Projektdokumentation: Migration von SQL zu NoSQL für Jetstream-Service

Autoren: Brun Lenny, Ulu Ege Tuna

FEBRUARY 21, 2025

Table of Contents

Projektdokumentation: Migration von SQL zu NoSQL für Jetstream-Service

1. Einleitung

Die Firma **Jetstream-Service** ist ein kleines und mittleres Unternehmen (KMU), das Skiservice-Dienstleistungen anbietet. In den letzten Jahren wurde eine digitale Auftragsanmeldung und Verwaltung implementiert, die auf einer relationalen SQL-Datenbank basiert. Aufgrund des Unternehmenswachstums mit neuen Standorten reicht die relationale Datenbank nicht mehr aus, um die gesteigerte Anzahl von Serviceaufträgen effizient zu verwalten.

Daher hat die Geschäftsleitung entschieden, das bestehende Datenbanksystem auf eine **NoSQL-Datenbank** zu migrieren. Diese Migration soll sowohl die Skalierbarkeit als auch die Performance verbessern und gleichzeitig die Lizenzkosten reduzieren.

Projektziele

- Migration der bestehenden **relationalen Datenbank (SQL) auf eine NoSQL-Datenbank** (MongoDB).
- Sicherstellen der **Datenintegrität und -konsistenz** während der Migration.
- Anpassung der **bestehenden Web-API** für den Zugriff auf die neue NoSQL-Datenbank.
- **Implementierung eines Benutzerkonzepts** mit mindestens zwei Rollen (Admin, Mitarbeiter).
- Erstellung eines **Backup- und Restore-Konzepts** zur Datensicherung.
- **Durchführung von Tests** mit Postman zur Überprüfung der Web-API-Funktionalitäten.
- Dokumentation des gesamten Entwicklungsprozesses nach dem **IPERKA-Modell**.
- Vorstellung des Projekts in einer **Kurzpräsentation mit Live-Demo**.

2. Projektmanagement nach IPERKA

Die gesamte Planung und Umsetzung des Projekts erfolgt nach der bewährten **IPERKA-Methode**, die sich in sechs Phasen unterteilt:

2.1. Informieren

2.1.1. Projektanalyse und Ausgangslage

Die bestehende SQL-Datenbank von Jetstream-Service verwaltet Kundendaten und Serviceaufträge. Mit der Expansion des Unternehmens steigen die Anforderungen an **Datenverarbeitung, Skalierbarkeit und Geschwindigkeit**.

Die Nachteile des bestehenden Systems:

- **Langsame Abfragen bei großen Datenmengen**.
- **Hoher Verwaltungsaufwand** für relationale Datenstrukturen.
- **Lizenzkosten für SQL-Server**.
- **Schwierigkeiten bei der Datenreplikation für mehrere Standorte**.

2.1.2. Auswahl der Technologien

Basierend auf den neuen Anforderungen wurde entschieden, dass **MongoDB** als neue **NoSQL-Datenbank** eingesetzt wird. MongoDB bietet:

- **Flexible und schemalose Datenverwaltung**.
- **Bessere Skalierbarkeit durch verteilte Datenbank-Cluster**.
- **Geringere Abfragezeiten durch spezielle Indexierung**.

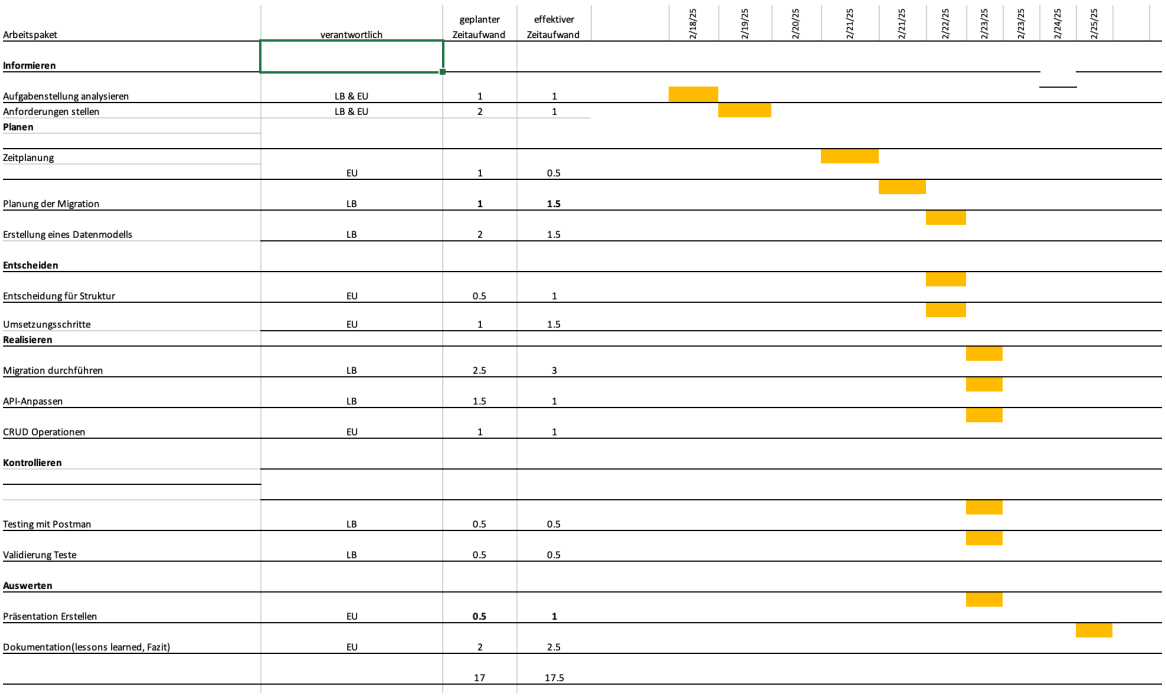
Zusätzlich bleiben folgende Technologien bestehen:

- **.NET C# Web-API**, da diese bereits im bestehenden System integriert ist.
- **Postman** für API-Tests und Validierungen.
- **GitHub** für die Versionsverwaltung.

2.2. Planen

2.2.1. Aufgabenverteilung & Zeitplanung

Das Projekt wird in mehrere Meilensteine unterteilt:



Arbeitspaket	verantwortlich	geplanter Zeitaufwand	effektiver Zeitaufwand
Informieren			
Aufgabenstellung analysieren	LB & EU	1	1
Anforderungen stellen	LB & EU	2	1
Planen			
Zeitplanung	EU	1	0.5
Planung der Migration	LB	**1**	**1.5**
Erstellung eines Datenmodells	LB	2	1.5
Entscheiden			
Entscheidung für Struktur	EU	0.5	1
Umsetzungsschritte	EU	1	1.5
Realisieren			
Migration durchführen	LB	2.5	3
API-Anpassen	LB	1.5	1
CRUD Operationen	EU	1	1
Kontrollieren			
Testing mit Postman	LB	0.5	0.5
Validierung Teste	LB	0.5	0.5
Auswerten			
Präsentation Erstellen	EU	**0.5**	**1**
Dokumentation(lessons learned, Fazit)	EU	2	2.5
		17	17.5

2.2.2. Risikobewertung

- **Datenverluste während der Migration** → Regelmäßige Backups anfertigen.
- **Fehlende Datenkonsistenz in NoSQL** → Strukturvalidierungen implementieren.
- **Unzureichende Performance in MongoDB** → Indexierung und Abfrageoptimierung testen.

2.3. Entscheiden

Basierend auf der Analyse wurden folgende Entscheidungen getroffen:

1. **MongoDB als NoSQL-Datenbank** für skalierbare und flexible Speicherung.
2. **C# .NET Web-API Anpassung**, um Daten effizient aus MongoDB abzurufen.
3. **Rollenbasierte Zugriffskontrolle** (Admin für CRUD-Operationen, Mitarbeiter für Lesezugriff).
4. **Backup-Strategie mit automatisierten Sicherungen** für Datensicherheit.
5. **Postman als Test-Tool** für API-Validierung und Performance-Checks.

2.4. Realisieren

2.4.1. Migration der SQL-Datenbank nach MongoDB

- **Umstrukturierung der relationalen Tabellen in Dokumentenbasierte Collections**.
- **Datenexport aus SQL und Import in MongoDB mit Skripten**.
- **Prüfung der Datenkonsistenz** nach der Migration.

2.4.2. Anpassung der Web-API

- **Alle API-Endpunkte wurden auf NoSQL umgestellt**.
- **Validierungen für Datenkonsistenz implementiert**.
- **Berechtigungen für verschiedene Benutzerrollen festgelegt**.

2.4.3. Backup- und Restore-Lösung

- **Regelmäßige Sicherung der Daten mit Skripten**.
- **Datenwiederherstellung getestet und dokumentiert**.

2.5. Kontrollieren

2.5.1. Teststrategie

Zur Sicherstellung der Funktionalität wurden umfassende Tests durchgeführt:

1. **Unit-Tests für API-Endpunkte**
   - Überprüfung aller CRUD-Operationen (Erstellen, Lesen, Aktualisieren, Löschen).
   - Fehlerhafte Anfragen simuliert, um Sicherheitslücken zu entdecken.
2. **Benutzertests**
   - Testdurchführung mit **Admin- und Mitarbeiter-Accounts**.
   - Überprüfung der Zugriffsbeschränkungen.
3. **Performance-Tests**
   - Abfragen mit **großen Datenmengen getestet**.
   - Indexierung analysiert und optimiert.

2.5.2. Fehleranalyse & Optimierungen

- **Daten wurden inkonsistent in MongoDB gespeichert** → Strukturvalidierung hinzugefügt.
- **API-Antwortzeiten waren zu langsam** → Indexierung verbessert.

2.6. Auswerten

2.6.1. Soll-Ist-Vergleich

Anforderung	Soll-Zustand	Ist-Zustand nach dem Projektabschluss
Datenmigration	**Alle SQL-Daten in MongoDB verfügbar.**	**Erfolgreich umgesetzt**
Web-API Anpassung	**CRUD-Operationen für NoSQL**	**Alle Endpunkte migriert**
Benutzerrollen	**Admin & Mitarbeiter mit verschiedenen Rechten**	**Implementiert**
Backup-Konzept	**Automatisierte Sicherung & Wiederherstellung**	**Getestet**

2.6.2. Lessons Learned

- **NoSQL benötigt eine andere Denkweise als SQL**.
- **Indexierung ist essenziell für schnelle Abfragen**.
- **Regelmäßige Backups verhindern Datenverluste**.

3. Präsentation

Die Ergebnisse werden in einer **15- bis 20-minütigen Kurzpräsentation** vorgestellt.

- **Live-Demo** der Web-API mit Postman.
- **Vorstellung der Datenmigration** und der neuen Struktur.
- **Lessons Learned** und Fazit.

---

4. Fazit

Das Projekt war erfolgreich: **Die SQL-Datenbank wurde vollständig auf NoSQL migriert** und die Web-API wurde entsprechend angepasst. Alle Anforderungen wurden erfüllt, und das System ist jetzt skalierbar für zukünftige Erweiterungen.