

Materiały do przedmiotu:

Bazy danych

Laboratorium nr 8:

Dokumentowe systemy bazodanowe NoSQL

Autor:

Albert Rachwał



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Spis treści

Wprowadzenie, tematyka zajęć	3
Cel ćwiczenia/laboratorium	3
Instrukcja laboratoryjna/ćwiczeniowa	4
Ćwiczenia samodzielne	18
Pytania pomocnicze	20
Podsumowanie	21
Literatura	22



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Wprowadzenie, tematyka zajęć

Bazy dokumentowe stanowią jedną z kluczowych gałęzi systemów NoSQL, zaprojektowanych z myślą o elastycznym i skalowalnym przechowywaniu danych w formacie półstrukturalnym. W odróżnieniu od tradycyjnych relacyjnych baz danych, które wymagają sztywnego schematu tabel i relacji, systemy dokumentowe przechowują dane w postaci obiektów JSON (lub BSON w przypadku MongoDB), umożliwiając dynamiczne dopasowanie struktury danych do potrzeb aplikacji. Ich popularność wynika z rosnących wymagań współczesnych systemów informatycznych – zwłaszcza aplikacji webowych i mobilnych – które generują zróżnicowane dane o zmiennej strukturze i wymagają szybkiego dostępu w czasie rzeczywistym.

W ramach laboratorium studenci poznają koncepcję i architekturę baz dokumentowych, ze szczególnym uwzględnieniem systemu MongoDB jako jednego z najczęściej wykorzystywanych silników tego typu. Omówione zostaną podstawy modelowania danych w formie dokumentów, różnice między zagnieżdżaniem a odwołaniami oraz dobre praktyki projektowania kolekcji. W części praktycznej uczestnicy nauczą się wykonywać operacje CRUD (Create, Read, Update, Delete), tworzyć zapytania filtrujące i agregujące z wykorzystaniem potoków, a także analizować i przekształcać dane przy użyciu operatorów logicznych i obliczeniowych.

Dodatkowo zostaną przedstawione przykłady zastosowań baz dokumentowych w kontekście nowoczesnych rozwiązań, takich jak RethinkDB czy Firebase Firestore, które integrują przechowywanie danych z funkcjami przetwarzania w czasie rzeczywistym.

Celem zajęć jest zrozumienie zasad organizacji danych w modelu dokumentowym oraz wyrobienie umiejętności projektowania i optymalizacji struktur kolekcji w sposób zgodny z logiką aplikacji, a nie ograniczeniami relacyjnego schematu.

Cel ćwiczenia/laboratorium

1. Zapoznanie z koncepcją baz dokumentowych NoSQL oraz ich różnicami względem relacyjnych systemów baz danych.
2. Poznanie zasad przechowywania danych w formacie JSON/BSON oraz organizacji danych w kolekcjach.
3. Zrozumienie architektury i sposobu działania systemu MongoDB jako przykładowej bazy dokumentowej.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



4. Nabycie umiejętności wykonywania operacji CRUD (Create, Read, Update, Delete) na kolekcjach MongoDB.
5. Opanowanie technik filtrowania danych z wykorzystaniem operatorów porównania, logicznych i tablicowych.
6. Poznanie potoków agregacji (Aggregation Pipeline) i umiejętność ich stosowania w analizie danych.
7. Ćwiczenie praktycznego tworzenia zapytań z wykorzystaniem operatorów \$match, \$group, \$project, \$lookup, \$sort, \$limit, \$unwind.
8. Zrozumienie zasad projektowania kolekcji – zagnieżdżania (embedding) i odwołań (referencing) – oraz ich wpływu na wydajność i spójność danych.
9. Praktyczne zapoznanie z różnicami między denormalizacją a normalizacją danych w środowisku dokumentowym.
10. Poznanie dobrych praktyk w zakresie indeksowania, projektowania struktur danych i optymalizacji zapytań w MongoDB.
11. Zrozumienie zastosowań innych baz dokumentowych, takich jak RethinkDB i Firebase Firestore, oraz ich roli w aplikacjach czasu rzeczywistego i środowiskach chmurowych.
12. Ukształtowanie umiejętności wyboru odpowiedniego modelu danych w zależności od wymagań systemu i charakteru danych.

Instrukcja laboratoryjna/ćwiczeniowa

Dokumentowe bazy NoSQL

Bazy dokumentowe stanowią jedną z najważniejszych gałęzi systemów NoSQL i są projektowane w odpowiedzi na ograniczenia klasycznych relacyjnych baz danych w środowiskach o dużej zmienności struktury danych, skalowalności horyzontalnej oraz wymogach pracy w czasie rzeczywistym. Ich podstawową cechą jest przechowywanie informacji w postaci dokumentów, zazwyczaj w formacie JSON lub jego binarnej reprezentacji BSON, co pozwala na naturalne odwzorowanie danych aplikacyjnych w strukturze odpowiadającej obiektom znanym z programowania. Każdy dokument jest niezależną jednostką zawierającą komplet informacji, nie wymaga odwoływania się do innych tabel czy relacji, co eliminuje konieczność wykonywania złożonych operacji łączenia (JOIN) charakterystycznych dla systemów relacyjnych.

Model dokumentowy stanowi kompromis między elastycznością a spójnością. W odróżnieniu od relacyjnego podejścia, w którym dane są ściśle znormalizowane i powiązane, bazy dokumentowe kładą nacisk na swobodę kształtowania struktury



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



danych i szybki dostęp. Dzięki temu są fundamentem nowoczesnych systemów informatycznych, w których adaptacyjność, skalowalność i czas reakcji są ważniejsze niż bezwzględna kontrola nad relacjami między rekordami. Model dokumentowy zakłada elastyczność schematu (tzw. schema-less lub schema-flexible), dzięki czemu kolejne dokumenty w tej samej kolekcji mogą różnić się zestawem pól, typami wartości, a nawet głębokością zagnieżdżeń. Taka konstrukcja szczególnie dobrze sprawdza się w środowiskach, gdzie dane pochodzą z dynamicznych źródeł – np. z aplikacji webowych, czujników IoT, platform e-commerce czy systemów analitycznych, w których struktura informacji może ewoluować w czasie. Elastyczność modelu dokumentowego sprzyja szybkiemu prototypowaniu, integracji z nowymi źródłami danych i iteracyjnemu rozwojowi aplikacji bez konieczności migracji schematów.

W systemach dokumentowych, takich jak MongoDB, CouchDB czy RethinkDB, dane grupowane są w kolekcje będące odpowiednikami tabel, a zapytania wykonywane są przy użyciu języków inspirowanych JSON-em. W przypadku MongoDB operacje realizowane są w postaci zagnieżdżonych struktur opisujących warunki, projekcje i agregacje. Kluczową rolę odgrywa tu potok agregacji (aggregation pipeline), który pozwala przetwarzać dokumenty w wielu etapach: filtrować, grupować, sortować, przekształcać i analizować, w sposób przypominający przepływ danych w systemach przetwarzania strumieniowego. Taki model operowania na danych umożliwia wykonywanie złożonych analiz bez konieczności przenoszenia danych do zewnętrznych narzędzi, co znacząco redukuje opóźnienia i obciążenie sieci.

Bazy dokumentowe charakteryzują się także wysoką skalowalnością horyzontalną, co oznacza, że można je rozbudowywać przez dodawanie kolejnych węzłów (ang. sharding), a nie tylko poprzez zwiększanie mocy pojedynczej maszyny. Dzięki temu możliwe jest przetwarzanie dużych wolumenów danych w rozproszonych środowiskach, takich jak klastry serwerów czy chmury obliczeniowe. Wbudowane mechanizmy replikacji i rozproszonego zapisu zapewniają odporność na awarie oraz równoległy dostęp wielu użytkowników. W przeciwieństwie do tradycyjnych systemów transakcyjnych (ACID), wiele implementacji baz dokumentowych realizuje model spójności opartej na zasadzie eventual consistency, co oznacza, że dane w różnych węzłach systemu mogą być chwilowo niespójne, lecz z czasem osiągają zgodność. Takie podejście pozwala uzyskać większą dostępność i odporność na opóźnienia w środowiskach sieciowych o dużej skali.

Zaletą baz dokumentowych jest ich naturalne dopasowanie do współczesnych modeli aplikacyjnych, zwłaszcza w architekturze mikroserwisowej, gdzie każdy komponent może korzystać z własnego zestawu danych w odpowiednim dla siebie



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



formacie. Umożliwia to redukcję kosztów integracji oraz łatwiejsze odwzorowanie danych biznesowych na poziomie kodu aplikacji. Ponadto struktura dokumentowa pozwala przechowywać dane heterogeniczne – tekstowe, liczbowo-analityczne, a nawet binarne (np. obrazy w postaci osadzonych obiektów).

Jednocześnie bazy dokumentowe nie są pozbawione ograniczeń. Brak sztywnego schematu utrudnia egzekwowanie spójności danych oraz może prowadzić do rozbieżności strukturalnych przy niekontrolowanym rozwoju aplikacji. Operacje wymagające korelacji między różnymi kolekcjami (odpowiednikiem relacji w SQL) bywają bardziej kosztowne i trudniejsze do optymalizacji, ponieważ model dokumentowy preferuje denormalizację – czyli powielanie danych w różnych dokumentach – w celu poprawy szybkości odczytu. W efekcie aktualizacja powtarzających się informacji wymaga mechanizmów aplikacyjnych, a nie wbudowanych relacji.

Bazy dokumentowe są zatem wybierane tam, gdzie elastyczność i wydajność odczytu mają wyższy priorytet niż ścisła integralność danych. Sprawdzają się szczególnie w aplikacjach o dużym wolumenie nieustrukturyzowanych informacji, takich jak logi systemowe, dane telemetryczne, profile użytkowników, systemy rekomendacji, analityka w czasie rzeczywistym czy aplikacje mobilne i webowe, które muszą szybko reagować na zmieniające się dane. W porównaniu z klasycznymi bazami SQL, systemy dokumentowe wygrywają w środowiskach o dużej dynamice danych, rozproszonych architekturach i konieczności pracy w trybie online.

System MongoDB

MongoDB stanowi najbardziej rozpowszechnioną implementację modelu dokumentowego, łączącą prostotę pracy z danymi w formacie JSON z wysoką skalowalnością i elastycznością architektury. Jej projekt zakładał odejście od sztywnych schematów typowych dla SQL i umożliwienie dynamicznego modelowania danych w sposób zbliżony do ich rzeczywistej struktury w aplikacjach. Dokumenty w MongoDB są zapisywane w formacie BSON (Binary JSON), który stanowi rozszerzenie klasycznego JSON-a o typy binarne, daty, liczby całkowite o różnych rozmiarach, a nawet wskaźniki do innych dokumentów. Dzięki temu możliwe jest efektywne przechowywanie i przetwarzanie dużych wolumenów danych o zróżnicowanej naturze, od metadanych po wyniki analityczne i dane telemetryczne.

MongoDB należy rozpatrywać nie tylko jako bazę danych, ale jako kompletną platformę przetwarzania informacji, zdolną obsługiwać dane transakcyjne, analityczne i strumieniowe w jednym środowisku. Jej architektura łączy cechy



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



systemu OLTP z elementami analizy danych w stylu OLAP, dzięki czemu stanowi fundament wielu współczesnych systemów e-commerce, portali społecznościowych, aplikacji mobilnych czy systemów IoT. W praktyce jej przewaga nie wynika jedynie z wydajności, ale z adaptacyjności, z umiejętności dostosowania się do dynamicznego charakteru danych i aplikacji, które te dane generują.

Jednym z filarów MongoDB jest indeksowanie. System oferuje szeroki wachlarz typów indeksów, od klasycznych jednopolowych, przez złożone (compound), po indeksy tekstowe, geolokalizacyjne i wielokluczowe (multikey), które umożliwiają szybkie wyszukiwanie w tablicach. Dobrze zaprojektowany zestaw indeksów ma kluczowe znaczenie dla wydajności, zwłaszcza przy dużych kolekcjach. MongoDB posiada też mechanizm tzw. partial i sparse indexes, pozwalający na indeksowanie tylko wybranych dokumentów lub pól, co jest istotne przy danych o nieregularnej strukturze. Dzięki temu można ograniczyć rozmiar indeksów i poprawić efektywność operacji odczytu.

Kolejnym aspektem wyróżniającym MongoDB jest replikacja i wysoka dostępność. Baza operuje w modelu tzw. replica set, czyli zestawu serwerów, w którym jeden pełni rolę głównego węzła (primary), a pozostałe stanowią jego kopie zapasowe (secondaries). W przypadku awarii system automatycznie przeprowadza wybór nowego węzła głównego (automatic failover), co gwarantuje ciągłość pracy aplikacji. Replikacja może być również wykorzystywana do odciążania serwera głównego – zapytania tylko do odczytu mogą być kierowane do węzłów podrzędnych.

MongoDB wspiera również sharding, czyli partycjonowanie danych na wiele węzłów. Mechanizm ten rozdziela dokumenty między serwery na podstawie wybranego klucza shardującego. W praktyce pozwala to na niemal liniową skalowalność w poziomie – przy rosnącej liczbie danych wystarczy dodawać kolejne maszyny zamiast zwiększać moc jednej. Jednak wybór odpowiedniego klucza shardującego jest decyzją architektoniczną o dużym znaczeniu: zbyt nierównomierny rozkład wartości może prowadzić do tzw. hot spots, czyli przeciążenia wybranych shardów.

MongoDB zapewnia również mechanizmy transakcyjne, które wprowadzono od wersji 4.0. Choć pierwotnie operacje w MongoDB były atomowe tylko na poziomie pojedynczego dokumentu, obecnie można wykonywać transakcje obejmujące wiele dokumentów i kolekcji. Pozwala to na realizację złożonych scenariuszy biznesowych przy zachowaniu pełnej spójności danych. Warto jednak pamiętać, że transakcje te zwiększają koszty operacyjne i mogą ograniczyć przepustowość przy dużych wolumenach danych – dlatego stosuje się je głównie w sytuacjach wymagających ścisłej integralności logicznej.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Z punktu widzenia analizy danych MongoDB dysponuje rozbudowanym silnikiem agregacji opartym na koncepcji potoków (aggregation pipelines). Pozwala on łączyć wiele etapów przetwarzania: filtrowania, grupowania, sortowania, projekcji, mapowania i transformacji w jeden spójny proces. W praktyce pełni on rolę zbliżoną do podzapytań i funkcji agregujących w SQL, jednak jego elastyczność jest znacznie większa. Dodatkowo MongoDB oferuje integrację z językami analitycznymi i platformami przetwarzania danych, w tym z Pythonem (biblioteka pymongo), środowiskiem Spark czy frameworkami uczenia maszynowego.

Istotnym elementem architektury MongoDB jest mechanizm walidacji schematów. Choć baza pozostaje systemem schemaless, umożliwia definiowanie reguł kontrolujących poprawność dokumentów na poziomie kolekcji. Administrator może wymusić, aby określone pola istniały, miały konkretny typ lub spełniały warunki logiczne. Takie rozwiązanie łączy elastyczność NoSQL z częściową formalizacją znaną z relacyjnych baz danych, pozwalając ograniczyć ryzyko wprowadzenia niepoprawnych danych.

MongoDB wyróżnia się również bogatym ekosystemem narzędzi administracyjnych. Narzędzie mongosh stanowi interaktywną powłokę umożliwiającą pracę z dokumentami, natomiast MongoDB Compass oferuje graficzny interfejs do eksploracji danych, wizualizacji zapytań i analizy indeksów. W środowiskach produkcyjnych wykorzystuje się także platformę Atlas, zapewniającą zarządzanie klastrami w chmurze, automatyczne kopie zapasowe, monitorowanie i mechanizmy bezpieczeństwa oparte na szyfrowaniu w spoczynku i w transmisji.

Na szczególną uwagę zasługuje podejście MongoDB do modelowania danych. W przeciwieństwie do relacyjnych baz, gdzie nacisk kładzie się na normalizację, w MongoDB często preferuje się modelowanie z użyciem zagnieżdżonych dokumentów i tablic, co eliminuje konieczność łączenia wielu kolekcji. Wybór między zagnieżdżeniem (embedding) a referencją (referencing) stanowi kluczową decyzję projektową. Zagnieżdżenie poprawia wydajność odczytu, natomiast referencje zwiększają spójność danych przy dużej redundancji. W praktyce stosuje się podejścia hybrydowe, łączące obie strategie w zależności od charakteru aplikacji.

Z perspektywy bezpieczeństwa MongoDB zapewnia autoryzację opartą na rolach (RBAC) oraz mechanizmy uwierzytelniania oparte o SCRAM, LDAP lub X.509. System pozwala także na rejestrowanie operacji administracyjnych, co ułatwia audyt i detekcję nieautoryzowanych działań. W nowoczesnych wdrożeniach szczególną rolę odgrywa szyfrowanie komunikacji (TLS) oraz izolacja danych użytkowników w środowiskach wielodomenowych.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



System RethinkDB

RethinkDB jest dokumentową bazą danych o odmiennej filozofii działania niż MongoDB, została zaprojektowana z myślą o natychmiastowej reakcji na zmiany danych, oraz o integracji z aplikacjami czasu rzeczywistego. Jej kluczowym wyróżnikiem jest mechanizm realtime push, dzięki któremu aplikacja kliencka nie musi cyklicznie odpytywać bazy o nowe dane. Zamiast tego RethinkDB utrzymuje otwarte połączenie i przesyła zmiany do klienta w momencie ich wystąpienia. To podejście czyni ją szczególnie atrakcyjną w zastosowaniach takich jak systemy monitorowania, tablice wyników, aplikacje kolaboracyjne czy dashboardy biznesowe, gdzie kluczowa jest aktualność informacji. RethinkDB należy traktować jako bazę reaktywną, a nie wyłącznie dokumentową. Wygrywa w zastosowaniach, gdzie najważniejsza jest aktualność informacji, prostota tworzenia strumieni danych i bezpośrednia komunikacja z warstwą aplikacji. W systemach, które stawiają na niskie opóźnienia i ciągłą synchronizację między użytkownikami, RethinkDB zapewnia funkcjonalność, której w MongoDB i innych bazach NoSQL trzeba dopiero zbudować za pomocą dodatkowych komponentów.

W przeciwieństwie do MongoDB, które koncentruje się na prostocie zapisu i skalowalności, RethinkDB stawia nacisk na reaktywność i spójność logiczną danych. Oferuje natywny język zapytań ReQL, przypominający składnią języki funkcyjne, co pozwala tworzyć złożone operacje mapowania, filtrowania i agregacji bezpośrednio w logice bazy. Operacje te są deklaratywne i strumieniowe, dzięki czemu mogą być przetwarzane sekwencyjnie bez konieczności ładowania całego zbioru danych do pamięci. W odróżnieniu od MongoDB, które operuje na dokumentach w formacie BSON, RethinkDB przechowuje dane w czystym formacie JSON, co ułatwia ich integrację z aplikacjami webowymi i językami skryptowymi. Pod względem architektury RethinkDB jest bazą rozproszoną z natywnym wsparciem dla replikacji i sharding-u, jednak jej celem nie jest obsługa setek terabajtów danych, lecz zapewnienie niskich opóźnień w komunikacji między warstwą danych a aplikacją kliencką. Silnik bazy implementuje transakcje oparte na zasadzie write-durability i zapewnia spójność odczytu w ramach pojedynczej repliki, co pozwala uniknąć typowych problemów eventual consistency spotykanych w innych systemach NoSQL.

Z punktu widzenia projektanta systemów, RethinkDB zyskuje przewagę tam, gdzie dane muszą być natychmiast propagowane do użytkownika, np. w czatach, systemach powiadomień, giełdach danych, grach online, analityce w czasie rzeczywistym czy panelach administracyjnych, które dynamicznie odświeżają wyniki. MongoDB, choć szybsze przy klasycznych operacjach odczytu i zapisu, wymaga dodatkowych narzędzi (np. Change Streams) do osiągnięcia podobnej



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



funkcjonalności, podczas gdy RethinkDB oferuje ją natywnie. Zaletą RethinkDB jest również przejrzysty model replikacji i kontroli klastrów – każdy węzeł może pełnić różne role (primary, secondary), a system automatycznie równoważy obciążenie i synchronizuje dane między replikami. Interfejs administracyjny bazy, dostępny w przeglądarce, pozwala wizualizować przepływ danych w czasie rzeczywistym, analizować zapytania i monitorować zużycie zasobów, co znacząco ułatwia utrzymanie środowisk developerskich i produkcyjnych. W kontekście ograniczeń należy zauważyć, że RethinkDB nie dorównuje MongoDB pod względem popularności, ekosystemu i liczby integracji. Projekt, choć otwartoźródłowy, przez pewien czas był w stagnacji po zakończeniu komercyjnego wsparcia, a jego rozwój kontynuowany jest obecnie przez społeczność. Nie stanowi jednak alternatywy uniwersalnej, jest to baza specjalizowana, przeznaczona do systemów wymagających natychmiastowej propagacji danych i prostoty programistycznej, a nie do hurtowni danych czy długoterminowego przechowywania miliardów rekordów.

Usługa Cloud Firestore

Cloud Firestore, będąca częścią ekosystemu Firebase rozwijanego przez Google, reprezentuje nowoczesne podejście do baz dokumentowych zaprojektowanych z myślą o środowiskach rozproszonych, mobilnych i webowych. W odróżnieniu od klasycznych baz NoSQL, które koncentrują się na lokalnej wydajności i elastycznym modelu danych, Firestore jest bazą natywnie chmurową, zarządzaną, skalowaną i replikowaną automatycznie w infrastrukturze Google Cloud, co eliminuje potrzebę administracji serwerami i konfiguracji klastrów. Pod względem modelu danych Firestore wykorzystuje hierarchiczną strukturę kolekcji i dokumentów, w której każda kolekcja może zawierać wiele dokumentów, a każdy dokument może mieć swoje podkolekcje. To rozwiązanie łączy prostotę modelu JSON znaną z MongoDB z logiczną strukturą przypominającą system plików. Dzięki temu Firestore umożliwia intuicyjne odwzorowanie hierarchii danych aplikacji, np. użytkownicy → zamówienia → szczegóły pozycji, bez potrzeby stosowania relacji lub dodatkowych warstw mapujących.

Jednym z kluczowych wyróżników Firestore jest jego asynchroniczny model synchronizacji w czasie rzeczywistym. Dane zapisane w bazie są natychmiast propagowane do wszystkich połączonych klientów, zarówno aplikacji webowych, jak i mobilnych. Klient SDK utrzymuje lokalny bufor (cache) danych i automatycznie synchronizuje go z chmurą przy każdej zmianie po stronie serwera lub klienta. Ten mechanizm sprawia, że aplikacje oparte na Firestore działają nawet w trybie offline, a po przywróceniu połączenia zmiany są automatycznie scalane (tzw. offline persistence with conflict resolution). W praktyce pozwala to budować aplikacje o



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



zachowaniu „real-time” bez konieczności implementacji własnych serwerów push czy kolejek komunikacyjnych.

Firestore działa w oparciu o rozproszony model spójności o wysokiej dostępności (multi-region replication). Dane przechowywane są równolegle w kilku centrach danych, co gwarantuje odporność na awarie i minimalne opóźnienia globalne. System zapewnia silną spójność odczytów (strong consistency) w obrębie dokumentu, co odróżnia go od wielu innych baz NoSQL, które oferują jedynie spójność ostateczną (eventual consistency). Dla aplikacji użytkowych oznacza to, że dane widoczne po zapisie są natychmiast aktualne, bez ryzyka odczytu niesynchronizowanej wersji. Z punktu widzenia programisty Firestore integruje się bezpośrednio z aplikacjami poprzez API SDK dostępne w popularnych językach (JavaScript, Python, Kotlin, Swift, Go). Zapytania mają charakter deklaratywny i opierają się na filtrach, zakresach i indeksach, wszystkie operacje są indeksowane automatycznie, a dla zapytań złożonych użytkownik może definiować indeksy złożone (composite indexes). W przeciwieństwie do MongoDB, Firestore nie pozwala na dowolne zapytania agregacyjne czy pełne przetwarzanie danych po stronie serwera, jednak jego siłą jest prostota, deterministyczne działanie i minimalne opóźnienia w komunikacji. W kontekście bezpieczeństwa i autoryzacji Firestore wykorzystuje model Firebase Security Rules, który łączy warstwę danych z logiką uwierzytelniania. Reguły bezpieczeństwa definiowane są w języku deklaratywnym i opisują, kto może odczytywać, zapisywać lub modyfikować dane na poziomie kolekcji i dokumentów. Można w nich odwoływać się do kontekstu użytkownika (np. `request.auth.uid`) oraz wartości samych dokumentów, co pozwala budować złożone, dynamiczne polityki bezpieczeństwa bez pośrednictwa backendu. To rozwiązanie czyni Firestore szczególnie atrakcyjnym dla projektów typu serverless, gdzie logika biznesowa znajduje się po stronie klienta, a kontrola dostępu w samej bazie.

Nie jest to jednak baza uniwersalna. Firestore ma pewne ograniczenia, takie jak limit głębokości hierarchii, brak klasycznych transakcji wieloetapowych w stylu SQL oraz koszty zależne od liczby operacji, a nie od rozmiaru danych. W projektach wymagających rozbudowanej analityki, przetwarzania masowych zapytań lub integracji z danymi relacyjnymi, lepiej sprawdzają się narzędzia takie jak MongoDB.

Projektowanie kolekcji w MongoDB



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Projektowanie kolekcji w MongoDB jest jednym z kluczowych etapów budowy efektywnego systemu opartego na bazie dokumentowej. W przeciwieństwie do relacyjnych baz danych, w których struktura jest sztywno określona przez schemat tabel, w MongoDB projektowanie polega na doborze odpowiedniego modelu dokumentów i relacji między nimi — czyli świadomym wyborze pomiędzy zagnieżdżaniem danych (embedding) a odwołaniami (referencing). Oba podejścia mogą współistnieć, a właściwe ich zastosowanie decyduje o wydajności, skalowalności i prostocie utrzymania systemu.

Model zagnieżdżania (Embedding)

Zagnieżdżanie polega na umieszczaniu powiązanych danych w jednym dokumencie JSON. Przykładowo, zamiast tworzyć osobną kolekcję dla adresów użytkowników, można przechowywać je wewnątrz dokumentu użytkownika:

```
{
  _id: 1,
  name: "Anna Kowalska",
  email: "anna.kowalska@example.com",
  address: {
    city: "Kraków",
    street: "Długa 5",
    postalCode: "31-123"
  }
}
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Taki model zapewnia lokalność danych – wszystkie informacje potrzebne do obsługi użytkownika znajdują się w jednym miejscu, dzięki czemu odczyt jest szybki, a zapytania nie wymagają łączenia wielu kolekcji. To rozwiązanie idealne w przypadkach, gdy dane są ściśle powiązane semantycznie i często pobierane razem. Zagnieżdżanie minimalizuje liczbę zapytań do bazy, co znacząco poprawia wydajność w aplikacjach odczytowo-intensywnych. Jednak zbyt głębokie zagnieżdżenie może prowadzić do nadmiernego rozrostu dokumentów, co utrudnia ich aktualizację. MongoDB ma ograniczenie rozmiaru pojedynczego dokumentu (16 MB), dlatego zagnieżdżanie dużych struktur, takich jak historia transakcji lub setki komentarzy, nie jest zalecane. Aktualizacje w strukturze zagnieżdżonej są kosztowne – każda zmiana wymaga modyfikacji całego dokumentu, co może generować blokady i spadek wydajności przy dużym współczynniku zapisu. W takich przypadkach bardziej elastyczne okazuje się podejście oparte na odwołaniach.

Model odwołań (Referencing)

Odwołania (referencje) polegają na łączeniu dokumentów z różnych kolekcji za pomocą identyfikatorów, zamiast umieszczania danych bezpośrednio w sobie. Na przykład użytkownik może mieć pole `address_id`, które wskazuje na dokument w kolekcji `addresses`:

```
{
  _id: 1,
  name: "Anna Kowalska",
  email: "anna.kowalska@example.com",
  address_id: ObjectId("64bce2c9f1a3a0d9c2a3d2e1")
}
```

a odpowiadający adres może znajdować się w osobnej kolekcji:

```
{
  _id: ObjectId("64bce2c9f1a3a0d9c2a3d2e1"),
  city: "Kraków",
}
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



street: "Długa 5",

postalCode: "31-123"

}

To podejście przypomina relacje w klasycznym modelu SQL, ale bez sztywnego wymuszania integralności referencyjnej. Odwołania są korzystne, gdy dane są współdzielone między wieloma dokumentami lub gdy ich rozmiar i liczba rosną niezależnie. Ułatwiają aktualizacje – zmiana jednego adresu automatycznie odnosi się do wszystkich użytkowników, którzy się do niego odwołują.

Z drugiej strony, model referencyjny wymaga łączenia danych na poziomie aplikacji lub potoków agregacji, np. za pomocą operatora \$lookup. Wpływa to na wydajność, ponieważ każde łączenie wymaga dodatkowego przetwarzania po stronie serwera i może generować narzut przy dużej liczbie dokumentów. Dlatego nie powinno się nadużywać tego podejścia w przypadkach, gdy dane są zawsze pobierane razem.

Zagnieżdżanie kontra odwołania — wybór strategii

Decyzja, czy użyć zagnieżdżania, czy odwołań, zależy od charakteru powiązań między danymi.

Zagnieżdżanie preferuje się, gdy:

- dane są ściśle powiązane i współwystępują w zapytaniach,
- rozmiar dokumentu pozostaje niewielki,
- dane są niemodyfikowalne lub rzadko aktualizowane,
- priorytetem jest szybkość odczytu.

Odwołania stosuje się, gdy:

- dane są współdzielone między wieloma dokumentami,
- ich liczba lub rozmiar jest dynamiczny,
- występuje potrzeba aktualizacji niezależnych fragmentów,
- zależy nam na modularności i skalowalności danych.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



Często w praktyce wykorzystuje się podejście hybrydowe, w którym dane główne są zagnieżdżone, a bardziej szczegółowe, często zmieniające się informacje – odwoływane. Przykładowo, w systemie e-commerce dokument zamówienia może zawierać wbudowaną listę produktów (z nazwą i ceną), ale odwoływać się do osobnej kolekcji customers, która przechowuje pełne dane klienta. Pozwala to zachować balans między wydajnością odczytu a elastycznością modyfikacji.

Dobre praktyki projektowania kolekcji

Efektywne projektowanie w MongoDB wymaga myślenia od strony zapytań, a nie struktury danych. Oznacza to, że najpierw należy przeanalizować, jakie operacje będą wykonywane najczęściej, a następnie dostosować model dokumentów do tych przypadków użycia. MongoDB premiuje modelowanie pod kątem odczytu, nawet kosztem pewnej redundancji danych — ponieważ odczyty są znacznie częstsze niż zapisy.

Kluczowe zalecenia obejmują:

- Projektuj pod kątem zapytań, nie odwrotnie. Dane, które są często pobierane razem, powinny znajdować się w jednym dokumencie.
- Unikaj nadmiernej głębokości zagnieżdżenia. MongoDB ma limit 100 poziomów zagnieżdżenia, ale w praktyce zaleca się nie więcej niż kilka.
- Kontroluj rozmiar dokumentów. Duże dokumenty mogą powodować fragmentację i spowolnienia operacji I/O.
- Stosuj indeksy z rozwagą. Tworzenie indeksów na polach często używanych w zapytaniach znacznie przyspiesza ich działanie, ale zbyt wiele indeksów obniża wydajność zapisów.
- Denormalizuj dane tam, gdzie to uzasadnione. MongoDB dobrze znosi powielanie informacji, o ile przyspiesza to odczyt.
- Używaj agregacji zamiast nadmiarowych zapytań. Potoki agregacyjne mogą zastąpić złożone operacje łączenia i przetwarzania w warstwie aplikacji.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



- Zachowuj spójność logiczną. W przypadku danych krytycznych warto rozważyć użycie transakcji ACID, które MongoDB obsługuje w ramach dokumentów i wielu kolekcji w tym samym klastrze.

W ujęciu inżynierskim projektowanie kolekcji w MongoDB to proces równoważenia trzech aspektów: wydajności, spójności i prostoty zapytań. Zagnieżdżanie zapewnia szybkość i lokalność, ale ogranicza elastyczność; odwołania oferują modularność i skalowalność, kosztem większego narzutu zapytań. W praktyce najlepsze rozwiązania powstają wtedy, gdy architekt traktuje model danych nie jako strukturę statyczną, lecz jako element dostosowany do logiki systemu i jego dynamiki – elastyczny, czytelny i skalowalny w czasie. MongoDB umożliwia pełną swobodę w tym zakresie, dlatego umiejętność właściwego wyboru strategii modelowania stanowi o jakości i długowieczności całego projektu.

Zapytania w MongoDB

W MongoDB zapytania stanowią jeden z najważniejszych elementów pracy z danymi i jednocześnie klucz do pełnego wykorzystania potencjału tego systemu. W przeciwieństwie do języka SQL, MongoDB operuje w sposób dokumentowy, co oznacza, że zapytania są formułowane w postaci obiektów JSON opisujących warunki, operatory i projekcje danych. Dzięki temu praca z danymi staje się bardziej naturalna z punktu widzenia programisty, zwłaszcza w aplikacjach opartych na strukturach zagnieżdżonych i dynamicznych.

Podstawą każdego zapytania jest polecenie `find()`, które pełni rolę odpowiednika `SELECT` z języka SQL. Jego składnia pozwala jednocześnie określić filtr, projekcję i sposób sortowania. Na przykład zapytanie:

```
db.students.find({ age: { $gt: 20 } }, { name: 1, age: 1, _id: 0 }).sort({ age: -1 })
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



zwróci wszystkie dokumenty, w których pole age jest większe niż 20, wyświetlając jedynie imię i wiek, posortowane malejąco po wieku. W tym miejscu warto zauważyć, że MongoDB operuje na operatorach porównania (\$eq, \$ne, \$gt, \$lt, \$gte, \$lte), operatorach logicznych (\$and, \$or, \$not, \$nor) oraz operatorach tablicowych (\$in, \$nin, \$all, \$elemMatch), co daje bardzo dużą swobodę w formułowaniu złożonych warunków filtrujących. Z kolei projekcja określa, które pola mają zostać zwrócone, co w praktyce ogranicza rozmiar przesyłanych danych i przyspiesza odczyt.

MongoDB udostępnia również zaawansowany mechanizm przetwarzania danych w potokach, zwany Aggregation Pipeline. To potężne narzędzie pozwala wykonywać wieloetapowe operacje na danych, w których wynik jednego etapu staje się wejściem dla kolejnego, w sposób analogiczny do potoków w systemach uniksowych. Potok agregacji składa się z szeregu etapów zdefiniowanych za pomocą operatorów takich jak \$match, \$group, \$project, \$sort, \$limit, \$skip, \$unwind czy \$lookup. Każdy etap realizuje określone zadanie — filtrowanie, grupowanie, transformację, łączenie danych lub tworzenie nowych pól.

Etap \$match odpowiada za filtrowanie danych na wzór klauzuli WHERE w SQL. Przykładowo:

```
{ $match: { status: "active" } }
```

wybiera tylko dokumenty o określonym stanie. Kolejnym etapem może być \$group, który pełni funkcję zbliżoną do GROUP BY. Umożliwia grupowanie dokumentów według określonego pola i obliczanie agregatów, takich jak suma, średnia czy liczność. Przykładowa definicja:

```
{ $group: { _id: "$category", total: { $sum: "$price" }, avgPrice: { $avg: "$price" } } }
```



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



pozwała uzyskać łączną i średnią wartość produktów w każdej kategorii. Etap \$project z kolei służy do przekształcania struktury danych, wybierania konkretnych pól lub tworzenia nowych, np. poprzez wyrażenia arytmetyczne lub logiczne. W połączeniu z \$sort, \$limit i \$skip można kontrolować kolejność i liczbę zwracanych dokumentów, co jest szczególnie przydatne w paginacji wyników.

Jednym z najbardziej interesujących elementów agregacji jest operator \$unwind, który rozbija tablice na pojedyncze dokumenty, co pozwala analizować dane złożone. Z kolei \$lookup pełni rolę łączenia kolekcji (odpowiednik JOIN), umożliwiając pobieranie powiązanych informacji bez konieczności ręcznego wykonywania wielu zapytań. Przykładowe użycie:

```
{  
  $lookup: {  
    from: "departments",  
    localField: "dept_id",  
    foreignField: "_id",  
    as: "department_info"  
  }  
}
```

łącząc kolekcję employees z departments na podstawie identyfikatora działu. MongoDB wspiera także tzw. pipeline operators w ramach \$lookup, które umożliwiają wprowadzenie dodatkowych etapów filtrujących lub projekcyjnych już podczas łączenia danych, co znacząco zwiększa elastyczność zapytań.

W potokach agregacji można wykorzystywać również wyrażenia obliczeniowe i warunki logiczne, takie jak \$cond, \$ifNull, \$add, \$multiply, \$concat, \$toUpper czy



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



\$dateToString. Dzięki temu możliwe jest dynamiczne tworzenie pól, konwersja typów i przekształcanie wartości bezpośrednio po stronie serwera, bez potrzeby ich obróbki w aplikacji. MongoDB traktuje potoki agregacyjne jako mechanizm przetwarzania danych bliski modelowi map-reduce, lecz znacznie bardziej czytelny i wydajny.

Istotnym uzupełnieniem potoków są operatory wyszukiwania tekstowego (\$text, \$search), umożliwiające tworzenie indeksów pełnotekstowych i zapytań opartych na dopasowaniu słów kluczowych. W połączeniu z operatorami wyrażeń regularnych (\$regex) pozwalają one budować wyszukiwarki i systemy rekomendacji. Dodatkowo MongoDB umożliwia stosowanie filtrów geolokalizacyjnych (\$geoWithin, \$near, \$geoIntersects) do zapytań przestrzennych, wykorzystywanych w aplikacjach mapowych i IoT.

W praktyce zapytania w MongoDB można traktować jako kombinację trzech głównych komponentów: filtrowania danych (\$match, warunki w find()), projekcji i transformacji (\$project, find(..., projection)), oraz agregacji i obliczeń (\$group, \$sum, \$avg, \$count). Ich składanie w potoki tworzy logiczne, deterministyczne i w pełni równoległe przepływy danych, które są wykonywane w obrębie serwera, minimalizując transfer i obciążenie po stronie klienta.

Ćwiczenia samodzielne

Do wykonania zadań użyj kolekcji z datasetu sample_training dostępnych pod linkiem:

https://github.com/neelabalan/mongodb-sample-dataset/tree/main/sample_training

Analiza kolekcji stories

1. Wyświetl tytuły dokumentów, w których liczba komentarzy przekracza 100, a status to „popular”.
2. Policz średnią wartość pola diggs wśród wpisów, w których media to „news”.
3. Dla każdej wartości pola container.name oblicz sumę komentarzy i posortuj wynik malejąco.



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



4. Wskaż użytkownika (user.name), który ma najwyższą sumaryczną liczbę profileviews.
5. Zapisz wynik zapytania z pkt. 7 do nowej kolekcji top_users przy pomocy operatora \$out.
6. Użyj wyrażenia regularnego, aby znaleźć dokumenty, w których user.name zaczyna się literą „M” i posiada więcej niż 50 komentarzy.

Analiza kolekcji sales

7. Znajdź wszystkie transakcje, w których przynajmniej jeden produkt zawiera w polu tags wartość „school”.
8. Wyświetl adresy e-mail klientów oraz datę zakupu w formacie DD-MM-YYYY.
9. Oblicz średnią wartość zakupu (total), grupując po polu storeLocation.
10. Znajdź klienta, który dokonał największej liczby transakcji metodą „Online”.
11. Dla każdego sklepu (storeLocation) oblicz sumaryczną sprzedaż tylko dla klientów z oceną customer.satisfaction > 4.
12. Policz, ile transakcji zawierało dokładnie cztery elementy w tablicy items.

Analiza kolekcji trips

13. Oblicz średni czas trwania wycieczek (tripduration) dla każdej stacji startowej i posortuj wynik malejąco.
14. Znajdź dni tygodnia z największą liczbą rozpoczętych podróży.
15. Policz liczbę podróży w zależności od typu użytkownika (usertype).
16. Oblicz średnią długość podróży tylko dla subskrybentów (usertype: "Subscriber").
17. Znajdź wszystkie wycieczki, które rozpoczęły się w stacjach zawierających w nazwie słowo „Centre”.
18. Oblicz średnią długość podróży w zależności od godziny rozpoczęcia (\$hour: "\$start time").
19. Określ, która stacja startowa jest najczęściej używana wśród użytkowników płci „2”.

Agregacja między kolekcjami

20. Wykorzystując \$lookup, połącz dane z kolekcji sales i customers (jeśli występują) na podstawie adresu e-mail klienta.
21. Utwórz agregację, która dla każdego klienta pokaże liczbę transakcji oraz łączną wartość zakupów.
22. Zapisz wynik do kolekcji customer_stats i wyświetl pięć pierwszych rekordów.

Zadania rozszerzone



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



23. Wykorzystaj operator \$facet, aby jednocześnie obliczyć:
24. średnią liczbę komentarzy w stories,
25. średnią długość wycieczki w trips,
26. średnią wartość sprzedaży w sales.
27. Za pomocą \$merge zapisz wynik z pkt. 26 do nowej kolekcji summary_metrics.
28. Zastosuj \$bucket w kolekcji sales, aby pogrupować transakcje według wartości total w przedziałach: <50, 50–200, >200.
29. Utwórz zapytanie wykorzystujące \$regex i \$expr, które wyświetli wszystkie produkty (items.name), których nazwa zawiera słowo „Book” i których ilość (items.quantity) przekracza 5.
30. Przygotuj raport łączący dane z sales i stores, w którym pokażesz lokalizacje z najwyższym średnim wynikiem customer.satisfaction.

Pytania pomocnicze

1. Czym różni się model danych w bazach dokumentowych od modelu relacyjnego?
2. W jakim formacie przechowywane są dane w MongoDB i czym jest BSON?
3. Na czym polega różnica między kolekcją a tabelą w klasycznej bazie SQL?
4. Jakie operacje składają się na cykl CRUD i jak są realizowane w MongoDB?
5. Jak działa polecenie find() i w jaki sposób można stosować filtry z operatorami \$gt, \$lt, \$and, \$or?
6. Do czego służy potok agregacji i jakie etapy może obejmować (\$match, \$group, \$project, \$lookup, \$sort)?
7. Jakie są zalety stosowania operatora \$lookup w porównaniu z tradycyjnym zagnieżdżaniem danych?
8. Czym różni się zagnieżdżanie danych (embedding) od odwołań (referencing)?
9. Kiedy warto stosować zagnieżdżanie, a kiedy lepiej użyć odwołań?
10. Jakie są dobre praktyki projektowania kolekcji w MongoDB pod kątem wydajności i skalowalności?
11. Jakie ograniczenia dotyczą rozmiaru dokumentów w MongoDB i jakie ma to znaczenie przy projektowaniu schematów danych?



Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską



12. Czym jest denormalizacja danych i dlaczego jest często stosowana w bazach NoSQL?
13. Jak działają indeksy w MongoDB i w jaki sposób wpływają na szybkość zapytań?
14. W jakich przypadkach warto sięgnąć po inne bazy dokumentowe, takie jak RethinkDB lub Firebase Firestore?

Podsumowanie

Zajęcia poświęcone bazom dokumentowym pozwalają zrozumieć nowoczesne podejście do przechowywania i przetwarzania danych, w którym elastyczność i skalowalność stają się równie istotne jak spójność i wydajność. Praca z MongoDB i pokrewnymi systemami pokazała, że dokumentowy model danych umożliwia projektowanie struktur bliższych rzeczywistej logice aplikacji, bez konieczności dopasowywania się do sztywnych tabel i relacji. Studenci zapoznali się z praktycznym wykorzystaniem zapytań, filtrów, operatorów agregujących oraz potoków przetwarzania danych, które w MongoDB pełnią rolę równoważną złożonym zapytaniom SQL, lecz w sposób znacznie bardziej elastyczny i naturalny dla danych półstrukturalnych.

Zrozumienie zasad zagnieżdżania, odwołań i denormalizacji danych pozwala świadomie projektować kolekcje, równoważąc wydajność, prostotę zapytań i skalowalność systemu. Dzięki temu uczestnicy potrafią dobrać właściwe strategie modelowania danych do różnych scenariuszy – od prostych aplikacji użytkowych po złożone systemy rozproszone.

W efekcie zdobyta wiedza stanowi solidną podstawę do dalszej pracy z systemami NoSQL, projektowania nowoczesnych architektur danych oraz integracji baz dokumentowych z aplikacjami w środowiskach chmurowych i mikroservisowych.

Literatura

1. How to Create Database and Collection in MongoDB, <https://www.geeksforgeeks.org/mongodb/how-to-create-database-collection-in-mongodb/>, (data ostatniego dostępu 25.09.25)
2. Getting Started with MongoDB, <https://www.mongodb.com/docs/manual/tutorial/getting-started/>, (data ostatniego dostępu 25.09.25)



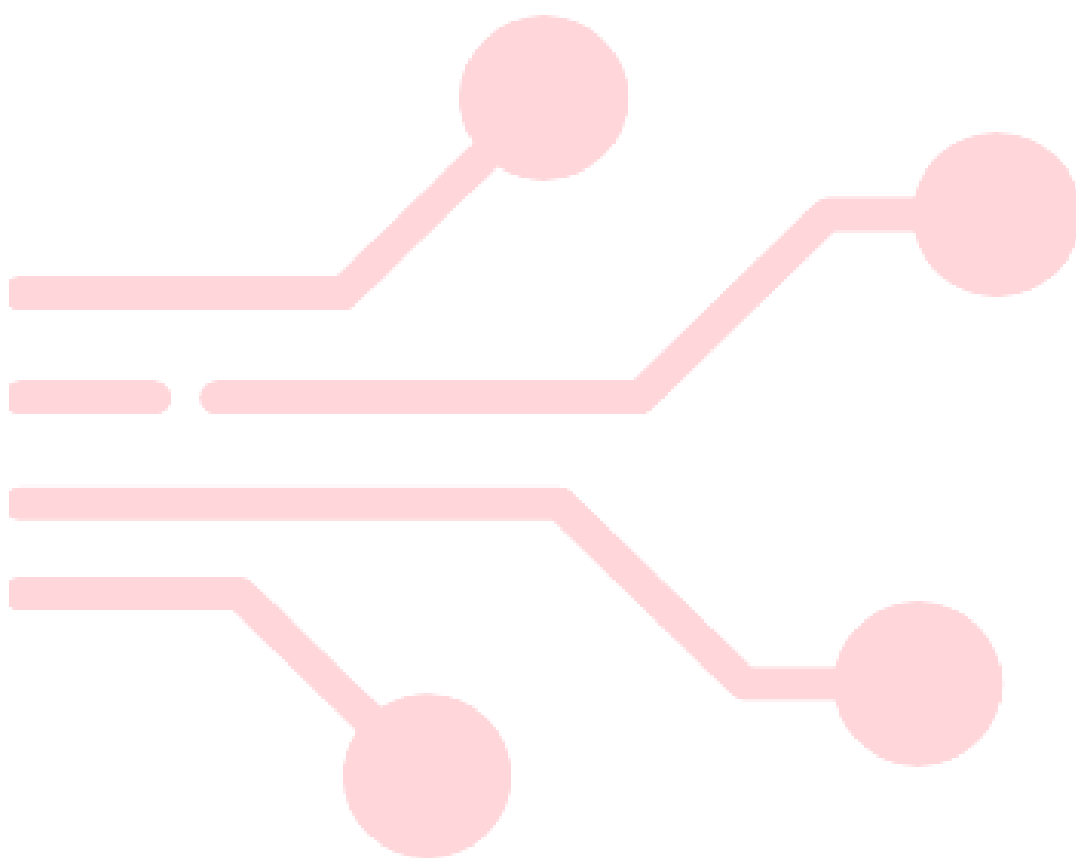
Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską





Fundusze Europejskie
dla Rozwoju Społecznego



Rzeczpospolita
Polska

Dofinansowane przez
Unię Europejską

