shop Database oraz Showroom and Service Database działają niezależnie
   - Brak wzajemnych blokad między różnymi obszarami funkcjonalnymi
   - Możliwość optymalizacji każdej bazy pod kątem specyficznych zapytań

3.1.6) Punkty wrażliwości i kompromisy
Punkty wrażliwości:
- Cold start funkcji Lambda może powodować opóźnienia pierwszego żądania
- Zapytania wymagające złączeń między różnymi bazami danych mogą przekroczyć zakładany czas odpowiedzi
- Zbyt agresywne cachowanie w CloudFront może prowadzić do zwracania nieaktualnych danych

Kompromisy:
- Zachowanie minimalnej liczby warm instancji Lambda zwiększa koszty operacyjne, ale redukuje opóźnienia
- Krótszy TTL cache zwiększa świeżość danych, ale obciąża backend większą liczbą żądań
- Denormalizacja danych w bazach poprawia wydajność odczytu, ale komplikuje operacje zapisu

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

3.2.1) Atrybut jakości
Wydajność i skalowalność

3.2.2) Kontekst
W godzinach szczytu (np. podczas promocji, premier nowych modeli) system obsługuje znaczną liczbę równoczesnych użytkowników wykonujących różnorodne operacje.

3.2.3) Bodziec

1000 lub więcej użytkowników jednocześnie korzysta z systemu, wykonując operacje takie jak:

- Przeglądanie oferty samochodów i części
- Dodawanie produktów do koszyka
- Składanie zamówień
- Rezerwacja wizyt w salonie i serwisie

3.2.4) Odpowiedź

System obsługuje wszystkich użytkowników bez degradacji wydajności, utrzymując średni czas odpowiedzi API poniżej 300ms.

3.2.5) Realizacja w architekturze

Decyzje architektoniczne wspierające scenariusz:

1. **Automatyczne skalowanie funkcji Lambda**
   - AWS Lambda automatycznie tworzy nowe instancje funkcji w odpowiedzi na wzrost liczby żądań
   - Każdy mikroserwis (Auth, Shop, Showroom and Service) skaluje się niezależnie
   - Brak konieczności ręcznej konfiguracji klastrów serwerów
2. **API Gateway z zarządzaniem ruchem**
   - Throttling na poziomie API Gateway chroni backend przed przeciążeniem
   - Możliwość ustawienia limitów żądań per użytkownik/API key
   - Queue management dla kolejkowania żądań w okresach szczytowego obciążenia
3. **CloudFront jako warstwa cachowania**
   - Statyczne zasoby (katalog produktów, obrazy, cenniki) serwowane z cache
   - Redukcja liczby żądań docierających do funkcji Lambda
   - Geograficzna dystrybucja obciążenia
4. **Rozdzielone bazy danych**
   - Każda baza (Account, Shop, Showroom and Service) obsługuje dedykowany obszar funkcjonalny
   - Brak współdzielonych zasobów między różnymi typami operacji
   - Możliwość niezależnego skalowania i optymalizacji każdej bazy

3.2.6) Punkty wrażliwości i kompromisy

Punkty wrażliwości:

- Limity AWS Lambda (domyślnie 1000 równoczesnych wykonań na region) mogą wymagać zwiększenia
- Bazy danych mogą stać się wąskim gardłem przy bardzo dużej liczbie operacji zapisu
- Koszty rosną proporcjonalnie do liczby równoczesnych użytkowników (model pay-per-use)

Kompromisy:

- Agresywne cachowanie zmniejsza obciążenie, ale może prowadzić do nieaktualnych danych
- Connection pooling w bazach danych poprawia wydajność, ale zwiększa złożoność zarządzania
- Read replicas dla baz danych zwiększają przepustowość odczytu, ale wprowadzają opóźnienia replikacji

3.3) Scenariusz 3: Zapewnienie szyfrowania wszystkich haseł użytkowników

3.3.1) Atrybut jakości
Bezpieczeństwo

3.3.2) Kontekst
System przechowuje dane uwierzytelniające użytkowników w Account Database. Hasła należą do danych szczególnie wrażliwych i wymagają najwyższego poziomu ochrony.

3.3.3) Bodziec
- Nowy użytkownik rejestruje się w systemie i tworzy hasło
- Użytkownik zmienia swoje hasło
- Użytkownik loguje się do systemu
- Potencjalny scenariusz naruszenia: nieautoryzowany dostęp do bazy danych

3.3.4) Odpowiedź
- Hasła są przechowywane w formie zahaszowanej przy użyciu algorytmu bcrypt lub Argon2
- Weryfikacja hasła podczas logowania nie wymaga jego odszyfrowania
- W przypadku wycieku bazy danych, atakujący nie może odzyskać haseł w postaci jawnej

3.3.5) Realizacja w architekturze
Decyzje architektoniczne wspierające scenariusz:

1. **Auth Lambda z implementacją haszowania**
   - Moduł Auth Lambda odpowiedzialny za przetwarzanie operacji uwierzytelniania
   - Implementacja bezpiecznych algorytmów haszowania (bcrypt/Argon2)
   - Hasła są haszowane przed zapisem do Account Database
2. **Mechanizm salt dla haseł**
   - Każde hasło otrzymuje unikalny salt
   - Ochrona przed atakami rainbow table
   - Salt przechowywany wraz z hashem w bazie danych
3. **Regulowana złożoność obliczeniowa**
   - Parametry work factor (bcrypt) lub memory cost (Argon2) konfigurowane dla zwiększenia odporności na brute force
   - Możliwość dostosowania poziomu zabezpieczeń w zależności od wymagań
4. **Izolacja Account Database**
   - Dedykowana baza danych tylko dla danych uwierzytelniających
   - Ograniczony dostęp tylko dla Auth Lambda
   - Dodatkowa warstwa zabezpieczeń przez separację od innych danych

3.3.6) Punkty wrażliwości i kompromisy
Punkty wrażliwości:
- Implementacja algorytmu haszowania w Auth Lambda - błąd w implementacji może osłabić zabezpieczenia
- Parametry konfiguracyjne (work factor, memory cost) - zbyt niskie wartości osłabiają ochronę
- Dostęp do Account Database - nieodpowiednia konfiguracja uprawnień może umożliwić nieautoryzowany dostęp

Kompromisy:

- Wyższa złożoność obliczeniowa zwiększa bezpieczeństwo, ale wydłuża czas logowania i rejestracji
- Silniejsze parametry haszowania zwiększają obciążenie CPU funkcji Lambda, co przekłada się na wyższe koszty
- Dodatkowa walidacja siły hasła (wymóg długości, znaków specjalnych) poprawia bezpieczeństwo, ale może frustrować użytkowników

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

3.4.1) Atrybut jakości

Bezpieczeństwo

3.4.2) Kontekst

System obsługuje przesyłanie danych wrażliwych między przeglądarką użytkownika a serwerem, w tym danych logowania, informacji osobowych, danych płatności oraz szczegółów rezerwacji.

3.4.3) Bodziec

- Użytkownik loguje się do systemu, przesyłając dane uwierzytelniające
- Użytkownik dokonuje zakupu w sklepie internetowym, podając dane płatnicze
- Użytkownik rezerwuje wizytę w salonie lub serwisie, przesyłając dane osobowe
- Potencjalny atak: próba przechwycenia komunikacji (man-in-the-middle)

3.4.4) Odpowiedź

- Cała komunikacja między klientem a serwerem odbywa się przez HTTPS/TLS
- Dane w trakcie transmisji są zaszyfrowane i chronione przed przechwyceniem
- Zapewniona jest autentyczność serwera (weryfikacja certyfikatu)
- Zachowana jest integralność danych (wykrycie modyfikacji)

3.4.5) Realizacja w architekturze

Decyzje architektoniczne wspierające scenariusz:

1. **CloudFront z obsługą HTTPS**
   - Frontend CloudFront terminuje połączenia HTTPS
   - Automatyczne zarządzanie certyfikatami TLS przez AWS Certificate Manager
   - Wymuszenie HTTPS dla wszystkich żądań (przekierowanie HTTP → HTTPS)
2. **API Gateway z HTTPS**
   - API CloudFront i API Gateway wymagają HTTPS dla wszystkich żądań API
   - Szyfrowanie end-to-end od przeglądarki użytkownika do API Gateway
   - Możliwość konfiguracji minimalnej wersji TLS (TLS 1.2+)
3. **Route 53 z obsługą HTTPS**
   - Routing DNS wspiera protokół HTTPS
   - Możliwość konfiguracji DNSSEC dla dodatkowej warstwy bezpieczeństwa
4. **Szyfrowane połączenia do baz danych**
   - Komunikacja między funkcjami Lambda a bazami danych również szyfrowana
   - Wykorzystanie AWS VPC dla izolacji sieci
   - Encryption at rest dla danych w bazach
5. **Bezpieczna integracja z tpay**
   - Komunikacja z zewnętrznym dostawcą płatności (tpay) przez HTTPS
   - Weryfikacja certyfikatu SSL/TLS dostawcy

- Dodatkowe mechanizmy autoryzacji (API keys, tokeny)

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