2024

Projektarbeit Modul 165

NOSQL MIT MONGODB

AIMO ALTORFER

Contents

No table of contents entries found.

Migration der Ski-Service Auftragsverwaltung von SQL zu NoSQL

Initialisierung

Projektübersicht

Projekttitel: Migration der Ski-Service Auftragsverwaltung von SQL zu NoSQL

Projektziel: Umstellung der bestehenden SQL-basierten Datenbank des Ski-Service Auftragsverwaltungssystems auf eine NoSQL-Datenbank (MongoDB oder Neo4j), um Skalierbarkeit, Performance und Kosteneffizienz zu verbessern.

Hintergrund

Die Firma Jetstream-Service benötigt aufgrund der Expansion und des erhöhten Datenaufkommens eine leistungsfähigere, skalierbarere und kosteneffektivere Datenbanklösung. Die aktuelle relationale Datenbanklösung stösst an ihre Grenzen bezüglich Datenverteilung und Skalierung.

Projektumfang

Datenbankmigration: Migration der bestehenden Daten von einer SQL-Datenbank zu einer NoSQL-Datenbank.

API-Integration: Anpassung der bestehenden WebAPI für die Interaktion mit der neuen NoSQL-Datenbank.

Testdurchführung: Entwicklung und Durchführung von Tests zur Sicherstellung der Funktionalität und Leistung.

Dokumentation: Umfassende Dokumentation des Migrationsprozesses und des neuen Systems.

Technologische Anforderungen

NoSQL-Datenbank: Mongodb.

Programmiersprachen: C# für die WebAPI.

Testwerkzeuge: Postman für Web-API-Tests.

Versionskontrolle: Git für die Codeverwaltung.

Methodik

Das Projekt wird unter Verwendung der IPERKA-Methodik durchgeführt, um eine systematische und effiziente Umsetzung zu gewährleisten.

Schlüsselmeilensteine

Analyse und Planung

Design der NoSQL-Datenbank

Datenmigration

Anpassung der WebAPI

Testphase

Abschlussdokumentation und Präsentation

Planung

Zeitplanung

Aktivität	Geschätzte Zeit (Stunden)	Details
- Zieldefinition und Anforderungsanalyse	1	
- Auswahl der NoSQL-Datenbank	1	Entscheidung zwischen MongoDB und Neo4j
- Entwicklung des Datenbankschemas	4	Entwerfen des Schemas für die NoSQL-Datenbank
- Erstellung von Datenmodellen und Plänen	2	Entwicklung der Datenmodelle und Migrationspläne
- Entwicklung von Migrationsskripten	4	Programmierung der Skripte für die Datenübertragung von SQL zu NoSQL
- Test und Anpassung der Migrationsskripte	3	Überprüfung und Feinabstimmung der Migrationsskripte
- Anpassung der API für NoSQL-Integration	4	Anpassung der Schnittstellen für die Interaktion mit der neuen Datenbank
- Test der API-Anpassungen	3	Durchführung von Funktions- und Integrationstests
- Dokumentation der Migration	4	Erstellen der Projektdokumentation und des Migrationsberichts
- Vorbereitung der Abschlusspräsentation	2	Erstellung der Präsentationsunterlagen

Gesamt: 28

Toolübersicht

1. **Neo4j:**
    - **Verwendungszweck:** Als NoSQL-Datenbank für die Speicherung und Verwaltung der Auftragsdaten.
    - **Besonderheiten:** Ideal für komplexe Datenbeziehungen und Graph-basierte Datenmodelle.
2. **Visual Studio (VS):**
    - **Verwendungszweck:** Entwicklung der WebAPI.
    - **Sprachen und Frameworks:** Unterstützung für C#, .NET, und Integration mit anderen Microsoft-Produkten.
    - **Besonderheiten:** Umfassende Entwicklungsumgebung mit Debugging-Tools, Git-Integration und anderen hilfreichen Funktionen.
3. **GitHub:**
    - **Verwendungszweck:** Versionskontrolle und Repository-Verwaltung für den gesamten Code.
    - **Besonderheiten:** Ermöglicht die Nachverfolgung von Änderungen, Branching und Zusammenarbeit, auch wenn ich alleine arbeiten.
4. **Postman:**
    - **Verwendungszweck:** Testen der WebAPI.
    - **Besonderheiten:** Erleichtert das Senden von Anfragen, Überprüfen von Antworten und Durchführen von Integrationstests.
5. **Migrationsskripte:**
    - **Verwendungszweck:** Übertragen der Daten von der SQL-Datenbank zu Neo4j.
6. **Dokumentationswerkzeuge:**
    - **Verwendungszweck:** Erstellen der Projektdokumentation. (Word u. MarkText)
7. **Präsentationssoftware:**
    - **Verwendungszweck:** Vorbereitung der Abschlusspräsentation. (PowerPoint)

Entscheidung

Auswahl der NoSQL-Datenbank

Nach sorgfältiger Überlegung und Abwägung der verschiedenen Optionen habe ich mich für die Verwendung von MongoDB als NoSQL-Datenbank für die Migration der Ski-Service Auftragsverwaltung entschieden. Die Wahl fiel auf MongoDB aufgrund seiner hervorragenden Skalierbarkeit, Flexibilität im Umgang mit verschiedenen Datentypen und seiner Leistungsfähigkeit bei der Handhabung grosser Datenmengen.

Gründe für die Wahl von MongoDB:

Dokumentenorientierte Speicherung: MongoDB speichert Daten in einem flexiblen, JSON-ähnlichen Format, was eine dynamische Schemadefinition und eine einfache Anpassung an sich ändernde Datenanforderungen ermöglicht.

Skalierbarkeit: MongoDB unterstützt horizontale Skalierung durch Sharding, was ideal für unsere wachsenden Datensätze und die steigende Nutzerlast ist.

Leistung: MongoDB bietet schnelle Lese- und Schreibvorgänge, was für unsere Anwendung mit hohem Datenaufkommen entscheidend ist.

Reife und Community-Unterstützung: MongoDB ist eine weit verbreitete und reife NoSQL-Datenbank mit einer aktiven Community und umfangreichen Ressourcen für Entwicklung und Support.

Integration mit bestehenden Technologien: MongoDB lässt sich gut mit unseren bestehenden Technologien und Frameworks, einschliesslich C# und .NET, integrieren.

Realisierung

Die Realisierung der Migration der Ski-Service Auftragsverwaltung von einer SQL- zu einer MongoDB-basierten NoSQL-Datenbank war ein mehrstufiger Prozess, der sorgfältig geplant und umgesetzt wurde, um eine effiziente, nahtlose Integration und zuverlässige Funktionalität zu gewährleisten.

Erstellung von Datenmodell

Beschreibung

SQL-Datenbank: SkiServiceManagement

Tabelle: Orders

OrderID (INT, PRIMARY KEY, IDENTITY(1,1)): Eindeutige Kennung für jede Bestellung. Auto-Inkrementierung beginnend bei 1.

CustomerName (NVARCHAR(100)): Name des Kunden. Kann NULL sein.

CustomerEmail (NVARCHAR(100)): E-Mail-Adresse des Kunden. Kann NULL sein.

CustomerPhone (NVARCHAR(20)): Telefonnummer des Kunden. Kann NULL sein.

Priority (NVARCHAR(50)): Priorität der Bestellung. Kann NULL sein.

ServiceType (NVARCHAR(100)): Typ des angeforderten Dienstes. Kann NULL sein.

CreateDate (DATETIME): Datum und Uhrzeit der Bestellungserstellung.

PickupDate (DATETIME): Datum und Uhrzeit für das Abholen oder Bereitstellen der Dienstleistung.

Status (NVARCHAR(50), DEFAULT 'Offen'): Status der Bestellung, standardmässig auf 'Offen' gesetzt.

Comment (NVARCHAR(100), DEFAULT ''): Zusätzliche Anmerkungen zur Bestellung. Standardmässig leer.

Tabelle: Employees

EmployeeID (INT, PRIMARY KEY, IDENTITY(1,1)): Eindeutige Kennung für jeden Mitarbeiter. Auto-Inkrementierung beginnend bei 1.

Username (NVARCHAR(50), NOT NULL): Benutzername des Mitarbeiters. Muss angegeben werden.

Password (NVARCHAR(50), NOT NULL): Passwort des Mitarbeiters. Muss angegeben werden.

MongoDB-Kollektion: Orders

OrderID (ObjectId): Eindeutige Kennung für jede Bestellung in MongoDB. Automatisch generierte ObjectId.

CustomerName (String): Name des Kunden. Kann NULL sein.

CustomerEmail (String): E-Mail-Adresse des Kunden. Kann NULL sein.

CustomerPhone (String): Telefonnummer des Kunden. Kann NULL sein.

Priority (String): Priorität der Bestellung. Kann NULL sein.

ServiceType (String): Typ des angeforderten Dienstes. Kann NULL sein.

CreateDate (Date): Datum und Uhrzeit der Bestellungserstellung.

PickupDate (Date): Datum und Uhrzeit für das Abholen oder Bereitstellen der Dienstleistung.

Status (String, Default 'Offen'): Status der Bestellung, standardmässig auf 'Offen' gesetzt.

Comment (String, Default ''): Zusätzliche Anmerkungen zur Bestellung. Standardmässig leer.

Grafische Darstellung

Entwicklung der API mit MongoDB:

- Zu Beginn folgte ich dem ausführlichen Microsoft Tutorial zur Erstellung einer Web-API mit ASP.NET Core und MongoDB. Dieses Tutorial war entscheidend für den Einstieg in die Entwicklung, da es Schritt für Schritt die Einrichtung einer MongoDB-Datenbank, die Definition der Datenmodelle in C#, die Erstellung von Controller-Klassen für CRUD-Operationen (Create, Read, Update, Delete) und die Konfiguration der API-Endpunkte aufzeigte.
- Besonderer Fokus lag auf der Definition effizienter Datenmodelle, die die Vorteile von MongoDBs flexiblen, dokumentenbasierten Strukturen nutzen, und der Implementierung von robusten Methoden für die Datenmanipulation und -abfrage, die spezifisch für die Anforderungen unseres Ski-Service Auftragsverwaltungssystems zugeschnitten waren.

Implementierung von Unit Tests mit xUnit anstelle von Postman Collection:

- Nach der Entwicklung der API war der nächste Schritt die Implementierung von Unit Tests, um die Zuverlässigkeit und Funktionalität der API sicherzustellen. Hierzu wählte ich das xUnit-Testframework, bekannt für seine Flexibilität und Effektivität im .NET-Umfeld.
- Die Tests deckten eine Vielzahl von Szenarien ab, um sicherzustellen, dass die CRUD-Operationen korrekt funktionierten, die Datenvalidierung effektiv war und die Fehlerbehandlung korrekt implementiert wurde. Dies beinhaltete das Testen der API-Endpunkte, das Überprüfen der Reaktion auf ungültige oder unvollständige Daten und das Validieren der Leistung unter verschiedenen Lastbedingungen.

Integration mit dem bestehenden WPF-Frontend:

- Nach der erfolgreichen Implementierung und dem Testen der API schritt ich zur Integration mit dem bestehenden WPF (Windows Presentation Foundation) Frontend voran. Dieser Schritt war entscheidend, um eine nahtlose Benutzererfahrung zu gewährleisten.
- Die Integration erforderte eine sorgfältige Anpassung der Frontend-Komponenten, um die Kommunikation mit der neuen API zu ermöglichen. Dies umfasste die Aktualisierung der Datenbindungen, die Anpassung der Benutzeroberflächenlogik

und die Sicherstellung, dass das Frontend effizient mit den neuen, von der MongoDB-API bereitgestellten Datenstrukturen interagierte.

Erstellung von Backup- und Restore-Skripten:

- Ein kritischer Aspekt der Systemwartung und -sicherheit ist die Fähigkeit, Daten zu sichern und bei Bedarf wiederherzustellen. Daher entwickelte ich spezifische Skripte für das Backup und Restore der MongoDB-Daten.
- Diese Skripte wurden so konzipiert, dass sie regelmässige, automatisierte Backups der Datenbank ermöglichen und im Falle eines Systemausfalls oder Datenverlustes eine schnelle und effiziente Wiederherstellung der Daten unterstützen. Die Skripte beinhalteten auch Protokollierungsfunktionen, um den Backup- und Restore-Prozess zu überwachen und zu verifizieren.
- Ausführung der Skripte:

```
chmod +x /pfad/zum/skript/backup.sh
chmod +x /pfad/zum/skript/restore.sh
```

Kontrolle

Die Testphase des Projekts "Migration der Ski-Service Auftragsverwaltung von SQL zu NoSQL" umfasste sowohl automatisierte Tests mit xUnit als auch manuelle Tests, um eine umfassende Qualitätssicherung zu gewährleisten.

1. **xUnit-Tests:**
    - **Entwicklung der Testfälle:** Basierend auf den Anforderungen und Funktionalitäten der API wurden spezifische Testfälle entwickelt. Diese Testfälle zielten darauf ab, alle Aspekte der API zu prüfen, einschliesslich der korrekten Funktion der CRUD-Operationen, der Handhabung von Ausnahmen und der Leistung unter verschiedenen Bedingungen.
    - **Implementierung der Testmethoden:** Für jeden Testfall wurden entsprechende Testmethoden in xUnit implementiert. Diese Methoden umfassten Tests für die Datenbankverbindung, die Validierung von Datenmodellen, die Überprüfung der korrekten Rückgabe von Daten und Fehlermeldungen sowie die Leistungsbewertung von Datenbankabfragen.
    - **Durchführung und Überwachung der Tests:** Die Tests wurden regelmässig ausgeführt, um kontinuierlich die Funktionalität und Stabilität der API zu überprüfen. Besondere Aufmerksamkeit wurde auf Tests gelegt, die nach Änderungen am Code oder an den Datenstrukturen durchgeführt wurden, um Regressionen zu vermeiden.
2. **Manuelle Tests:**
    - **Benutzeroberflächentests:** Zusätzlich zu den automatisierten Tests wurden manuelle Tests an der Benutzeroberfläche durchgeführt, um die Integration der API mit dem WPF-Frontend zu überprüfen. Diese Tests konzentrierten sich auf die Benutzererfahrung, das korrekte Laden von Daten, die Reaktionsgeschwindigkeit der Oberfläche und die korrekte Darstellung von Daten.
    - **Fehlersimulation und Ausnahmehandhabung:** Manuelle Tests wurden auch verwendet, um Fehlerszenarien zu simulieren und zu überprüfen, wie das System auf unerwartete Eingaben oder Ausnahmesituationen reagiert. Dies beinhaltete das Testen der Fehlerbehandlung, der Datenvalidierung und der Sicherheitsfunktionen der Anwendung.

Auswerten

In der Auswertungsphase des Projekts "Migration der Ski-Service Auftragsverwaltung von SQL zu NoSQL" wurden die Ergebnisse und Leistungen des neuen Systems analysiert. Die Schlüsselerkenntnisse waren:

1. **Erfüllung der Projektziele:** Die Umstellung auf MongoDB verbesserte die Skalierbarkeit, Performance und Kosteneffizienz des Systems, was den ursprünglichen Projektzielen entsprach.
2. **Technische Bewertung:** Die xUnit-Tests und manuellen Tests bestätigten die technische Zuverlässigkeit und Funktionalität der neuen API und der Datenbankintegration.