UNIWERSYTET MARII CURIE-SKŁODOWSKIEJ
W LUBLINIE
Wydział Matematyki, Fizyki i Informatyki

Kierunek: Informatyka

Adam Wadowski
nr albumu: 303841

Porównanie wydajności relacyjnych i nierelacyjnych baz danych: Firebase i Oracle database na przykładzie aplikacji wyszukującej gry według zadanych preferencji w technologii Spring Boot

Performance comparison of relative and non relative databases: Firebase and Oracle database on example of searching games application according to given preferences in Spring Boot technology

Praca licencjacka
napisana w Katedrze Neuroinformatki i Inżynierii Biomedycznej
pod kierunkiem dr Anny Gajos-Balińskiej

Lublin rok 2023

Spis treści

Wstęp

Wielu programistów na początku budowy projektu zderza się z problemem wyboru odpowiedniej bazy danych. Programiści oczekują często jak najmniejszego czasu oczekiwania na wołane przez nich zapytania, łatwej obsługi oraz dobrej integracji z ich aplikacjami, które tworzą do użytku masowego przez wielu użytkowników. Bazy danych zwiększają swoją objętość, a ich przeszukiwanie staje się coraz dłuższe. Osobiście borykałem się z tym wyborem już wiele razy, podczas budowy aplikacji w trakcie studiów, życia prywatnego, a także zawodowego. Najczęstszym problemem pojawiającym się na samym początku jest rozpoznanie, która baza danych będzie najbardziej efektywna dla naszej aplikacji, czy będzie to baza relacyjna, czy może nierelacyjna. Jednakże na początku drogi programisty nikt nie jest w stanie dokonać najlepszego wyboru bez odpowiedniego rozeznania.

Zdecydowanie odpowiedniej bazy danych może mieć znaczący wpływ na ogólną wydajność, skalowalność i elastyczność aplikacji. Wybór jest szczególnie trudny, gdy stajemy przed decyzją pomiędzy bazą danych relacyjną a nierelacyjną. Obydwa rodzaje baz danych mają swoje zalety i wady, które mogą być kluczowe w kontekście różnych zastosowań i wymagań.

W tej pracy porównam dwie różne bazy danych: relacyjną bazę danych Oracle i nierelacyjną bazę danych Firebase. Oceniane będą one w kontekście ich integracji z aplikacją webową, stworzoną przy użyciu technologii Spring Boot, służącą do wyszukiwania gier wideo na podstawie zadanych preferencji użytkownika. Zostaną poddane analizie różne aspekty, takie jak czas odpowiedzi na zapytania, łatwość implementacji, koszty utrzymania oraz skalowalność.

Zarówno Oracle, jako przykład bazy relacyjnej, jak i Firebase, będąc reprezentantem baz nierelacyjnych są powszechnie używane w różnorodnych aplikacjach. Poprzez porównanie ich wydajności i funkcjonalności w realnym scenariuszu użycia, ta praca ma na celu dostarczenie praktycznych wskazówek, które pomogą programistom w podjęciu świadomej decyzji odnośnie wyboru bazy danych dla ich aplikacji.

W pierwszym rozdziale zostanie przedstawiony proces tworzenia aplikacji webowej. Na podstawie tego procesu zostaną omówione podstawowe błędy, problemy oraz ich rozwiązania w trakcie tworzenia aplikacji. Poza tworzeniem aplikacji zostanie poruszony również temat porównania bazy relacyjnej i nierelacyjnej pod kątem odpowiedniego wykorzystania.

W kolejnym rozdziale zostaną przedstawione technologie użyte do utworzenia aplikacji, która będzie służyła do mierzenia wydajności baz danych poprzez porównywanie czasu wykonywania zapytań. Omówione zostaną technologie użyte przy tworzeniu *backendu* jak i *frontendu*. Technologie zostaną przeanalizowane pod kątem wad i zalet oraz przypadków użycia.

W trzecim rozdziale opisane zostanie zastosowanie technologii podanych w poprzednim rozdziale. Przedstawiony będzie kod aplikacji oraz omówienie poszczególnych funkcjonalności, które prowadzą do zbierania danych badawczych.

Ostatni rozdział zostanie poświęcony omówieniu wyników badań. Zostaną wyciągnięte wnioski oraz sformułowane podsumowanie odpowiadające na róznicę w wydajności baz relacyjnych i nierelacyjnych. Mam nadzieję, że ta praca będzie użytecznym źródłem informacji dla programistów na różnych etapach kariery, zarówno dla tych, którzy dopiero zaczynają swoją przygodę z bazami danych, jak i dla doświadczonych deweloperów, poszukujących optymalizacji ich obecnych rozwiązań.

Rozdział 1

Tworzenie aplikacji webowej

1.1. Problem wyboru technologii

Wielu programistów podczas tworzenia nowej aplikacji, zastanawia się na początku jaki język programowania wybrać, aby rozwiązać problem jak najlepiej. Obecnie do tworzenia aplikacji webowych używamy wiele technologii.

1.1.1. Popularne języki i narzędzia

Najpopularniejsze technologie to JavaScript, HTML/CSS, Python, SQL oraz Java. Aby przełożyć kod na coś widocznego, potrzebny jest kompilator, który różni się w zależności od używanego języka. Dla przykładu używając języka Java kompilatorem będzie *javac*. Korzystając z języka jakim jest Java, można używać wielu frameworków. Do stworzenia aplikacji internetowej najbardziej popularnym wyborem wśród programistów jest Spring Boot. Głównym powodem wyboru wyżej wymienionego narzędzia jest szybkość budowania aplikacji. W kilka minut można wygenerować działającą aplikację i opublikować ją na prywatnym serwerze. Wraz z szybkością idą: obszerna i ogólnodostępna dokumentacja, która potrafi poprowadzić krok po kroku nawet największego laika, możliwość prostego testowania oraz modularyzacja naszego projektu, dzięki której możemy w łatwy sposób zmieniać zależności dla konkretnego kontekstu.

1.1.2. Interakcja z użytkownikiem

Przy tworzeniu aplikacji webowej język Java służy do wystawienia tzw. kontrolerów, które pod określonym adresem kryją swoje funkcjonalności. Do odczytu oraz użycia takiego kontrolera potrzebny jest drugi język programowania, który stworzy widok dla danych pobranych z konkretnej ścieżki. Najpopularniejszym językiem stosowanym w tym momencie do współpracy z Javą jest Angular, który jest prężnie rozwijany przez firmę Google. Jest to framework napisany w języku TypeScript, którego zaletami są między innymi szybkość użycia, wydajność oraz właśnie wymienione wyżej zastosowanie języka TypeScript. Pisanie w tym frameworku polega głównie na tworzeniu komponentów, które są modyfikowane zależnie od danych oraz logiki programu.

1.1.3. Bazy danych w aplikacjach webowych

W ten sposób programista jest w stanie stworzyć strony, lecz bez możliwości zbierania danych w żaden sposób. Do gromadzenia danych potrzebna jest więc baza. Bazy danych można podzielić według kryteriów. Z uwagi na miejsce inicjalizacji należy rozważać lokalne bazy danych lub typu klient-serwer. W pierwszym rodzaju są to najprostsze zbiory, które są gromadzone na jednym komputerze, a wszelkie zmiany będzie nanosił użytkownik. Drugi rodzaj, który jest przechowywany w zasobach serwera, jest traktowany jako osobny komputer, a dostęp do niego można uzyskać poprzez połączenie sieciowe. Ze względu na architekturę wyróżniane są dwa typy: jednowarstwowe oraz dwuwarstwowe. Bazy jednowarstwowe wykonują zmiany od razu, w przeciwieństwie do drugiego typu, w którym połączenie z serwerem odbywa się za pomocą specjalnego sterownika,

a kontrolowanie danych zależy od klienta. Ostatnim podziałem baz danych jest podział względem struktur danych, których używają. Jedne zwane prostymi lub kartotekowymi określają każdą tablicę jako osobny dokument, przez co nie ma między nimi żadnego połączenia. Drugie zwane relacyjnymi bazami danych określają wiele tablic, które mogą się łączyć. Na przykład dla książki, aby wyświetlić jej zawartość, ale również z innej tabeli można wyświetlić bibliotekę, w której się znajduje. Do takiej operacji potrzeba unikalnego kodu oraz relacji między tabelami. Trzecim typem są bazy obiektowe, które są zdefiniowane tylko jednym standardem z 1993 roku. Ostatnim typem są strumieniowe bazy danych. Przedstawiają one dane w postaci zbioru strumieni. System zarządzania jest nazwany strumieniowym systemem zarządzania danymi (ang. Data Stream Management System). Jest to nowy typ, który znajduje się w fazie prototypowej i nie istnieją dla niego żadne rozwiązania komercyjne.

1.2. Projektowanie aplikacji

1.2.1. Etap planowania i analizy

W fazie planowania i analizy kluczową rolę odgrywają analitycy, którzy spotykają się z klientem w celu omówienia kluczowych aspektów projektu. W tym czasie omawiane są funkcjonalności, wymagania oraz oczekiwania względem wydajności aplikacji. Jest to etap, w którym zbierane są wszystkie niezbędne informacje, potrzebne w kolejnych fazach projektu. Nawet jeśli zespół tworzy aplikację na własne potrzeby, warto poświęcić czas na dokładne zrozumienie i zdefiniowanie tych trzech kluczowych punktów. Zapisanie ich pomoże w późniejszym etapie projektowania i implementacji. Na koniec tej fazy powstaje ogólny zarys projektu, który będzie służył jako drogowskaz w kolejnych etapach [1].

1.2.2. Projektowanie UX/UI

Drugi etap skupia się na projektowaniu doświadczenia użytkownika (UX) oraz interfejsu użytkownika (UI). Bazując na informacjach zebranych w pierwszej fazie, projektanci tworzą wizualne reprezentacje aplikacji, które pomagają zrozumieć, jak będzie ona wyglądać i funkcjonować. Kluczowym celem tego etapu jest stworzenie interfejsu, który jest intuicyjny, interaktywny i przyjazny dla użytkownika. Dobre projektowanie UX/UI może znacząco wpłynąć na sukces aplikacji, ponieważ użytkownicy często oceniają aplikacje na podstawie ich wyglądu i użyteczności.

1.2.3. Tworzenie aplikacji

Faza trzecia to etap, w którym programiści przystępują do właściwej pracy nad kodem. Bazując na specyfikacjach z poprzednich faz, tworzą działające oprogramowanie. Jest to najbardziej pracochłonny etap całego procesu, wymagający nie tylko umiejętności technicznych, ale także zdolności do rozwiązywania problemów i dostosowywania się do ewentualnych zmian w specyfikacji.

1.2.4. Testowanie

Po zakończeniu fazy tworzenia, następuje etap testowania. Jego głównym celem jest sprawdzenie, czy aplikacja działa zgodnie z oczekiwaniami klienta oraz czy nie zawiera błędów. Testerzy przeprowadzają różnego rodzaju testy, od testów jednostkowych po testy integracyjne, aby upewnić się, że wszystko działa poprawnie.

1.2.5. Wdrożenie

Przedostatni etap polega na wdrożeniu aplikacji u klienta. W tym czasie programiści i inżynierowie ds. wdrożeń pracują razem, aby zapewnić płynne i bezproblemowe uruchomienie aplikacji w środowisku produkcyjnym.

1.2.6. Utrzymanie i rozwój

Ostatnia faza to utrzymanie i rozwój aplikacji. W tym etapie programiści i testerzy monitorują działanie aplikacji, analizując jej wydajność i rozwiązując ewentualne problemy. Jeśli klient zgłasza potrzebę wprowadzenia nowych funkcjonalności, proces projektowania i wytwarzania oprogramowania rozpoczyna się ponownie, zaczynając od fazy tworzenia.

Współczesne metody wytwarzania oprogramowania są stale rozwijane, aby sprostać rosnącym wymaganiom rynku. Nowe technologie, biblioteki i frameworki pojawiają się regularnie, oferując programistom narzędzia, które ułatwiają i przyspieszają proces tworzenia aplikacji. Jednak niezależnie od używanych narzędzi, kluczem do sukcesu jest zrozumienie potrzeb klienta i dostarczenie rozwiązania, które spełni te potrzeby.

1.3. Budowa aplikacji webowej

Tworzenie aplikacji webowej to proces skomplikowany i wieloetapowy, który wymaga połączenia różnych technologii, narzędzi i praktyk. Aby zrozumieć, jak jest zbudowana typowa aplikacja webowa, warto przyjrzeć się jej głównym składnikom [2].

1.3.1. Frontend

Frontend to część aplikacji, z którą bezpośrednio kontaktuje się użytkownik. Obejmuje ona interfejs użytkownika (UI) oraz wszystkie elementy wizualne, które są prezentowane w przeglądarce. Główne technologie używane we frontendzie to:

- **HTML** – język znaczników, który opisuje strukturę strony,
- **CSS** – służy do stylizacji elementów HTML, decydując o wyglądzie strony,
- **JavaScript (lub TypeScript w przypadku Angulara)** - język programowania, który pozwala na tworzenie interaktywnych elementów strony.

1.3.2. Backend

Backend to serwerowa część aplikacji, która zajmuje się przetwarzaniem danych, komunikacją z bazą danych i realizacją logiki biznesowej. Główne składniki backendu to:

- **Serwer** – maszyna lub oprogramowanie, które obsługuje żądania od klientów i zwraca odpowiednie dane,
- **Baza danych** – system przechowywania danych, który pozwala na ich szybkie wyszukiwanie, modyfikowanie i przechowywanie,
- **API (Application Programming Interface)** – zestaw reguł i mechanizmów, które pozwalają różnym częściom oprogramowania komunikować się ze sobą.

Budowa aplikacji webowej to nie tylko kwestia techniczna, ale także planowania, projektowania i testowania. Ważne jest, aby cały proces był dobrze zorganizowany i skoordynowany, co pozwoli na stworzenie funkcjonalnej, wydajnej i bezpiecznej aplikacji.

1.4. Optymalizacja i bezpieczeństwo aplikacji webowej

W dzisiejszych czasach, kiedy konkurencja w świecie aplikacji internetowych jest ogromna, nie wystarczy stworzyć funkcjonalną stronę. Aplikacja musi być również szybka, responsywna i przede wszystkim bezpieczna. W tym rozdziale omówimy kluczowe aspekty optymalizacji i zabezpieczania aplikacji webowych.

1.4.1. Optymalizacja

Poniżej przedstawione są kroki, które wpływają na pozytywną optymalizację budowanej aplikacji:

- **Minimalizacja i kompresja plików** – zmniejszenie rozmiaru plików CSS, JavaScript i obrazów może znacząco przyspieszyć ładowanie strony. Wykorzystanie pamięci podręcznej - przechowywanie często używanych danych w pamięci podręcznej przeglądarki pozwala na szybsze ładowanie strony podczas kolejnych wizyt.
- **Optymalizacja obrazów** – stosowanie odpowiedniego formatu i rozmiaru obrazów, a także leniwe ładowanie (ang. lazy loading), które polega na ładowaniu obrazów dopiero wtedy, gdy są one widoczne dla użytkownika.
- **Optymalizacja baz danych** – regularne indeksowanie, czyszczenie i aktualizacja baz danych zapewniają ich wydajne działanie [3].

1.4.2. Bezpieczeństwo

Podstawowe zabezpieczenia wykorzystywane w aplikacjach to:

- **Szyfrowanie** – stosowanie protokołu HTTPS zapewnia, że dane przesyłane między serwerem a klientem są zaszyfrowane i trudne do przechwycenia.
- **Autentykacja i autoryzacja** – upewnienie się, że tylko uprawnieni użytkownicy mają dostęp do pewnych zasobów i funkcji aplikacji.
- **Zabezpieczenie przed atakami** – takimi jak SQL Injection, Cross-Site Scripting (XSS) czy Cross-Site Request Forgery (CSRF). Wymaga to stałego monitorowania, aktualizacji oprogramowania i stosowania najlepszych praktyk programistycznych.
- **Regularne kopie zapasowe** – w przypadku awarii lub ataku, ważne jest posiadanie aktualnych kopii zapasowych danych i kodu aplikacji, aby móc szybko przywrócić jej działanie [4].

1.4.3. Monitoring i aktualizacje

Nieustanny monitoring aplikacji pozwala na szybkie wykrywanie i reagowanie na potencjalne problemy. Regularne aktualizacje, zarówno samej aplikacji, jak i używanych technologii, zapewniają jej stabilność, wydajność i bezpieczeństwo.

Podsumowując, tworzenie aplikacji webowej to jedno, ale jej optymalizacja i zabezpieczanie to kluczowe kroki, które decydują o sukcesie projektu. Wysoka wydajność i bezpieczeństwo to czynniki, które przyciągają i zatrzymują użytkowników, dlatego nie można ich lekceważyć.

1.5. Integracja z innymi narzędziami i serwisami

Współczesne aplikacje webowe rzadko funkcjonują w izolacji. Często integrują się z różnymi zewnętrznymi narzędziami, platformami i serwisami, aby rozszerzyć swoje funkcjonalności, poprawić wydajność i dostarczyć użytkownikom bardziej kompleksowe doświadczenie. W tej sekcji przyjrzymy się kluczowym aspektom integracji aplikacji webowych z innymi systemami.

1.5.1. API - klucz do integracji

API, czyli interfejs programistyczny aplikacji, to zestaw reguł i definicji, które pozwalają różnym aplikacjom komunikować się ze sobą. Dzięki API, aplikacja webowa może na przykład pobierać dane z mediów społecznościowych, korzystać z funkcji płatności czy integrować się z systemami zarządzania treścią [5].

1.5.2. Popularne integracje

- **Płatności** – integracja z platformami takimi jak PayPal, Stripe czy PayU pozwala na szybkie i bezpieczne przeprowadzanie transakcji finansowych w aplikacji,
- **Media społecznościowe** – połączenie z Facebookiem, Twitterem czy Instagramem umożliwia na przykład udostępnianie treści, logowanie za pomocą konta w mediach społecznościowych czy analizę aktywności użytkowników,
- **Analityka** – narzędzia takie jak Google Analytics czy Hotjar dostarczają cennych informacji o zachowaniach użytkowników, co pozwala na optymalizację aplikacji i dostosowywanie jej do potrzeb odbiorców,
- **Chmura** – integracja z platformami chmurowymi, takimi jak AWS, Google Cloud czy Azure, pozwala na skalowanie aplikacji, przechowywanie danych czy korzystanie z zaawansowanych narzędzi do analizy i przetwarzania informacji.

1.5.3. Wyzwania związane z integracją

Integracja z zewnętrznymi narzędziami i serwisami niesie ze sobą pewne wyzwania. Należy między innymi uwzględnić:

- **Bezpieczeństwo** – przesyłanie i odbieranie danych z zewnętrznych źródeł musi być zabezpieczone, aby chronić prywatność użytkowników i unikać potencjalnych ataków,
- **Kompatybilność** – różne systemy mogą korzystać z różnych technologii i standardów, co może powodować problemy z integracją,
- **Aktualizacje** – zarówno aplikacja, jak i zewnętrzne narzędzia, z którymi się integruje, są regularnie aktualizowane. Należy monitorować te zmiany i dostosowywać integrację, aby wszystko działało poprawnie.

Podsumowując, integracja z innymi narzędziami i serwisami pozwala na wzbogacenie funkcjonalności aplikacji webowej i dostarczenie użytkownikom bardziej zaawansowanych i spersonalizowanych doświadczeń. Jednakże wymaga to również uwzględnienia pewnych wyzwań i stałego monitorowania połączeń z zewnętrznymi systemami.

1.6. Porównanie baz relacyjnych i nierelacyjnych: Firebase i Oracle Database

Aby zrozumieć różnice między bazami relacyjnymi a nierelacyjnymi, warto przyjrzeć się dwóm popularnym systemom zarządzania bazami danych: Firebase (nierelacyjna) i Oracle Database (relacyjna). Oba te systemy mają swoje unikalne cechy i są odpowiednie do różnych zastosowań. W kontekście aplikacji wyszukującej gry według zadanych preferencji w technologii Spring Boot, porównanie tych dwóch baz danych może pomóc w wyborze odpowiedniej technologii [6].

1.6.1. Bazy relacyjne: Oracle Database

Bazy relacyjne, takie jak Oracle Database, opierają się na strukturze tabelarycznej, gdzie dane są przechowywane w tabelach składających się z wierszy i kolumn. Kluczowe cechy baz relacyjnych to:

- **Struktura tabelaryczna:** Dane są przechowywane w tabelach, które mają zdefiniowaną strukturę kolumn.
- **Relacje między tabelami:** Możliwość tworzenia relacji między różnymi tabelami za pomocą kluczy obcych.
- **Język zapytań SQL:** Umożliwia tworzenie, modyfikowanie, usuwanie i wyszukiwanie danych za pomocą języka SQL.

- **Transakcyjność:** Gwarancja integralności danych poprzez zastosowanie mechanizmów transakcyjnych [7].

Zalety Oracle Database:

- Zaawansowane funkcje zarządzania i optymalizacji,
- Wysoka wydajność i skalowalność,
- Bogate wsparcie dla procedur składowanych i funkcji.

Wady Oracle Database:

- Wysokie koszty licencji,
- Złożoność konfiguracji i zarządzania.

1.6.2. Bazy nierelacyjne: Firebase

Firebase, będący częścią ekosystemu Google, jest bazą danych typu NoSQL, która przechowuje dane w formie obiektów JSON. Kluczowe cechy Firebase to:

- **Schemat elastyczny:** Brak stałej struktury tabelarycznej, co pozwala na łatwe modyfikacje i skalowanie.
- **Real-time:** Automatyczne aktualizacje danych w czasie rzeczywistym dla wszystkich połączonych klientów.
- **Integracja z platformą Google:** Łatwa integracja z innymi usługami Google, takimi jak autentykacja czy analiza.
- **Skalowalność:** Automatyczne skalowanie w zależności od potrzeb.

Zalety Firebase:

- Szybkość i łatwość wdrożenia,
- Elastyczność schematu,
- Wysoka dostępność i redundancja.

Wady Firebase:

- Ograniczenia w zakresie zapytań złożonych,
- Koszty mogą rosnąć w miarę zwiększania się ilości danych i zapytań [8].

1.6.3. Podsumowanie

Wybór między bazą relacyjną a nierelacyjną zależy od specyfiki projektu. Jeśli aplikacja wymaga skomplikowanych zapytań, relacji między danymi i transakcyjności, baza relacyjna jak Oracle może być lepszym wyborem. Jeśli natomiast priorytetem jest szybkość wdrożenia, elastyczność i skalowalność, Firebase może okazać się bardziej odpowiedni.

W kontekście aplikacji wyszukującej gry według zadanych preferencji, jeśli dane o grach są złożone i wymagają relacji (np. gry powiązane z twórcami, gatunkami, recenzjami), Oracle Database może być lepszym wyborem. Jeśli jednak aplikacja ma charakter bardziej dynamiczny, z częstymi aktualizacjami i interakcją w czasie rzeczywistym, należy postawić na bazę danych Firebase od Google.

Rozdział 2

Technologie i języki programowania wykorzystane do stworzenia aplikacji webowej

2.1. Java

Rysunek 2.1: Logo języka Java (źródło: https://ubiqum.com)

Java to język programowania wysokiego poziomu oraz platforma obliczeniowa, która została zaprojektowana w połowie lat 90. przez firmę Sun Microsystems. Od tego czasu Java stała się jednym z najbardziej popularnych języków programowania na świecie. Java została stworzona przez grupę inżynierów pod kierownictwem Jamesa Goslinga w Sun Microsystems w 1991 roku jako część projektu Green, który miał na celu rozwój inteligentnych urządzeń do domu. Jednakże, zamiast tego, Java znalazła swoje miejsce w świecie internetu. Pierwsza oficjalna wersja (Java 1.0) została wydana w 1996 roku.

Charakterystyczne cechy języka to między innymi:

- **Przenośność:** Dzięki maszynie wirtualnej Java (JVM), kod napisany w Javie jest przenośny i może być uruchamiany na różnych platformach bez konieczności modyfikacji.
- **Bezpieczeństwo:** Java oferuje różne mechanizmy bezpieczeństwa, takie jak zarządzanie pamięcią, które chronią przed wieloma popularnymi błędami programistycznymi.
- **Wielowątkowość:** Java obsługuje programowanie wielowątkowe, co pozwala na równoczesne wykonywanie wielu zadań.

- **Obiektowość:** Java jest językiem w pełni obiektowym, co ułatwia organizację i strukturyzację kodu.

Język ten ma bardzo bogate zastosowanie:

- **Aplikacje webowe:** Java jest często używana do tworzenia serwerów aplikacji, serwisów RESTful czy aplikacji opartych o mikroserwisy.
- **Aplikacje mobilne:** Java jest głównym językiem programowania dla systemu Android.
- **Aplikacje desktopowe:** Za pomocą JavaFX czy Swing można tworzyć bogate interfejsy użytkownika.
- **Systemy wbudowane i IoT:** Java jest używana w urządzeniach wbudowanych, takich jak telewizory, samochody czy różnego rodzaju sensory.
- **Aplikacje korporacyjne:** Java jest często wybierana do tworzenia dużych, rozbudowanych systemów korporacyjnych ze względu na jej wydajność i skalowalność.

Aby dobrze wykorzystać potencjał tego języka należy zwrócić uwagę na następujące zalecenia:

- **Ucz się bibliotek:** Java ma ogromną bibliotekę standardową oraz wiele zewnętrznych bibliotek. Znajomość tych narzędzi może znacząco przyspieszyć proces tworzenia aplikacji.
- **Dbaj o jakość kodu:** Java jest językiem, który łatwo się komplikuje. Regularne przeglądy kodu, testy jednostkowe i stosowanie wzorców projektowych mogą pomóc w utrzymaniu kodu w dobrej kondycji.
- **Bądź na bieżąco:** Java jest ciągle rozwijana. Nowe wersje przynoszą nowe funkcje, które mogą ułatwić i przyspieszyć pracę [9].

2.2. Spring Boot

Rysunek 2.2: Logo frameworku Spring Boot (źródło: https://4.bp.blogspot.com/)

Spring Boot, będący kluczowym elementem ekosystemu Spring, to nowoczesny framework zaprojektowany z myślą o uproszczeniu procesu tworzenia aplikacji opartych na Springu. Jego głównym celem jest eliminacja konieczności ręcznej konfiguracji, co pozwala programistom skupić się na tworzeniu funkcjonalności aplikacji i szybkim wdrażaniu jej.

- **Automatyzacja konfiguracji:** Spring Boot automatycznie konfiguruje aplikację na podstawie dostępnych w projekcie bibliotek. Dzięki temu programiści mogą skupić się na kodzie, nie martwiąc się o skomplikowane ustawienia.
- **Starters:** Są to zestawy zależności, które upraszczają dodawanie funkcjonalności do aplikacji. Na przykład, chcąc dodać wsparcie dla bazy danych MongoDB, wystarczy dodać odpowiedni *starter*, a Spring Boot zajmie się resztą.

- **Narzędzia produkcyjne:** Spring Boot zawiera narzędzia przeznaczone do monitorowania i zarządzania aplikacją w środowisku produkcyjnym, takie jak monitorowanie metryk, przeglądanie logów czy analiza stanu aplikacji w czasie rzeczywistym.
- **Zalety Spring Boot:**
  - **Szybkość tworzenia:** Automatyzacja konfiguracji i dostępność *starters* przyspieszają proces tworzenia aplikacji.
  - **Mikroserwisy:** Doskonała współpraca z Spring Cloud ułatwia tworzenie architektury opartej na mikroserwisach.
  - **Integracja z bazami danych:** Wsparcie dla wielu popularnych baz danych, zarówno relacyjnych, jak i nierelacyjnych.
  - **Bezpieczeństwo:** Łatwa integracja z Spring Security umożliwia dodawanie funkcji bezpieczeństwa do aplikacji.
- **Zastosowania:** Spring Boot jest niezwykle wszechstronny. Wykorzystywany jest do tworzenia aplikacji biznesowych, systemów e-commerce, aplikacji korporacyjnych oraz w rozwiązaniach opartych o mikroserwisy. Jego elastyczność i łatwość użycia sprawiają, że jest jednym z najpopularniejszych frameworków w świecie Javy [10].

2.3. Lombok

Rysunek 2.3: Logo biblioteki Lombok (źródło: https://avatars.githubusercontent.com)

Biblioteka Lombok to narzędzie dla języka Java, które znacząco upraszcza proces tworzenia kodu poprzez eliminację powtarzalnych fragmentów, takich jak gettery (metody pozwalające na pobranie zawartości prywatnego pola), settery (metody pozwalające na ustawienie prywatnego pola), konstruktory czy metody *hashCode()* i *equals()*. Te często powtarzające się fragmenty są nieodłącznym elementem tradycyjnych aplikacji Java, ale dzięki Lombokowi można je zastąpić kilkoma adnotacjami, co sprawia, że kod staje się bardziej zwięzły i czytelny.

Główne cechy i zalety Lomboka:

- **Automatyczna generacja kodu:** Lombok automatycznie generuje kod dla wielu powszechnie używanych funkcji, takich jak gettery, settery, konstruktory czy metody *toString()*. Wystarczy dodać odpowiednią adnotację do klasy lub pola, a Lombok zajmie się resztą.
- **Zwięzłość:** Dzięki Lombokowi, klasy stają się znacznie krótsze i bardziej zrozumiałe. Na przykład zamiast ręcznie pisać cały kod dla gettera i settera, można po prostu użyć adnotacji *@Getter* i *@Setter*.

- **Redukcja błędów**: Ręczne pisanie powtarzalnego kodu jest podatne na błędy. Lombok redukuje ryzyko wprowadzenia błędów poprzez automatyzację tego procesu.
- **Integracja z IDE:** Popularne środowiska programistyczne, takie jak IntelliJ IDEA czy Eclipse, oferują wsparcie dla Lomboka, co ułatwia pracę z tą biblioteką.
- **Elastyczność:** Lombok oferuje wiele adnotacji, które można dostosować do indywidualnych potrzeb. Na przykład, można kontrolować, które pola są uwzględniane w generowanych metodach czy jakie modyfikatory dostępu mają generowane metody.
- **Wsparcie dla wzorców projektowych:** Lombok ułatwia implementację niektórych wzorców projektowych, takich jak wzorzec Singleton, poprzez dostarczenie dedykowanych adnotacji, takich jak @Singleton.

Przykłady użycia Lomboka:

- **@Data:** Jest to jedna z najbardziej wszechstronnych adnotacji w Lomboku. Generuje gettery, settery, hashCode(), equals() oraz toString() dla całej klasy.
- **@Slf4j:** Dodaje loggera do klasy, co jest przydatne w aplikacjach korzystających z logowania.

Mimo wielu zalet, warto również pamiętać o pewnych ograniczeniach i potencjalnych problemach związanych z używaniem Lomboka, takich jak kompatybilność z niektórymi narzędziami czy trudności w debugowaniu automatycznie wygenerowanego kodu [11].

2.4. Liquibase

Rysunek 2.4: Logo narzędzia Liquibase (źródło: https://encrypted-tbn0.gstatic.com)

Liquibase to otwarte narzędzie do zarządzania i śledzenia zmian w bazie danych. Umożliwia programistom kontrolę wersji schematu bazy danych, co jest niezbędne w dynamicznie rozwijających się aplikacjach, gdzie struktura bazy danych może ulegać częstym modyfikacjom.

Główne cechy i zalety Liquibase to między innymi:

- **Kontrola wersji schematu bazy danych:** Podobnie jak systemy kontroli wersji dla kodu źródłowego, Liquibase pozwala śledzić, dokumentować i cofać zmiany w bazie danych.
- **Niezmienność:** Każda zmiana jest traktowana jako nieodwracalna. Oznacza to, że raz zastosowana migracja nie jest modyfikowana, co zapewnia spójność i powtarzalność procesu aktualizacji.
- **Formaty opisu zmian:** Zmiany w bazie danych można opisywać w różnych formatach, takich jak XML, YAML, JSON czy SQL.
- **Nie zależy od bazy danych:** Liquibase został zaprojektowany tak, aby działać z wieloma systemami baz danych. Dzięki temu można używać tego samego narzędzia niezależnie od wybranej technologii bazy danych.
- **Integracja z narzędziami budowy:** Liquibase łatwo integruje się z popularnymi narzędziami do budowy i wdrażania aplikacji, takimi jak Maven, Gradle czy Jenkins.

- **Wsparcie dla środowisk wielu deweloperów:** Wspiera scenariusze, w których wielu deweloperów pracuje nad jednym projektem, pomagając rozwiązywać konflikty i zapewniając spójność schematu bazy danych.

Przykłady użycia Liquibase:

- **Aktualizacja schematu:** Jeśli deweloper wprowadza zmiany w schemacie bazy danych, może je opisać w pliku zmian Liquibase, a następnie zastosować te zmiany w bazie danych za pomocą narzędzia.
- **Rollback:** Jeśli wprowadzona zmiana powoduje problemy, Liquibase umożliwia łatwe cofnięcie tej zmiany.

Liquibase podobnie jak każde narzędzie ma pewne ograniczenia. Może wymagać pewnego nakładu pracy przy konfiguracji, a także pewnej krzywej uczenia się, zwłaszcza dla tych, którzy nie są zaznajomieni z kontrolą wersji dla baz danych. Jedną z wad przy budowie tej aplikacji był brak integracji z Firebase. Dla wielu zespołów deweloperskich korzyści płynące z użycia Liquibase, takie jak spójność, powtarzalność i kontrola nad zmianami w bazie danych, prowadzą właśnie do wyboru tej biblioteki [12].

2.5. TypeScript

Rysunek 2.5: Logo języka TypeScript (źródło: https://blog.toothpickapp.com)

TypeScript to język programowania opracowany przez Microsoft, który rozszerza JavaScript o opcjonalne typowanie statyczne i inne funkcje. Jego głównym celem jest ułatwienie tworzenia dużych i złożonych aplikacji w JavaScript, zapewniając narzędzia i funkcje, które pomagają w identyfikacji błędów na wczesnym etapie procesu deweloperskiego.

Główne cechy i zalety TypeScript:

- **Opcjonalne typowanie statyczne:** Jedną z głównych cech TypeScript jest możliwość dodawania opcjonalnych typów do zmiennych, argumentów funkcji i wartości zwracanych. Pomaga to w wykrywaniu błędów typów na etapie kompilacji.
- **Wsparcie dla najnowszych funkcji ECMAScript:** TypeScript obsługuje najnowsze funkcje i składnię ECMAScript, a także umożliwia kompilację kodu do starszych wersji JavaScript, co jest przydatne dla zachowania kompatybilności z różnymi środowiskami.
- **Interfejsy i klasy:** TypeScript wprowadza koncepcję interfejsów i klas, które pomagają w organizacji kodu i tworzeniu bardziej modularnych i skalowalnych aplikacji.
- **Dekoratory i metadane:** TypeScript oferuje dekoratory, które pozwalają na dodawanie metadanych do klas, metod i właściwości, co jest przydatne w wielu scenariuszach, takich jak programowanie sterowane aspektami.
- **Narzędzia deweloperskie:** Dzięki integracji z popularnymi środowiskami IDE, takimi jak Visual Studio Code, TypeScript oferuje zaawansowane funkcje, takie jak podświetlanie składni, autouzupełnianie kodu i nawigacja po kodzie źródłowym.

- **Wsparcie dla typów zewnętrznych:** Społeczność TypeScript dostarcza definicje typów dla wielu popularnych bibliotek JavaScript, co ułatwia ich używanie w projektach TypeScript.

Przykłady użycia języka TypeScript to:

- **Aplikacje jednostronicowe (SPA):** TypeScript jest często używany do tworzenia zaawansowanych aplikacji internetowych, takich jak Angular, React czy Vue.
- **Aplikacje serwerowe:** Dzięki integracji z Node.js, TypeScript jest również używany do tworzenia aplikacji serwerowych.

Wprowadzenie TypeScript do istniejącego projektu JavaScript może wymagać pewnych modyfikacji w kodzie. Ponadto, choć opcjonalne typowanie jest jednym z głównych atutów TypeScript, może również wprowadzić pewną złożoność, zwłaszcza dla tych, którzy są nowi w świecie silnie typowanych języków. Dla wielu deweloperów korzyści płynące z użycia TypeScript, takie jak lepsza organizacja kodu, łatwiejsze debugowanie i większa produktywność przeważają nad potencjalnymi wadami [13].

2.6. Angular

Rysunek 2.6: Logo frameworku Angular (źródło: https://miro.medium.com)

Angular to popularny framework do tworzenia aplikacji internetowych opracowany i utrzymywany przez Google. Został zaprojektowany z myślą o tworzeniu dynamicznych, jednostronicowych aplikacji internetowych (SPA) i oferuje zestaw narzędzi do efektywnego zarządzania danymi, logiką biznesową i interfejsem użytkownika.

Główne cechy i zalety frameworku Angular to między innymi:

- **Komponentowy system architektury:** Angular opiera się na komponentach, które są niezależnymi jednostkami logiki i interfejsu użytkownika. Umożliwia to modularność, łatwe testowanie i ponowne użycie kodu.
- **Dwukierunkowe wiązanie danych (two-way data binding):** Umożliwia automatyczną synchronizację pomiędzy modelem a widokiem, co sprawia, że aktualizacje w jednym miejscu są natychmiast odzwierciedlane w drugim.
- **Wsparcie dla SPA:** Angular został zaprojektowany z myślą o tworzeniu jednostronicowych aplikacji internetowych, które oferują płynne przejścia między widokami bez konieczności przeładowywania całej strony.
- **Zaawansowany routing:** Angular oferuje potężny system routingu, który pozwala na ładowanie komponentów w oparciu o stan URL, leniwe ładowanie modułów i zagnieżdżone widoki.
- **Wsparcie dla formularzy:** Angular dostarcza narzędzia do tworzenia reaktywnych i szablonowych formularzy, które ułatwiają walidację i obsługę danych wejściowych.

- **Wbudowane narzędzia do testowania:** Angular zawiera narzędzia do jednostkowego i integracyjnego testowania aplikacji, co ułatwia zapewnienie jakości kodu.
- **Wsparcie dla programowania reaktywnego:** Dzięki integracji z biblioteką RxJS, Angular umożliwia tworzenie reaktywnych aplikacji opartych na strumieniach danych.

Przykłady użycia Angulara to na przykład:

- **Aplikacje korporacyjne:** Dzięki swojej skalowalności i wsparciu dla modułowości, Angular jest często wybierany do tworzenia dużych aplikacji korporacyjnych.
- **Platformy e-commerce:** Angular oferuje narzędzia do tworzenia dynamicznych sklepów internetowych z zaawansowanymi funkcjami.
- **Aplikacje mobilne:** Za pomocą narzędzi takich jak Ionic, Angular może być również używany do tworzenia aplikacji mobilnych.

Krzywa uczenia się Angulara jest stosunkowo stroma, zwłaszcza dla tych, którzy są nowi w świecie frameworków front-endowych. Ponadto, choć Angular jest bardzo wszechstronny, może być *zbyt ciężki* dla prostych projektów, gdzie lżejsze rozwiązania, takie jak React czy Vue, mogą być bardziej odpowiednie. Jednak dla wielu deweloperów korzyści płynące z użycia Angulara, takie jak jego wszechstronność, wsparcie społeczności i ciągłe aktualizacje decydują o wyborze tego frameworka [14].

2.7. PrimeNG

Rysunek 2.7: Logo narzędzia PrimeNG (źródło: https://primefaces.org)

PrimeNG to zestaw komponentów interfejsu użytkownika dla Angulara. Opracowany przez PrimeTek, PrimeNG dostarcza bogatą kolekcję gotowych do użycia komponentów, które ułatwiają szybkie tworzenie zaawansowanych aplikacji internetowych z wykorzystaniem Angulara.

Główne cechy i zalety narzędzia PrimeNG to między innymi:

- **Bogata kolekcja komponentów:** PrimeNG oferuje szeroką gamę komponentów, od podstawowych elementów, takich jak przyciski czy listy, po zaawansowane komponenty, takie jak wykresy, drzewa czy tabelki z funkcją paginacji.
- **Tematyzacja:** PrimeNG dostarcza zestaw gotowych motywów, które pozwalają na szybkie dostosowanie wyglądu aplikacji do indywidualnych potrzeb. Dodatkowo, dzięki wsparciu dla narzędzia theming API, użytkownicy mogą tworzyć własne motywy.
- **Wsparcie dla mobilności:** Komponenty PrimeNG są responsywne i dostosowują się do różnych rozmiarów ekranów, co czyni je odpowiednimi zarówno dla aplikacji desktopowych, jak i mobilnych.
- **Integracja z Angularem:** Jako że PrimeNG został zaprojektowany specjalnie dla Angulara, jego komponenty doskonale integrują się z tym frameworkiem, oferując spójne API i wydajność.

- **Wysoka wydajność:** Komponenty PrimeNG są zoptymalizowane pod względem wydajności, co zapewnia płynne działanie nawet w dużych i złożonych aplikacjach.
- **Wsparcie społeczności i dokumentacja:** PrimeNG posiada aktywną społeczność, która regularnie dzieli się wskazówkami i rozwiązaniami problemów. Ponadto, biblioteka ta jest dobrze udokumentowana, co ułatwia jej wdrożenie i użycie.

Przykłady użycia narzędzia PrimeNG w aplikacjach webowych to:

- **Zaawansowane panele administracyjne:** Dzięki szerokiej gamie komponentów, takich jak tabele, wykresy czy formularze, PrimeNG jest idealnym wyborem do tworzenia zaawansowanych paneli administracyjnych.
- **Aplikacje biznesowe:** PrimeNG jest często wykorzystywany w aplikacjach korporacyjnych, gdzie potrzebne są zaawansowane komponenty interfejsu użytkownika.
- **Platformy e-commerce:** Komponenty takie jak karuzele produktów, listy rozwijane czy kalendarze czynią PrimeNG atrakcyjnym wyborem dla platform e-commerce.

PrimeNG może wymagać pewnego nakładu pracy przy konfiguracji, a niektóre komponenty mogą nie być dostosowane do bardzo specyficznych potrzeb. Korzyści płynące z użycia PrimeNG, takie jak bogata kolekcja komponentów, wsparcie społeczności i ciągłe aktualizacje, przeważają przy wyborze odpowiedniego narzędzia [15].

2.8. Podsumowanie

Wybór odpowiednich technologii i narzędzi jest jednym z najważniejszych etapów w procesie tworzenia aplikacji. Decyzje te mają bezpośredni wpływ na wydajność, skalowalność i utrzymanie projektu w przyszłości. Omówione w tym rozdziale technologie, takie jak Java, Spring Boot, Lombok, Liquibase, TypeScript, Angular i PrimeNG, stanowią przykład zaawansowanych i sprawdzonych rozwiązań, które są szeroko stosowane w branży IT. Każde z nich przynosi unikalne korzyści, które mogą znacząco przyczynić się do sukcesu projektu.

Jednakże, niezależnie od zalet poszczególnych technologii, kluczem jest ich prawidłowe zastosowanie i integracja. Ważne jest, aby zrozumieć specyfikę projektu, jego wymagania oraz oczekiwania użytkowników. Tylko wtedy można dokonać świadomego wyboru, który przyniesie oczekiwane korzyści.

Współczesny świat technologii oferuje nieskończone możliwości. Dlatego też, nieustanne kształcenie się, eksplorowanie nowych narzędzi i adaptacja do zmieniającego się środowiska są kluczem do tworzenia innowacyjnych i skutecznych rozwiązań. W końcu to nie tylko technologia, ale przede wszystkim ludzie i ich wizja determinują sukces każdego projektu.

Rozdział 3

Budowa aplikacji do zbierania danych badawczych

Zbieranie danych badawczych jest kluczowym elementem wielu projektów naukowych i biznesowych. Aby ułatwić ten proces, postanowiłem stworzyć aplikację, która pozwoli na efektywne i systematyczne gromadzenie informacji. W tym rozdziale zostanie opisane jak zbudowano aplikację, składającą się z dwóch głównych komponentów: backendu i frontendu.

3.1. Backend

Backend aplikacji został zbudowany z użyciem języka Java oraz frameworka Spring Boot, który był niezbędny do utworzenia aplikacji webowej i wspomagał routing. Do utworzonego projektu najpierw zaimportowano bibliotekę Lombok oraz narzędzie Liquibase do sprawnego korzystania z baz danych, poprzez zależności takie jak na Listingu 3.1.

```
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
</dependency>
<dependency>
    <groupId>org.liquibase</groupId>
    <artifactId>liquibase-core</artifactId>
    <version>4.20.0</version>
</dependency>
```

Listing 3.1: Zależności dla Lomboka i Liquibase

Następnie zajęto się utworzeniem połączenie do bazy danych Oracle poprzez dodanie odpowiedniej zależności dla Mavena widocznej na Listingu 3.2.

```
<dependency>
   <groupId>com.oracle.database.jdbc</groupId>
   <artifactId>ojdbc8</artifactId>
   <version>21.9.0.0</version>
</dependency>
```

Listing 3.2: Zależności dla Oracle database

Po uwzględnieniu zależności ustawiono konfigurację w *application.properties* dla indywidualnego podłączenia do bazy danych, która powinna wyglądać tak jak na Listingu **??**

```
spring.liquibase.change-log=classpath:/db/changelog/db.changelog
    -master.xml
spring.liquibase.contexts=development
```

```
3 spring.datasource.url=jdbc:oracle:thin:@10.169.168.5:1521:xe
4 spring.datasource.username=C##AWMED
5 spring.datasource.password=pzespol
6 spring.datasource.driver-class-name=oracle.jdbc.OracleDriver
7
8 spring.jpa.database-platform=org.hibernate.dialect.
      Oracle12cDialect
9 spring.jpa.hibernate.ddl-auto=update
10 firebase.config.path=key.json
```

Listing 3.3: Konfiguracja application.properties

W ten sposób uzyskano pełne podłączenie do jednej z dwóch baz danych potrzebnych do testów wydajności. Kolejnym krokiem było podłączenie bazy Firebase od Google, czyli dodanie kolejnej zależności:

```
1 <dependency>
2    <groupId>com.google.firebase</groupId>
3    <artifactId>firebase-admin</artifactId>
4    <version>9.2.0</version>
5 </dependency>
```

Listing 3.4: Zależności dla Firebase

Po dodaniu zależności należało utworzyć klasę konfiguracyjną w kodzie, która będzie pozwalała na ustanowienie połączenia z Firebase oraz pobieranie i wysyłanie danych na serwer. Tą klasą jest *FirebaseConfig* z odpowiednią adnotacją *@Configuration* powodującą zainicjalizowanie tej klasy przy uruchomieniu aplikacji. Klasa ta została przedstawiona na Listingu 3.5.

```
1
2 @Configuration
3 public class FirebaseConfig {
4
5
6     @Bean
7     public FirebaseApp firebaseApp() throws IOException {
8         InputStream refreshToken = getClass().
              getResourceAsStream("/key.json");
9         FirebaseOptions options = FirebaseOptions.builder()
10                .setCredentials(GoogleCredentials.fromStream(
                      refreshToken))
11                .setDatabaseUrl("https://databases-3fd9e.
                      firebaseio.com/")
12                .build();
13
14        FirebaseApp.initializeApp(options);
15
16        return FirebaseApp.getInstance();
17    }
18
19    public List<GameApi> getFromFireBase(){
20        List<GameApi> gameApiList = new ArrayList<>();
21        DatabaseReference rootRef = FirebaseDatabase.getInstance
              ().getReference();
22        rootRef.addListenerForSingleValueEvent(new
              ValueEventListener() {
23            @Override
24            public void onDataChange(DataSnapshot dataSnapshot)
                  {
```

```
25                  for(DataSnapshot ds : dataSnapshot.getChildren()
                        ) {
26                      GameApi gameApi = ds.getValue(GameApi.class)
                            ;
27                      gameApiList.add(gameApi);
28                  }
29              }
30
31              @Override
32              public void onCancelled(DatabaseError error) {
33                  System.out.println("The read failed: " + error.
                        getCode());
34              }
35          });
36          return gameApiList;
37      }
38
39      public void addToFirebase(FireBaseApi gameApi) {
40          Firestore firestore = FirestoreClient.getFirestore();
41          CollectionReference gamesCollection = firestore.
                collection("games");
42
43          ApiFuture<DocumentReference> future = gamesCollection.
                add(gameApi);
44
45          try {
46              DocumentReference documentReference = future.get();
47              System.out.println("Game added to Firestore
                    successfully at: " + documentReference.getPath())
                    ;
48          } catch (InterruptedException | ExecutionException e) {
49              System.out.println("Error adding game to Firestore:
                    " + e.getMessage());
50          }
51      }
52  }
```

Listing 3.5: Klasa do konfiguracji połączenia z Firebase

- *firebaseApp()* – metoda inicjalizująca połączenie z bazą Firebase, wywołuje klasę *FirebaseOptions*, która po ustawieniu odpowiednich „credentials" oraz „databaseUrl" odnoszącego się do adresu mojej bazy tworzy połączenie i zwraca instancję tego połączenia.
- *getFromFireBase()* – metoda pozwalająca na pobranie obiektów z bazy danych, wołana później w momencie uzyskiwania rekordów dla użytkownika, co będzie omawiane w dalszej części rozdziału
- *addToFireBase()* – metoda pozwalająca na dodawanie rekordów do bazy danych, klasa *CollectionRefference* pozwalała na określenie katalogu dla konkretnych rekordów

Na tym etapie, w aplikacji jest już połączenie do bazy obu baz danych oraz zaimportowane potrzebne zależności. Skupiono się więc na tworzeniu modelu API gry, która będzie przechowywana na bazie oraz zwracana do wyświetlenia u użytkownika. Model ten został przedstawiony na Listingu 3.6.

```
1 public class GameApi {
2     private Long id;
```

```
3     private String gameName;
4     private String gameType;
5     private String multiplayer;
6     private String platform;
7     private Long age;
8     private String wydawca;
9     private LocalDate dateOfOut;
10    private String transactions;
11    private String motyw;
12    private String description;
13    private Long ranking;
14 }
```

Listing 3.6: Klasa API dla gry

Aby poprawnie dodać obiekt do Firebase, daty powinny być zapisywane w formacie *String*, przez co utworzono dedykowany model API dla Firebase, który posiadał zmieniony format dat.

Po ustaleniu odpowiedniego API skupiono się na utworzeniu tabeli w bazie Oracle przy użyciu liquibase. Liquibase jest zapisany w formacie *XML* i wygląda tak jak na Listingu przedstawionym poniżej.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <databaseChangeLog
3         xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
4         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5         xsi:schemaLocation="http://www.liquibase.org/xml/ns/
              dbchangelog
6          http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog
               -3.8.xsd">
7
8     <changeSet id="GAMES_LIST_TABLE" author="adam.wadowski">
9         <preConditions onFail="MARK_RAN">
10            <not>
11                <tableExists tableName="games_list"/>
12            </not>
13        </preConditions>
14        <createTable tableName="games_list">
15            <column name="id" type="number">
16                <constraints nullable="false" unique="true"/>
17            </column>
18            <column name="game_type" type="varchar2(255)">
19                <constraints nullable="true"/>
20            </column>
21            <column name="game_name" type="varchar2(255)">
22                <constraints nullable="true"/>
23            </column>
24            <column name="multiplayer" type="varchar2(2)">
25                <constraints nullable="true"/>
26            </column>
27            <column name="platform" type="varchar2(255)">
28                <constraints nullable="true"/>
29            </column>
30            <column name="age" type="number">
31                <constraints nullable="true"/>
32            </column>
33            <column name="wydawca" type="varchar2(255)">
34                <constraints nullable="true"/>
```

```
35              </column>
36              <column name="date_of_out" type="date">
37                  <constraints nullable="true"/>
38              </column>
39              <column name="transactions" type="varchar2(2)">
40                  <constraints nullable="true"/>
41              </column>
42              <column name="motyw" type="varchar2(255)">
43                  <constraints nullable="true"/>
44              </column>
45          </createTable>
46      </changeSet>
47
48      <changeSet id="GAMES_LIST_SEQ" author="adam.wadowski">
49          <preConditions onFail="MARK_RAN">
50              <not>
51                  <sequenceExists sequenceName="games_list_id_seq"
                        />
52              </not>
53          </preConditions>
54          <createSequence sequenceName="games_list_id_seq"
                startValue="1"/>
55      </changeSet>
56
57      <changeSet id="GAMES_LIST_add_description-1" author="adam.
            wadowski">
58          <preConditions onFail="MARK_RAN">
59              <not>
60                  <columnExists columnName="DESCRIPTION" tableName
                        ="GAMES_LIST"/>
61              </not>
62          </preConditions>
63          <addColumn tableName="GAMES_LIST">
64              <column name="DESCRIPTION" remarks="opis" type="
                    VARCHAR(1500)"></column>
65          </addColumn>
66      </changeSet>
67
68      <changeSet id="GAMES_LIST_add_ranking-1" author="adam.
            wadowski">
69          <preConditions onFail="MARK_RAN">
70              <not>
71                  <columnExists columnName="ranking" tableName="
                        GAMES_LIST"/>
72              </not>
73          </preConditions>
74          <addColumn tableName="GAMES_LIST">
75              <column name="ranking" remarks="ranking" type="
                    number"></column>
76          </addColumn>
77      </changeSet>
78
79  </databaseChangeLog>
```

Listing 3.7: Tworzenie tabeli przy użyciu Liquibase

Został jeszcze problem przechowywania wyników dla czasów wyszukiwania poszczególnych baz danych, dlatego utworzono również tabelę *SEARCHING*, która zawierała pola takie jak:

- *searching type* – typ wyszukiwania, rozróżnienie na wyszukiwanie w bazie Oracle lub Firebase
- *searching mode* – czas wyszukiwania podawany w milisekundach

Po skonfigurowaniu bazy danych Oracle, utworzono „controllery" wystawiające usługi od odczytu przez aplikację frontendową. Jeden controller przyjmuje obiekt GameApi w którym sprecyzowane są wartości do wyszukiwania gier i zwraca listę gier, które spełniają określone kryteria podane przez użytkownika. Controller ten został przedstawiony na Listingu 3.8.

```
@PostMapping("/findByCriteria")
    public List<GameApi> findByCriteria(@RequestBody GameApi
        gameApi) {
        return gameService.findByCriteria(gameApi);
    }
```

Listing 3.8: Kontroler do znajdowania gier

Jak widać obsługa tego controllera zapisana jest w gameService. Szczegółowy kod z klasy gameService jest widoczny na Listingu 3.9.

```
public List<GameApi> findByCriteria(GameApi criteria) {
        List<GameApi> results = findByCriteriaInternal(criteria)
            ;
        findFromFirebase(criteria);
        if (results.size()<3) {
            if (criteria.getMotyw() != null) {
                criteria.setMotyw(null);
            } else if (criteria.getTransactions() != null) {
                criteria.setTransactions(null);
            } else if (criteria.getDateOfOut() != null) {
                criteria.setDateOfOut(null);
            } else if (criteria.getWydawca() != null) {
                criteria.setWydawca(null);
            } else if (criteria.getAge() != null) {
                criteria.setAge(null);
            } else if (criteria.getPlatform() != null) {
                criteria.setPlatform(null);
            } else if (criteria.getMultiplayer() != null) {
                criteria.setMultiplayer(null);
            } else if (criteria.getGameType() != null) {
                criteria.setGameType(null);
            }
            return findByCriteria(criteria);
        }
        results.sort((o1, o2) -> o2.getRanking().compareTo(o1.
            getRanking()));
        return results.subList(0,3);
    }

        private List<GameApi> findByCriteriaInternal(GameApi
            criteria) {
        SearchingEntity searchingEntity = new SearchingEntity();
        searchingEntity.setSearchingType("oracle");
        long startTime = System.nanoTime();
        List<GameEntity> results = new ArrayList<>();
        gameRepository.findAll().forEach(game -> {
```

```
34              if ((criteria.getGameType() == null || game.
                    getGameType().equals(criteria.getGameType())) &&
35                      (criteria.getMultiplayer() == null || game.
                            getMultiplayer().equals(criteria.
                            getMultiplayer())) &&
36                      (criteria.getPlatform() == null || game.
                            getPlatform().equals(criteria.getPlatform
                            ())) &&
37                      (criteria.getAge() == null || game.getAge()
                            <= criteria.getAge()) &&
38                      (criteria.getWydawca() == null || game.
                            getWydawca().equals(criteria.getWydawca()
                            )) &&
39                      (criteria.getDateOfOut() == null || game.
                            getDateOfOut().equals(criteria.
                            getDateOfOut())) &&
40                      (criteria.getTransactions() == null || game.
                            getTransactions().equals(criteria.
                            getTransactions())) &&
41                      (criteria.getMotyw() == null || game.
                            getMotyw().equals(criteria.getMotyw()))))
                            {
42                  results.add(game);
43              }
44          });
45          long endTime = System.nanoTime();
46          long duration = (endTime - startTime);
47          searchingEntity.setSearchingTime(String.valueOf(duration
                /1_000));
48          searchingRepository.save(searchingEntity);
49          return gameConverter.fromEntityList(results);
50      }
51
52      @Async
53      protected void findFromFirebase(GameApi criteria){
54          SearchingEntity searchingEntity = new SearchingEntity();
55          searchingEntity.setSearchingType("firebase");
56          long startTime = System.nanoTime();
57          List<GameApi> results = new ArrayList<>();
58          firebaseConfig.getFromFireBase().forEach(game -> {
59              if ((criteria.getGameType() == null || game.
                    getGameType().equals(criteria.getGameType())) &&
60                      (criteria.getMultiplayer() == null || game.
                            getMultiplayer().equals(criteria.
                            getMultiplayer())) &&
61                      (criteria.getPlatform() == null || game.
                            getPlatform().equals(criteria.getPlatform
                            ())) &&
62                      (criteria.getAge() == null || game.getAge()
                            <= criteria.getAge()) &&
63                      (criteria.getWydawca() == null || game.
                            getWydawca().equals(criteria.getWydawca()
                            )) &&
64                      (criteria.getDateOfOut() == null || game.
                            getDateOfOut().equals(criteria.
                            getDateOfOut())) &&
```

```
65                      (criteria.getTransactions() == null || game.
                            getTransactions().equals(criteria.
                            getTransactions())) &&
66                      (criteria.getMotyw() == null || game.
                            getMotyw().equals(criteria.getMotyw()))
                            {
67                  results.add(game);
68              }
69          });
70          long endTime = System.nanoTime();
71          long duration = (endTime - startTime);
72          searchingEntity.setSearchingTime(String.valueOf(duration
                /1_000));
73          searchingRepository.save(searchingEntity);
74      }
```

Listing 3.9: Proces wyszukiwania gry według kryteriów

Obiekty są wyszukiwane na podstawie podanych kryteriów przez użytkownika w dwóch bazach, jednak zwracana lista jest zwracana tylko z bazy Oracle. Obie metody działają identycznie mierząc czas dla takiej samej operacji, a później zapisując czas do tabeli *SEARCHING* przez co można łatwo odczytać wynik jednym zapytaniem.

Drugą usługą wystawioną w *Controllerze* jest usługa do zmiany rankingu danej pozycji, użytkownik po wyszukiwaniu gier, będzie miał możliwość oceny czy gra spełnia jego oczekiwania czy nie, co będzie wpływało na ranking konkretnej gry i skutkowało jej wyświetlaniem w późniejszym etapie. Usługa ta jest widoczna na Listingu 3.10.

```
1       @PutMapping("/changeRanking/{id}/{ranking}")
2       public void changeRanking(@PathVariable Long id,
            @PathVariable Long ranking){
3           gameService.changeRanking(id,ranking);
4       }
```

Listing 3.10: Kontroler do zmiany rankingu

W serwisie usługa prezentuje się w sposób przedstawiony na Listingu 3.11.

```
1       public void changeRanking(Long id, Long ranking){
2           GameEntity gameEntity = gameRepository.findById(id).
                orElseThrow(() -> new RuntimeException("Nie
                znaleziono gry o podanym id"));
3           gameEntity.setRanking(ranking);
4           gameRepository.save(gameEntity);
5       }
```

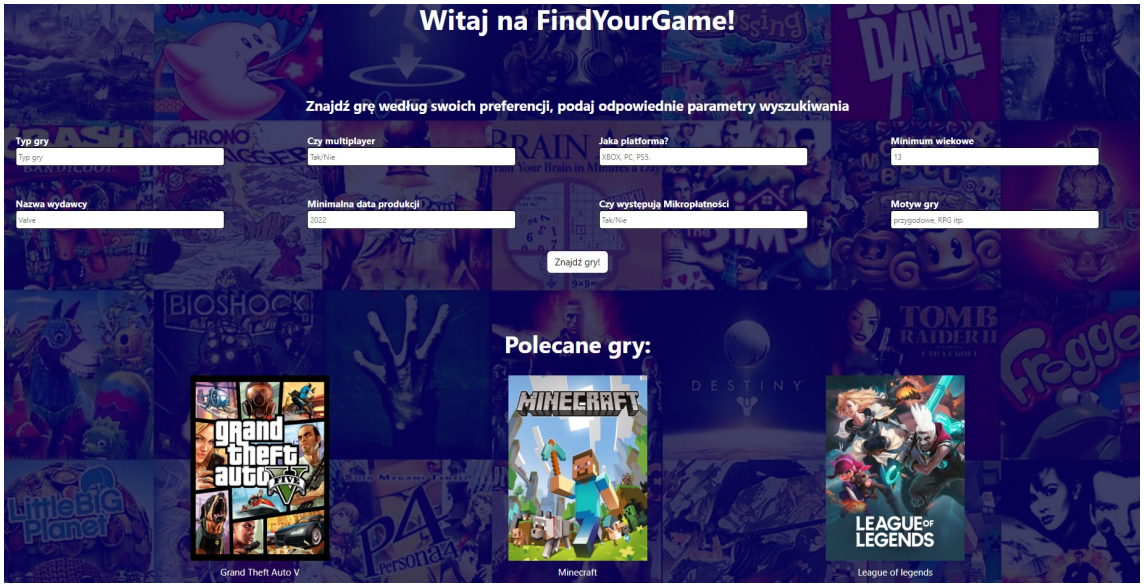
Listing 3.11: Obsługa zmiany rankingu

W ten sposób uzyskano poprawnie działający backend, który zawiera podłączenie do obu baz danych oraz zapisuje informacje odnośnie czasu wyszukiwania między nimi.

3.2. Frontend

Frontend aplikacji został zbudowany przy użyciu frameworka Angular, który jest jednym z najpopularniejszych narzędzi do tworzenia dynamicznych i zaawansowanych interfejsów użytkownika. Umożliwia on jednostronicowe renderowanie aplikacji oraz oferuje szereg funkcji, takich jak dwukierunkowe wiązanie danych, iniekcja zależności czy modularyzacja.

Do stylowania i budowy interfejsu użytkownika wykorzystano bibliotekę komponentów PrimeNG. Jest to zestaw bogatych komponentów UI dla Angulara, który znacznie ułatwia i przyspiesza proces rozwijania aplikacji.



Rysunek 3.1: Strona główna aplikacji

Strona główna utworzonej do celów badawczych aplikacji prezentuje się następująco:
Na stronie poza polami do wyszukiwania, widnieją również polecane gry przez innych użytkowników. Użytkownik może wprowadzić 8 parametrów do wyszukiwania gry takich jak:

- **Typ gry:** np. RPG, FPS, Adventure, Strategy, MOBA, Sports, Racing
- **Czy multiplayer:** pole powinno przyjmować wartości Tak lub Nie w zależności od tego na jakiej grze zależy użytkownikowi
- **Jaka platforma:** np. PC, XBOX One, XBOX Series X, PlayStation 4, Playstation 5, Google, Nintendo, VR, Steam
- **Minimum wiekowe:** pole określa wiek gry licząc datę teraźniejszą odjąć data wydania gry
- **Nazwa wydawcy:** np. Electronic Arts, Ubisoft, Nintendo, Rockstar Games
- **Minimalna data produkcji:** pole określa rok wydania minimalny dla gry. Jeśli gra została wydana wcześniej niż podano, nie zostanie wzięta pod uwagę
- **Czy występują Mikropłatności:** pole typu Tak/Nie
- **Motyw gry:** np: Apocalypse, Medieval, Space, Cyberpunk, Pirates, Western

Po wprowadzeniu parametrów w całości lub częściowo, użytkownik powinien kliknąć przycisk *Znajdź grę*. Kod dla przycisku został przedstawiony w Listingu 3.12.

```
1           <div class="button-container">
2             <button class="btn btn-lg">Znajdź gry!</button>
3           </div>
```

Listing 3.12: Kod przycisku Znajdź gry

Obsługa po kliknięciu przycisku jest widoczna na Listingu **??**.

```
1   submitGameForm():void {
2     console.log(this.gameForm.value)
3     const formData = this.gameForm.value;
4
5     this.http.post<Game[]>('http://localhost:8080/api/games/
          findByCriteria', formData).subscribe(
```

```
6         (response) => {
7           this.games = response;
8           this.gameService.setGames(this.games);
9           console.log(this.games);
10          this.router.navigate(['/secondpage']);
11        },
12        (error) => {
13          console.error('Error:', error);
14        }
15      );
16    }
```

Listing 3.13: Metoda wywoływana po wciśnięciu przycisku

Po wciśnięciu przycisku aplikacja frontendowa woła ścieżkę URL, na której został wystawiony z aplikacji backendowej controller wyszukujący odpowiednie gry dla zadanych parametrów. To właśnie w tym momencie zapisywany jest wynik mierzenia czasu pobierania danych z baz Oracle oraz Firebase. W przypadku braku gier, lista parametrów jest sukcesywnie skracana o kolejne parametry, aby znaleźć minimum 3 gry do wyświetlenia.

Po znalezieniu gier użytkownik zostaje przeniesiony na drugą stronę, na której znajduje się karuzela (panel przewijany) z 3 grami dopasowanymi do jego preferencji oraz przyciski do potwierdzenia lub odrzucenia dopasowania tej gry: Po wybraniu przycisku *Like* lub *Dislike*,

Rysunek 3.2: Strona z wynikiem wyszukiwania

przyciski znikają aby ograniczyć nadmierne głosy użytkowników.

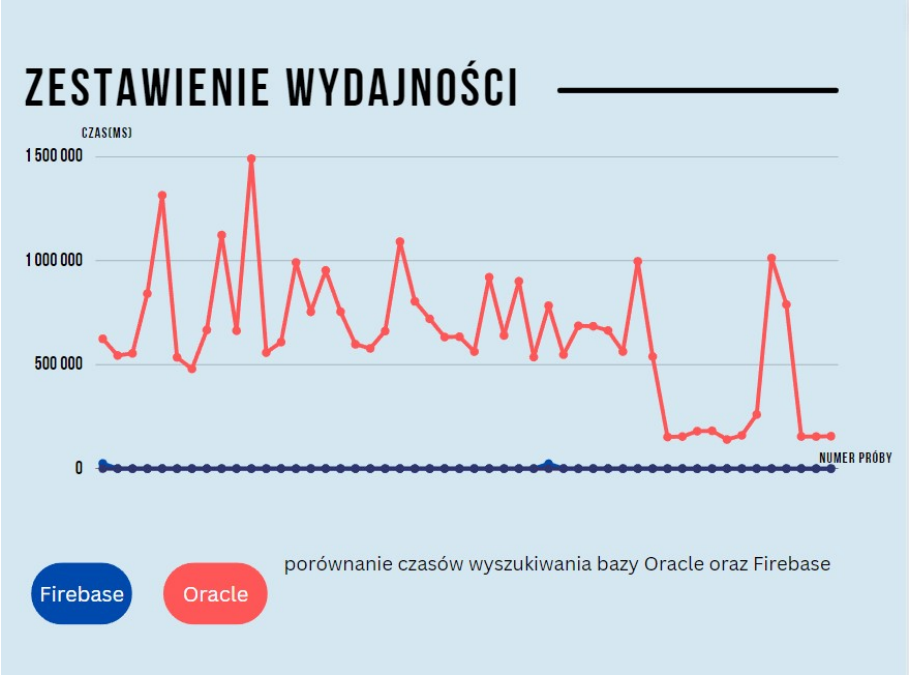
Rozdział 4

Porównanie wydajności baz relacyjnych i nierelacyjnych

W poprzednich rozdziałach skupiono się na porównaniu czysto teoretycznym oraz praktycznym. W tym rozdziale przedstawione zostaną rzeczywiste różnice po zaimplementowaniu dwóch rozwiązań w jednym projekcie. Obie bazy danych są postawione na zewnętrznym serwerze, opóźnienia połączenia w obu przypadkach są porównywalne, więc skupiono się na czasie odpytywania. Dane zebrane z pięćdziesięciu prób są zapisane w tabeli 4.1

4.1. Wnioski z analizy czasu wyszukiwania między bazami

Przy blisko pięćdziesięciu próbach wykres prezentuje się następująco:



Rysunek 4.1: Wykres przedstawiający czas zapytania w milisekundach dla każdej próby

4.1.1. Analiza czasu reakcji i skuteczności cache

Dane do powyższego wykresu zostały pobrane z tabeli 4.1 opisanej w poprzednim rozdziale. Na osi Y przedstawiony jest czas w milisekundach potrzebny do pobrania wszystkich rekordów z

bazy danych oraz przefiltrowania ich wobec preferencji użytkownika i zwróceniu do wyświetlenia użytkownikowi. Jak widać na powyższym wykresie różnica w wydajności działania bazy danych Firebase nad bazą danych Oracle jest miażdżąca. Z tej różnicy można wyciągnąć wniosek, iż Firebase jest wydajniejszy czasowo w stosunku do bazy relacyjnej Oracle.

Bazy nierelacyjne cechują się ogromną wydajnością i prędkością przeszukiwania, nie występuje tam tabelaryzacja jak po stronie baz relacyjnych, co ułatwia strukturę. Dla programistów implementacja jest dużo prostsza, bo wymaga tylko jednego serwisu do dodawania, usuwania oraz przeszukiwania rekordów, natomiast dla bazy relacyjnej potrzebujemy budować konkretne zapytania, jeśli mamy więcej warunków do wyszukiwania. Na przykładzie aplikacji *FindYourGame* stworzonej w celu porównania baz relacyjnych i nierelacyjnych przy użyciu Firebase oraz Oracle database, dochodzimy do wniosku iż baza danych Firebase jest około 10000 razy wydajniejsza.

Chociaż wyniki wykresu sugerują, że Firebase jest znacząco szybszy pod względem wykonywania zapytań niż Oracle, ważne jest również, aby przeanalizować sposób, w jaki obie bazy danych radzą sobie z cache'owaniem danych. W przypadku Firebase, mechanizmy cache'owania są wbudowane i automatycznie synchronizują dane na urządzeniach klienckich. W przypadku Oracle, mechanizmy te są zazwyczaj bardziej konfigurowalne i mogą wymagać zastosowania dodatkowych narzędzi, takich jak Oracle Coherence.

4.1.2. Skalowalność i zasoby

Wydajność czasowa nie jest jedyną metryką, którą należy uwzględnić przy wyborze bazy danych. Skalowalność, zarówno wertykalna, jak i horyzontalna, jest równie krytyczna. Firebase, będący bazą nierelacyjną oferuje znaczące korzyści w zakresie skalowalności horyzontalnej, dzięki czemu łatwo można go rozszerzyć dodając więcej maszyn. Oracle, choć skalowalny wertykalnie, może wymagać znaczących inwestycji w sprzęt i licencje przy dużej ilości danych.

4.1.3. Koszty

Firebase, oferowany jako usługa w chmurze, ma model cenowy zorientowany na zużycie, co może być korzystne dla start-upów i małych firm. Oracle, z drugiej strony, może wymagać znacznych inwestycji początkowych, zwłaszcza gdy licencje i koszty sprzętu są uwzględnione w równaniu.

4.1.4. Kompleksowość zapytań

Jednym z ograniczeń baz nierelacyjnych jest brak wsparcia dla złożonych zapytań i operacji, takich jak złączenia JOIN, które są standardem w bazach relacyjnych. W przypadku aplikacji wymagających złożonych zapytań i analiz, Oracle może oferować znacznie większą elastyczność.

4.1.5. Bezpieczeństwo i zarządzanie danymi

Oracle, będący jednym z najstarszych graczy w branży, oferuje znacznie bardziej rozbudowane opcje zarządzania i bezpieczeństwa danych. Firebase, choć łatwy w użyciu, może nie oferować tak zaawansowanych mechanizmów zarządzania danymi, co może być krytyczne dla przedsiębiorstw o dużej skali i regulowanych sektorach.

4.2. Podsumowanie

Mimo że badania wykonane w ramach tej pracy wskazują na znaczącą przewagę Firebase w kontekście wydajności odpytywania, wybór między Firebase a Oracle jako bazą danych nie powinien opierać się wyłącznie na tej jednej metryce. Wybór odpowiedniej bazy danych powinien również uwzględniać czynniki takie jak skalowalność, koszty, możliwości zapytań, bezpieczeństwo i zarządzanie danymi. W zależności od specyficznych potrzeb i wymagań projektu, każda z tych baz danych może okazać się bardziej odpowiednia.

Tabela 4.1: Dane wydajności czasowej podane w ms

Numer	Oracle	Firebase
1	624236	24784
2	544061	251
3	554330	213
4	841631	215
5	1314402	304
6	534944	224
7	479533	283
8	666913	144
9	1123189	244
10	663310	201
11	1490633	202
12	557887	212
13	608554	203
14	991726	213
15	754099	260
16	953251	350
17	754571	384
18	597960	234
19	578189	247
20	661637	231
21	1091230	293
22	804271	304
23	720156	221
24	632934	278
25	634113	229
26	563017	232
27	920276	221
28	640287	246
29	900809	237
30	536480	474
31	783558	23448
32	547951	309
33	686642	283
34	684797	341
35	664265	313
36	563123	226
37	996893	233
38	538921	250
39	152253	180
40	154191	130
41	180145	86
42	181780	86
43	140086	282
44	160119	83
45	260219	320
46	1012348	231
47	789020	157
48	1003324	228
49	832754	193
50	580395	228

Podsumowanie

W trakcie realizacji tej pracy dokonano szczegółowego porównania dwóch różnych baz danych: relacyjnej bazy Oracle oraz nierelacyjnej bazy Firebase. Analizy te miały miejsce w kontekście ich integracji z aplikacją webową napisaną w technologii Spring Boot oraz frontendem napisanym przy użyciu Angulara, służącą do wyszukiwania gier według zadanych preferencji użytkownika.

Jednym z najbardziej znaczących odkryć jest wyraźna przewaga Firebase pod względem wydajności i szybkości czasowej w stosunku do Oracle. W przypadku różnorodnych zapytań i transakcji, Firebase znacząco przyspieszył czas odpowiedzi, co jest kluczowe dla zapewnienia dobrej jakości użytkowania w aplikacjach o dużym natężeniu ruchu i dynamicznie zmieniających się danych.

Firebase wykazał się również dużą elastycznością w obszarze skalowania, zarówno wertykalnego jak i horyzontalnego. To czyni go idealnym wyborem dla start-upów oraz dla aplikacji, które doświadczają dynamicznego wzrostu liczby użytkowników czy też zmiennego wolumenu danych. Jego struktura oparta na JSON-ie ułatwia również pracę z różnymi formami nierelacyjnych danych, co może być atutem w wielu nowoczesnych aplikacjach webowych i mobilnych.

Nie można jednak pominąć, że Oracle z jego rozbudowanym zestawem funkcji i mechanizmów zapewniających konsystencję i trwałość danych, pozostaje solidnym wyborem dla zastosowań korporacyjnych, gdzie te cechy są niezbędne. Wszakże, w kontekście badanej aplikacji, te atuty nie były wystarczająco przekonujące, aby zrównoważyć różnice w wydajności.

Co więcej, warto zwrócić uwagę na różnice w kosztach i złożoności administracyjnej. Firebase, będąc rozwiązaniem chmurowym, oferuje znaczne ułatwienia w zarządzaniu i utrzymaniu bazy, co również może wpłynąć na końcowy wybór, szczególnie dla mniejszych zespołów deweloperskich lub indywidualnych programistów.

W kontekście tych odkryć, praca ta dostarcza wartościowych wskazówek dla praktyków i teoretyków w dziedzinie baz danych. Ostateczny wybór bazy danych powinien być dokonany po uwzględnieniu szeregu kryteriów, takich jak wydajność, skalowalność, koszt, oraz specyficzne potrzeby projektu. Jednak, w kontekście badanego przypadku i kryteriów wydajności, Firebase prezentuje się jako zdecydowanie lepsza opcja.

Bibliografia

[1] Adam Mateja, *Proces tworzenia aplikacji w pigułce – 6 kluczowych etapów*, [Online]. Available: https://studiosoftware.pl/blog/proces-tworzenia-aplikacji/

[2] Budowa Aplikacji Webowej, *Z czego składa się aplikacja webowa?*, [Online]. Available: https://thestory.is/pl/journal/z-czego-sklada-sie-aplikacja-webowa/

[3] Optymalizacja aplikacji webowej, *Jak zwiększyć wydajność aplikacji webowych dzięki optymalizacji kodu*, https://weastronauts.com/jak-zwiekszyc-wydajnosc-aplikacji-webowych-dzieki-optymalizacji-kodu/

[4] Bezpieczeństwo aplikacji webowej, Mariusz Sojak, Szymon Głowacki, Paweł Policewicz *METODY ZABEZPIECZEŃ PRZESYŁU DANYCH W SIECI INTERNET*

[5] Api- klucz do integracji, *Database Systems Journal*, https://www.dbjournal.ro/archive/2/2.pdf

[6] Różnice baz danych, David Pawlan *Relational vs. Non-Relational Database: Pros and Cons*, https://aloa.co/blog/relational-vs-non-relational-database-pros-cons#

[7] Cechy bazy Oracle, *Oracle documentation*, https://docs.oracle.com/cd/E63057_01/doc.30/e62305/mpimp_chapter3.htm

[8] Cechy bazy Firebase, *Firebase Documentation*, https://firebase.google.com/docs/database?hl=en

[9] Java *Java Language and Virtual Machine Specifications*, https://docs.oracle.com/javase/specs/

[10] Spring Boot *Advantages of Spring Boot*, https://www.adservio.fr/post/advantages-of-spring-boot

[11] Lombok *Lombok Documentation*, https://projectlombok.org/

[12] Liquibase *Liquibase Documentation*, https://docs.liquibase.com/home.html

[13] Typescript *TypeScript Documentation*, https://www.typescriptlang.org/

[14] Angular *Angular Documentation*, https://angular.io/

[15] PrimeNG *PrimeNG Documentation*, https://primeng.org/

Spis tabel

Spis listingów

Spis rysunków