Delta Lake to technologia opracowana przez firmę Databricks, ściśle zintegrowana z platformą Azure Databricks i działająca natywnie z Apache Spark. Umożliwia wykonywanie operacji typu *upsert* (merge), wspiera wersjonowanie danych, funkcję *time-travel* (czyli dostęp do wcześniejszych stanów danych), obsługuje transakcje zgodne z ACID oraz jest zoptymalizowana pod kątem zarówno przetwarzania strumieniowego, jak i wsadowego. Jest to doskonałe rozwiązanie, gdy korzystamy z Databricks i Sparka w celach takich jak analizy biznesowe, transformacje danych czy projekty machine learningowe.

Apache Iceberg to otwarty format wspierany przez społeczność i firmy takie jak Netflix czy Apple. Charakteryzuje się dużą neutralnością technologiczną – współpracuje z wieloma silnikami, w tym Spark, Flink, Trino, Presto czy Snowflake. Świetnie sprawdza się w dużych środowiskach hurtowni danych, gdzie liczy się skalowalność, elastyczność i uniezależnienie od konkretnej platformy (np. Databricks). Iceberg, podobnie jak Delta Lake, obsługuje wersjonowanie danych, time-travel i transakcje ACID.

Rekomendacja dla klienta:

- Jeśli klient pracuje w środowisku Azure Databricks i zależy mu na prostym, wydajnym i dobrze wspieranym rozwiązaniu – najlepszym wyborem będzie **Delta Lake**.
- Jeśli klient potrzebuje rozwiązania otwartego, wieloplatformowego, z integracją różnych silników i dużym naciskiem na skalowalność – lepszym wyborem będzie **Apache Iceberg**.

Architektura medalionowa dzieli dane na trzy warstwy:

- **Bronze** – surowe, nieprzetworzone dane,
- **Silver** – dane oczyszczone i wzbogacone,
- **Gold** – dane przygotowane do analiz biznesowych.

Choć model ten jest szeroko stosowany, ma również swoje wady. Poniżej przedstawiono 20 punktów krytycznych wraz z krótkim uzasadnieniem:

1. **Złożoność utrzymania** – wymaga tworzenia i zarządzania wieloma pipeline’ami między warstwami.

2. **Duplikacja danych** – te same dane mogą być przechowywane wielokrotnie, co zwiększa koszty.
3. **Wysokie zużycie zasobów** – każda warstwa to osobny proces obciążający CPU, RAM i czas obliczeń.
4. **Opóźniony dostęp do danych** – pełna analiza możliwa dopiero po przejściu przez wszystkie warstwy.
5. **Zbędność w małych projektach** – czasem wystarczy jedna lub dwie warstwy.
6. **Trudności w debugowaniu** – błędy mogą pojawiać się na różnych etapach, co utrudnia diagnozę.
7. **Wysoki próg wejścia** – dla małych zespołów architektura może być zbyt złożona.
8. **Brak automatyzacji** – często wymaga ręcznego konfigurowania przepływów danych.
9. **Ograniczona elastyczność** – sztywna struktura utrudnia szybkie zmiany w logice przetwarzania.
10. **Koszty przechowywania** – każda warstwa zwiększa zapotrzebowanie na przestrzeń dyskową.
11. **Nadmierne skupienie na strukturze** – może ograniczać eksplorację danych i inicjatywę analityków.
12. **Konieczność dodatkowego monitoringu** – każda warstwa wymaga osobnych logów i alertów.
13. **Ryzyko niespójności danych** – problemy w jednej warstwie mogą zaburzyć spójność w całym procesie.
14. **Niepraktyczne dla danych jednorazowych** – budowanie pełnej architektury dla danych ad-hoc jest nieefektywne.
15. **Mała użyteczność w strumieniach** – przy przetwarzaniu w czasie rzeczywistym trzy warstwy mogą być zbędne.
16. **Duża liczba plików pośrednich** – generowanie wielu plików pośrednich wymaga zarządzania i zasobów.
17. **Utrudnione testowanie** – testy jednostkowe są trudniejsze do wdrożenia.

18. **Zależność od konkretnych platform** – migracja rozwiązań bywa problematyczna.
19. **Brak standardu** – różne firmy mogą odmiennie definiować warstwy.
20. **Rosnący dług techniczny** – każda warstwa to dodatkowy kod, który trzeba utrzymywać i rozwijać.