Uniwersytet Gdański Wydział Matematyki, Fizyki i Informatyki Instytut Informatyki

Oliver Gruba, Maciej Nasiadka

14 stycznia 2026

Imię i Nazwisko (nr indeksu)	Oliver Gruba (292583) Maciej Nasiadka (292574)
Nazwa uczelni	Uniwersytet Gdański
Kierunek	Informatyka (profil praktyczny)
Prowadzący	dr inż. Stanisław Witkowski
Nazwa ćwiczenia	Projektowanie architektury rozwiązań i mikrousług IT poprzez tworzenie diagramów komponentów.
Numer sprawozdania	10
Data zajęć	08.01.2026
Data oddania	14.01.2026
Miejsce na ocenę	

Spis treści

1 Wprowadzenie **4**

2 Architektura mikrousług a diagram komponentów **4**

2.1 Granice mikrousług 4

2.2 Niezależność wdrożeniowa 4

2.3 Sprzężenie i spójność 4

3 Komunikacja pomiędzy komponentami i mikrousługami **5**

3.1 Komunikacja synchroniczna 5

3.2 Komunikacja asynchroniczna 5

3.3 Modelowanie komunikacji w diagramie 5

4 Notacja symboliczna diagramu komponentów **5**

4.1 Podstawowe elementy 5

4.1.1 Komponent (*component*) 6

4.1.2 Interfejs (*interface*) 6

4.2 Powiązania i relacje 7

4.3 Przepływ danych 7

5 Zasada użycia diagramu komponentów **8**

5.1 Moment wykorzystania 8

5.2 Poziom szczegółowości 9

5.3 Dobre praktyki 9

6 Bezpieczeństwo architektury komponentowej **9**

6.1 Izolacja odpowiedzialności 9

6.2 Punkty kontroli dostępu 9

6.3 Ochrona komunikacji 9

7 Przykład nr 1 — Diagram komponentów systemu ATM (UML) **10**

7.1 Opis systemu 10

7.2 Kluczowe komponenty i interfejsy 10

7.2.1 1. Terminal ATM (Strona Klienta) 11

7.2.2 2. Sieć Bankowa (Backend) 11

7.3 Analiza architektoniczna 11

8 Przykład nr 2 — Diagram komponentów systemu zarządzania zapasami 12
8.1 Opis systemu 13
8.2 Kluczowe komponenty i interfejsy 13
8.2.1 1. Backend (Inventory Core) 13
8.2.2 2. Warstwa Integracji (Data & Integration) 13
8.3 Analiza architektury 14

9 Przykład nr 3 — własna propozycja 14

10 Zastosowanie diagramów komponentów w procesie tworzenia oprogramowania 15
10.1 Wsparcie pracy zespołowej 15
10.2 Wsparcie architektury mikrousług 15
10.3 Utrzymanie i rozwój systemu 15

11 Relacja diagramu komponentów z innymi diagramami UML 15
11.1 Diagram klas a diagram komponentów 16
11.2 Diagram wdrożenia 16
11.3 Diagram sekwencji 16

12 Podsumowanie 16

1. Wprowadzenie

Projektowanie architektury rozwiązań informatycznych oraz mikrousług stanowi jeden z kluczowych etapów procesu wytwarzania nowoczesnych systemów IT. W szczególności w architekturach rozproszonych, takich jak mikroserwisy, konieczne jest precyzyjne określenie odpowiedzialności poszczególnych modułów, ich interfejsów oraz sposobów komunikacji.

Diagram komponentów UML jest narzędziem, które umożliwia modelowanie logicznej struktury systemu poprzez przedstawienie jego elementów funkcjonalnych oraz zależności pomiędzy nimi. Pozwala on na czytelne zobrazowanie architektury systemu zarówno na poziomie pojedynczej aplikacji, jak i całego ekosystemu mikrousług.

Celem niniejszego sprawozdania jest omówienie zasad projektowania architektury rozwiązań IT i mikrousług z wykorzystaniem diagramów komponentów, przedstawienie notacji symbolicznej, a także zaprezentowanie przykładowych zastosowań w rzeczywistych scenariuszach systemowych.

2. Architektura mikrousług a diagram komponentów

Architektura mikrousług zakłada podział systemu na niewielkie, niezależne usługi realizujące pojedyncze odpowiedzialności biznesowe. Diagram komponentów jest naturalnym narzędziem do modelowania takiej struktury.

2.1. Granice mikrousług

- każdy komponent powinien odpowiadać jednej domenowej odpowiedzialności,
- komponent nie powinien zawierać logiki niezwiązanej z jego główną funkcją,
- granice komponentów powinny odpowiadać granicom zespołów projektowych.

2.2. Niezależność wdrożeniowa

- komponent jako mikrousługa powinien być wdrażalny niezależnie,
- zmiana jednego komponentu nie powinna wymuszać rekompilacji innych,
- diagram pozwala sprawdzić, czy nie powstały ukryte zależności.

2.3. Sprzężenie i spójność

- diagram pomaga ograniczać sprzężenie (loose coupling),

- interfejsy wymuszają kontrakt komunikacyjny,
- spójność wewnętrzna komponentu powinna być maksymalna.

3. Komunikacja pomiędzy komponentami i mikrousługami

Sposób komunikacji ma kluczowe znaczenie dla wydajności i stabilności systemu.

3.1. Komunikacja synchroniczna

- realizowana przez REST, gRPC lub SOAP,
- prosta w implementacji, ale podatna na opóźnienia,
- zwiększa ryzyko kaskadowych awarii.

3.2. Komunikacja asynchroniczna

- oparta na kolejkach i brokerach zdarzeń,
- zwiększa odporność systemu,
- umożliwia skalowanie niezależne komponentów.

3.3. Modelowanie komunikacji w diagramie

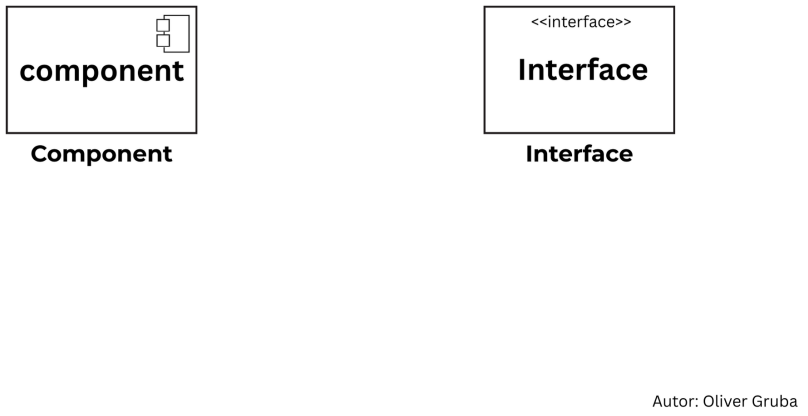
- strzałki pokazują kierunek wywołań,
- podpis relacji określa protokół,
- interfejs definiuje kontrakt komunikacyjny.

4. Notacja symboliczna diagramu komponentów

Notacja diagramu komponentów UML umożliwia jednoznaczne przedstawienie struktury systemu oraz relacji pomiędzy jego elementami. Prawidłowe stosowanie symboli jest kluczowe dla czytelności i poprawnej interpretacji diagramu.

4.1. Podstawowe elementy

Podstawowe elementy diagramu komponentów służą do definiowania modułów systemu oraz punktów komunikacji pomiędzy nimi.



Rysunek 1: Graficzne przedstawienie podstawowych elementów diagramu komponentów

4.1.1 Komponent (*component*)

Komponent reprezentuje logiczny lub fizyczny moduł systemu odpowiedzialny za realizację określonej funkcjonalności. Jest on niezależną jednostką, która może być rozwijana, testowana i wdrażana autonomicznie.

Cechy komponentu:

- posiada jasno określoną odpowiedzialność,
- ukrywa szczegóły implementacyjne,
- komunikuje się z innymi komponentami poprzez interfejsy,
- może reprezentować mikrousługę, moduł aplikacji lub bibliotekę.

4.1.2 Interfejs (*interface*)

Interfejs definiuje zbiór operacji udostępnianych lub wymaganych przez komponent. Stanowi kontrakt komunikacyjny pomiędzy komponentami.

Rozróżnia się:

- interfejsy dostarczane (provided),
- interfejsy wymagane (required).

Zastosowanie interfejsów sprzyja luźnemu powiązaniu komponentów oraz zwiększa elastyczność architektury.

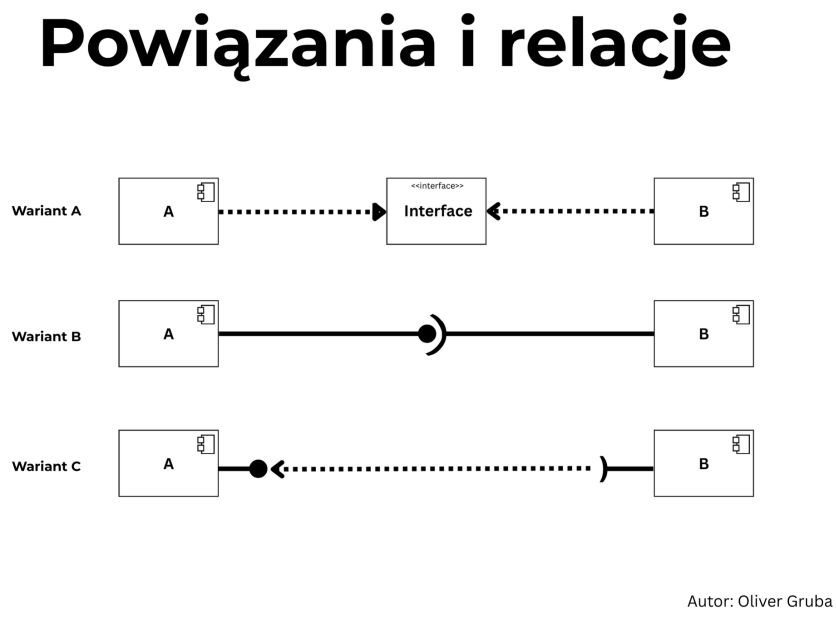
4.2. Powiązania i relacje

Relacje pomiędzy komponentami określają sposób ich współpracy oraz zależności funkcjonalne.

W diagramach komponentów najczęściej stosuje się trzy warianty powiązań:

- **Zależność (Dependency)** – wskazuje, że jeden komponent korzysta z funkcjonalności innego, lecz nie jest z nim silnie związany.
- **Połączenie przez interfejs** – komunikacja realizowana poprzez zdefiniowane interfejsy provided/required.
- **Delegacja** – przekazanie odpowiedzialności obsługi żądania do innego komponentu.

Dobór odpowiedniego rodzaju relacji ma istotny wpływ na czytelność diagramu oraz stopień sprzężenia systemu.



Rysunek 2: Graficzne przedstawienie powiązania i relacji diagramu komponentów

4.3. Przepływ danych

Przepływ danych obrazuje kierunek i charakter przekazywanych informacji pomiędzy komponentami.

Przepływ danych

Autor: Oliver Gruba

Rysunek 3: Graficzne przedstawienie przepływu danych w diagramie komponentów

W przypadku przepływu danych z komponentu B do komponentu A:

- komponent B pełni rolę dostawcy danych lub usług,
- komponent A jest odbiorcą danych,
- przepływ może reprezentować wywołanie usługi, komunikację asynchroniczną lub przesył zdarzeń.

Jawne zaznaczenie przepływu danych pozwala lepiej zrozumieć zależności systemowe oraz potencjalne punkty krytyczne.

5. Zasada użycia diagramu komponentów

Diagram komponentów powinien być stosowany jako narzędzie analizy i projektowania architektury logicznej systemu.

5.1. Moment wykorzystania

Diagramy komponentów tworzy się:

- na etapie projektowania architektury systemu,
- przed implementacją kluczowych modułów,
- podczas refaktoryzacji lub migracji do architektury mikrousług.

5.2. Poziom szczegółowości

- diagram wysokopoziomowy – przedstawia główne komponenty systemu,
- diagram szczegółowy – opisuje wewnętrzną strukturę wybranych modułów,
- diagramy pomocnicze – skupione na konkretnych procesach biznesowych.

5.3. Dobre praktyki

- unikanie nadmiernej liczby komponentów na jednym diagramie,
- konsekwentne nazewnictwo komponentów i interfejsów,
- zachowanie czytelnego kierunku zależności.

6. Bezpieczeństwo architektury komponentowej

Diagram komponentów pomaga projektować system z uwzględnieniem bezpieczeństwa.

6.1. Izolacja odpowiedzialności

- komponenty wrażliwe (np. autoryzacja) powinny być wydzielone,
- ograniczenie dostępu przez interfejsy,
- brak bezpośredniego dostępu do baz danych z wielu komponentów.

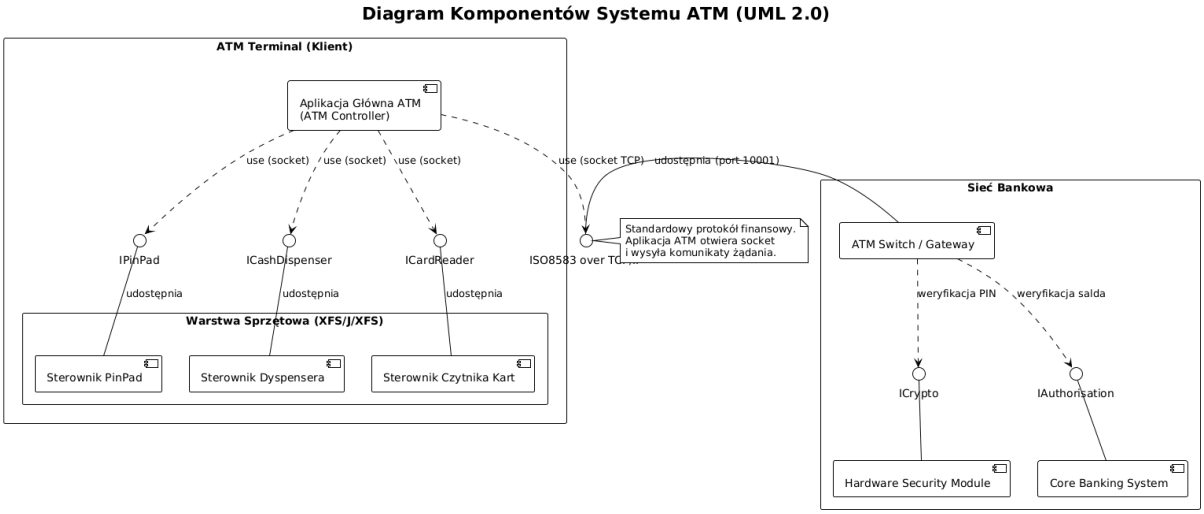
6.2. Punkty kontroli dostępu

- centralne komponenty autoryzacji,
- walidacja tożsamości na granicach systemu,
- diagram ujawnia miejsca wymagające zabezpieczeń.

6.3. Ochrona komunikacji

- szyfrowanie połączeń między komponentami,
- podpisy komunikatów,
- ograniczenie zaufania pomiędzy usługami.

7. Przykład nr 1 — Diagram komponentów systemu ATM (UML)



Rysunek 4: Diagram komponentów systemu bankomatowego z uwzględnieniem interfejsów i sterowników sprzętowych.

System obsługi transakcji bankomatowych został przedstawiony jako architektura rozproszona typu klient-serwer, w której kluczową rolę odgrywa separacja warstwy sprzętowej od logiki biznesowej oraz bezpieczna komunikacja z infrastrukturą banku.

7.1. Opis systemu

Prezentowany model realizuje procesy:

- **Obsługa sprzętu:** Sterowanie peryferiami (czytnik, dyspenser) poprzez dedykowane interfejsy programistyczne.
- **Komunikacja sieciowa:** Wymiana komunikatów w standardzie ISO 8583 przez protokół TCP/IP.
- **Bezpieczeństwo:** Weryfikacja kryptograficzna PIN z użyciem modułów HSM (Hardware Security Module).
- **Księgowanie:** Autoryzacja salda w centralnym systemie bankowym (Core Banking).

7.2. Kluczowe komponenty i interfejsy

Architektura została podzielona na dwa główne węzły logiczne:

7.2.1 1. Terminal ATM (Strona Klienta)

- **ATM Controller (Aplikacja Główna):** Centralny komponent orkiestrujący pracę bankomatu. Nie komunikuje się bezpośrednio ze sprzętem, lecz wykorzystuje interfejsy (tzw. *sockets*).
- **Warstwa Sterowników (Hardware Drivers):** Komponenty realizujące fizyczną obsługę urządzeń. Udostępniają one interfejsy (tzw. *lollipops*):
  - `ICardReader` – obsługa odczytu paska magnetycznego/chipa.
  - `ICashDispenser` – sterowanie mechanizmem wypłaty banknotów.
  - `IPinPad` – bezpieczne wprowadzanie i szyfrowanie PIN.

7.2.2 2. Sieć Bankowa (Backend)

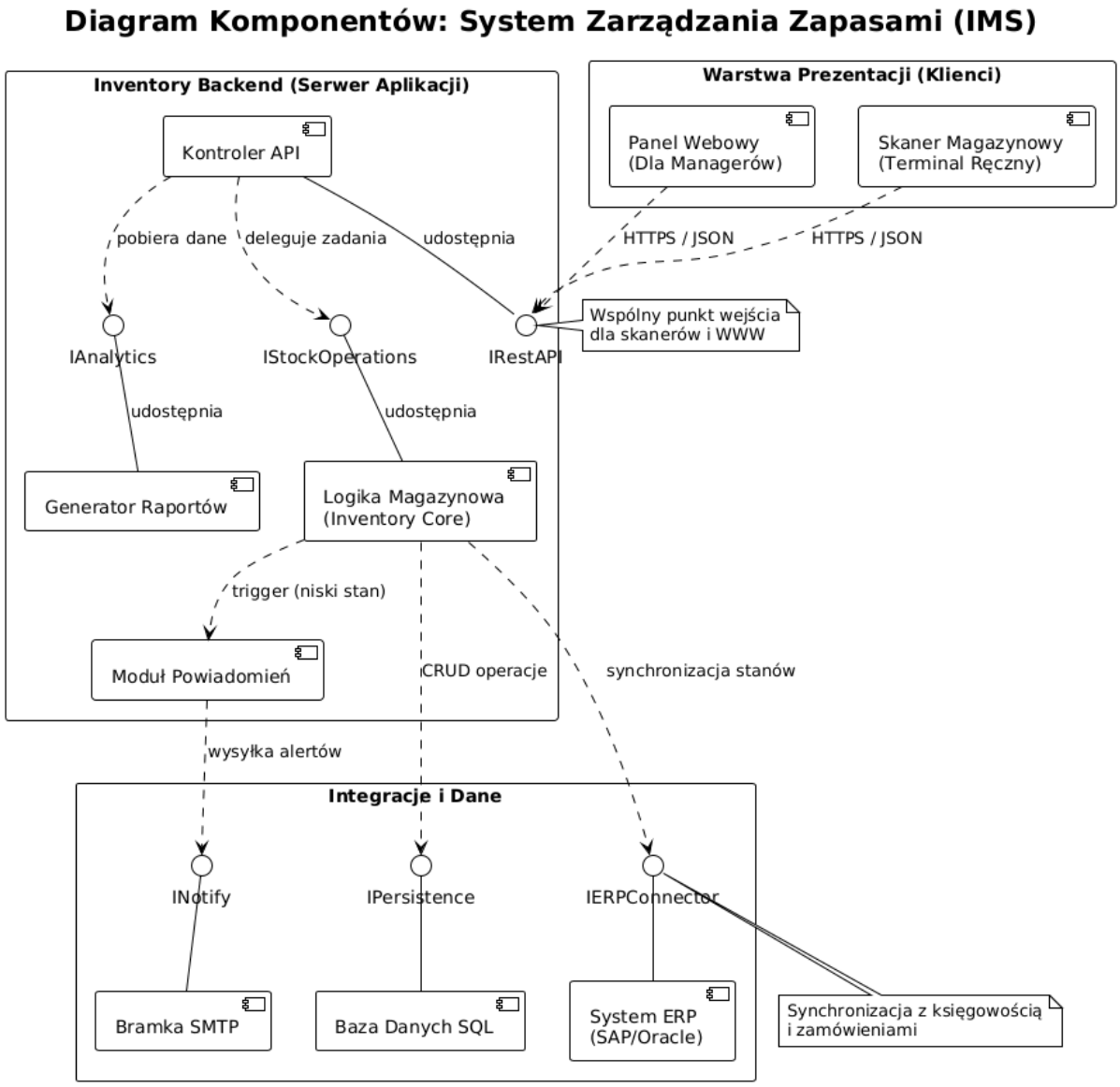
- **ATM Switch:** Brama sieciowa udostępniająca interfejs komunikacyjny (port nasłuchujący), odpowiedzialna za routing transakcji.
- **Core Banking System (CBS):** Udostępnia interfejs `IAuthorisation` do weryfikacji dostępności środków na rachunku.
- **HSM (Hardware Security Module):** Specjalistyczny komponent kryptograficzny udostępniający interfejs `ICrypto` do walidacji bloków PIN.

7.3. Analiza architektoniczna

Zastosowanie diagramu komponentów w notacji UML 2.0 pozwoliło na zidentyfikowanie:

1. **Zależności interfejsowych:** Wykorzystanie notacji „gniazdo-wtyczka” (socket-lollipop) obrazuje luźne powiązania (loose coupling) między aplikacją a sprzętem. Umożliwia to wymianę sterowników (np. na inny model czytnika) bez konieczności modyfikacji głównego kodu sterującego (*ATM Controller*).
2. **Punktów styku sieciowego:** Wyodrębnienie interfejsu `ISO8583 over TCP/IP` definiuje kontrakt komunikacyjny między strefą publiczną (bankomat) a strefą bezpieczną (bank).

8. Przykład nr 2 — Diagram komponentów systemu zarządzania zapasami



Rysunek 5: Diagram komponentów systemu Inventory Management System (IMS) z uwzględnieniem integracji ERP.

Projekt systemu zarządzania zapasami (Inventory Management System - IMS) opiera się na architekturze wielowarstwowej, mającej na celu oddzielenie logiki biznesowej od interfejsów użytkownika oraz zapewnienie spójności danych z zewnętrznymi systemami finansowymi.

8.1. Opis systemu

System IMS odpowiada za:

- śledzenie stanów magazynowych w czasie rzeczywistym,
- obsługę przyjęć i wydań towarów za pomocą terminali mobilnych,
- generowanie raportów dla kadry zarządzającej,
- automatyczną synchronizację z systemem klasy ERP (np. SAP).

8.2. Kluczowe komponenty i interfejsy

Architektura systemu została zdekomponowana na następujące moduły logiczne:

8.2.1 1. Backend (Inventory Core)

Serce systemu, realizujące logikę biznesową. Udostępnia funkcjonalności poprzez zdefiniowane interfejsy:

- **Interfejs `IRestAPI`:** Główny punkt wejścia (Fasada) dla wszystkich klientów. Ujednolica komunikację, sprawiając, że zarówno skaner magazyniera, jak i przeglądarka managera korzystają z tych samych metod.
- **Interfejs `IStockOperations`:** Wewnętrzny kontrakt realizujący operacje atomowe, takie jak *IncreaseStock*, *DecreaseStock* czy *MoveItem*.
- **Moduł Powiadomień:** Komponent nasłuchujący zdarzeń (np. spadek stanu poniżej minimum) i wykorzystujący interfejs `INotify` do wysyłki alertów e-mail przez bramkę SMTP.

8.2.2 2. Warstwa Integracji (Data & Integration)

System nie działa w próżni i wymaga połączenia z infrastrukturą przedsiębiorstwa:

- **System ERP:** Zewnętrzny system nadrzędny. Komponent logiki magazynowej wykorzystuje interfejs `IERPConnector` do raportowania wartości magazynu na potrzeby księgowości.
- **Baza Danych:** Trwałość danych zapewniana jest przez interfejs `IPersistence`, co pozwala na potencjalną wymianę silnika bazy danych (np. z PostgreSQL na Oracle) bez modyfikacji logiki biznesowej.

8.3. Analiza architektury

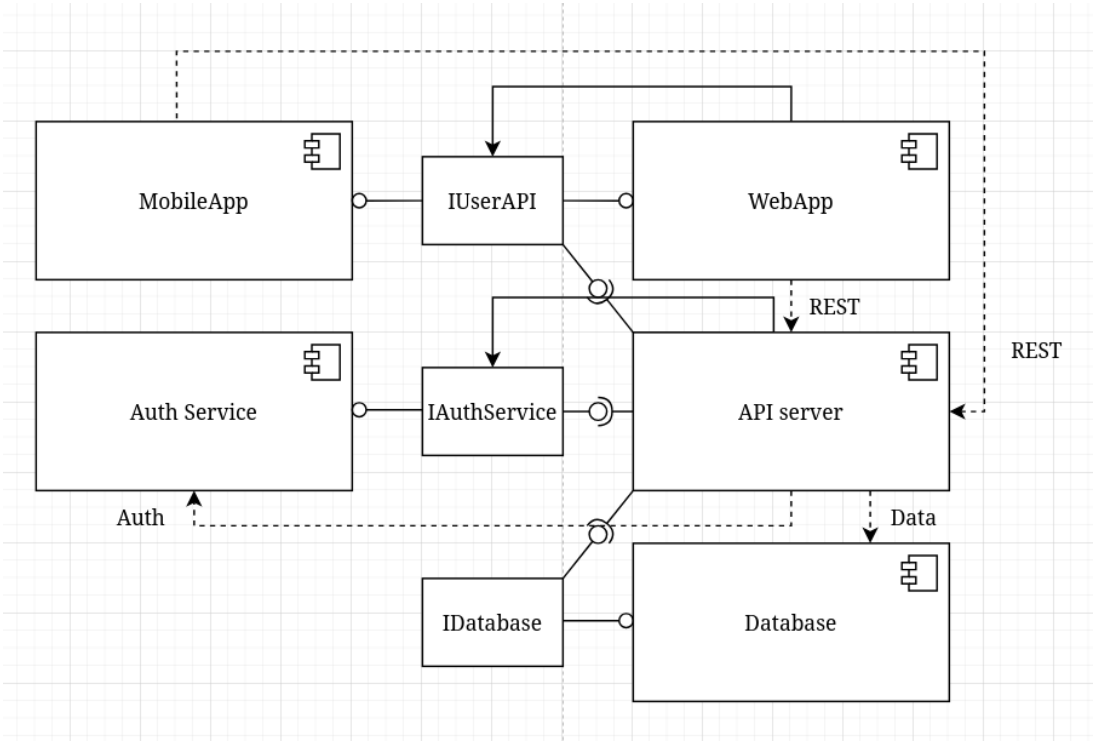
Przedstawiony diagram komponentów uwypukla następujące cechy rozwiązania:

1. **Centralizacja dostępu:** Zastosowanie komponentu *Kontroler API* wystawiającego interfejs REST izoluje logikę biznesową od warstwy prezentacji. Zmiana technologii w terminalach mobilnych (np. z Android na iOS) nie wymaga zmian w backendzie.

2. **Skalowalność integracji:** Wydzielenie interfejsu do ERP pozwala na łatwe tworzenie „mocków" (atrap) podczas testowania, co przyspiesza proces developmentu bez konieczności ciągłego połączenia z produkcyjnym systemem SAP.

9. Przykład nr 3 — własna propozycja

Diagram komponentów przedstawia logiczną strukturę systemu, pokazując główne moduły (komponenty), ich interfejsy oraz zależności pomiędzy nimi. Ułatwia zrozumienie podziału odpowiedzialności oraz komunikacji między częściami systemu.



Rysunek 6: Diagram komponentów przykładowej aplikacji mobilnej.

- **MobileApp** i **WebApp** – dwa różne frontendy korzystające z tych samych interfejsów API.
- **IUserAPI**, **IAuthService**, **IDatabase** – interfejsy definiujące punkty komunikacji pomiędzy komponentami.

- **Auth Service** – odpowiada za uwierzytelnianie, komunikuje się z API server przez interfejs IAuthService.
- **API server** – centralny komponent obsługujący logikę biznesową, komunikuje się z bazą danych (IDatabase) oraz obsługuje żądania REST od aplikacji frontendowych.
- **Database** – przechowuje dane, dostępna przez interfejs IDatabase.
- Połączenia REST i Auth są wyraźnie oznaczone, co ułatwia analizę przepływu danych i bezpieczeństwa.

10. Zastosowanie diagramów komponentów w procesie tworzenia oprogramowania

Diagramy komponentów są szeroko wykorzystywane w praktyce inżynierskiej.

10.1. Wsparcie pracy zespołowej

- wspólne zrozumienie architektury,
- jasny podział odpowiedzialności,
- ułatwienie komunikacji między zespołami.

10.2. Wsparcie architektury mikrousług

- definiowanie granic mikrousług,
- ograniczenie sprzężenia,
- planowanie komunikacji asynchronicznej.

10.3. Utrzymanie i rozwój systemu

- łatwiejsza analiza wpływu zmian,
- wsparcie refaktoryzacji,
- aktualna dokumentacja architektury.

11. Relacja diagramu komponentów z innymi diagramami UML

Diagram komponentów nie funkcjonuje samodzielnie.

11.1. Diagram klas a diagram komponentów

- diagram klas opisuje strukturę wewnętrzną,
- diagram komponentów – strukturę modułów,
- komponent zawiera wiele klas.

11.2. Diagram wdrożenia

- komponent pokazuje *co*,
- wdrożenie pokazuje *gdzie*,
- razem tworzą pełny obraz architektury.

11.3. Diagram sekwencji

- sekwencja pokazuje dynamikę,
- komponent pokazuje strukturę,
- sekwencja wykorzystuje komponenty jako uczestników.

12. Podsumowanie

Analiza architektury poprzez diagramy komponentów pozwala:

- projektować systemy modularne i skalowalne,
- ograniczać chaos architektoniczny,
- planować rozwój systemu w długim horyzoncie.

Diagram komponentów nie jest jedynie dokumentacją, ale narzędziem decyzyjnym, które:

- wpływa na podział pracy zespołowej,
- ułatwia kontrolę jakości architektury,
- zmniejsza ryzyko błędów projektowych.

W architekturach mikrousług jego rola jest szczególnie istotna, ponieważ pozwala zapanować nad złożonością systemów rozproszonych i zapewnić ich długoterminową utrzymywalność.

Diagram komponentów UML stanowi jedno z kluczowych narzędzi w projektowaniu architektury rozwiązań IT oraz systemów opartych o mikrousługi. Pozwala on na przejrzyste przedstawienie struktury systemu, zależności pomiędzy modułami oraz przepływu danych.

Prawidłowe wykorzystanie diagramów komponentów wspiera proces projektowy, zwiększa jakość architektury oraz ułatwia dalszy rozwój i utrzymanie systemu. W kontekście nowoczesnych, rozproszonych systemów informatycznych ich rola staje się szczególnie istotna.