**Ski Service No SQL**

**Modul 165 - NoSQL-Datenbanken einsetzen**

Von Tunahan und Felipe

Abgabe 25.02.2025

Inhalt

[1. Informieren 3](#_Toc191419033)

[1.1 Projekt Einleitung 3](#_Toc191419034)

[1.2 Projektbeschreibung 3](#_Toc191419035)

[2. Planen 3](#_Toc191419036)

[2.1 Werkzeuge & Technologien 3](#_Toc191419037)

[2.2 GANNT 4](#_Toc191419038)

[3. Entscheiden 4](#_Toc191419039)

[3.1 Wahl des NoSQL-Datenbanksystems 4](#_Toc191419040)

[3.2 Datenmodellierung 4](#_Toc191419041)

[4. Realisieren 5](#_Toc191419042)

[4.1 Umsetzung des Web-API Backend 5](#_Toc191419043)

[4.2 Datenmigration 5](#_Toc191419044)

[4.3 Backup & Restore 5](#_Toc191419045)

[5. Kontrollieren 5](#_Toc191419046)

[5.1 Testplan & Tests 5](#_Toc191419047)

[6. Auswerten 6](#_Toc191419048)

[6.1 Lessons Learned 6](#_Toc191419049)

[7. Fazit 6](#_Toc191419050)

# 1. Informieren

# 1.1 Projekt Einleitung

Aufgrund der steigenden Nachfrage und der Expansion in neue Standorte wurde entschieden, die bestehende relationale Datenbank durch das skalierbare NoSQL-Datenbanksystem MongoDB zu ersetzen. Ziel dieses Projekts ist es, die Backend-Infrastruktur der digitalen Auftragsverwaltung zu modernisieren, indem eine Migration auf MongoDB erfolgt.

# 1.2 Projektbeschreibung

**Kernpunkte des Projekts:**

* Migration der relationalen Datenbank zu MongoDB
* Entwicklung eines Web-API Backend
* Implementierung eines Benutzerkonzepts
* Umsetzung von Backup- und Restore-Strategien
* Erstellung eines Testplans und Testprojekts

# 2. Planen

# 2.1 Werkzeuge & Technologien

* **Datenbank:** MongoDB
* **Backend:** Java, C# oder Python
* **Test:** Postman
* **Versionskontrolle:** Git (GitHub/GitLab/Bitbucket)
* **Dokumentation & Planung:** Markdown, Diagrammtools (arrows.app)

# 2.2 GANNT

Ein Bild, das Text, Screenshot, Zahl, parallel enthält.

KI-generierte Inhalte können fehlerhaft sein.

# 3. Entscheiden

# 3.1 Wahl des NoSQL-Datenbanksystems

Die Entscheidung fiel auf MongoDB, da es dokumentenorientiert ist und sich gut für flexible und schnell skalierbare Datenmodelle eignet.

# 3.2 Datenmodellierung

Die Struktur wurde von relational auf dokumentenbasiert umgestellt:

* Collections: Users, Orders, Services
* Referenzen: Nutzer-ID mit Rollen, Serviceaufträge mit Status
* Indizes: E-Mail, Auftrag-ID zur schnellen Suche

# 4. Realisieren

# 4.1 Umsetzung des Web-API Backend

* Entwicklung eines RESTful APIs mit CRUD-Funktionalitäten
* Implementierung eines Authentifizierungssystems (JWT)
* Verbindung der API mit der MongoDB-Datenbank
* Validierung der Daten mit Schema-Regeln

# 4.2 Datenmigration

* Extraktion aus der relationalen Datenbank (SQL)
* Transformation in das JSON-Format
* Import in MongoDB mittels Skripten

# 4.3 Backup & Restore

* Erstellung von Backup-Skripten für die MongoDB-Datenbank
* Dokumentation der Wiederherstellungsstrategie

# 5. Kontrollieren

# 5.1 Testplan & Tests

* Postman-Tests:
  + Endpunkte testen (GET, POST, PUT, DELETE)
  + Authentifizierung testen
  + Validierungen und Fehlerfälle testen
* Integrationstests mit Frontend
* Überprüfung der Datenkonsistenz nach Migration

# 6. Auswerten

# 6.1 Lessons Learned

* **Herausforderungen:** Schema-Migration und Benutzerverwaltung
* **Lösungen:** Schema-Validierungen und Indexierung für Performance
* **Verbesserungsmöglichkeiten:** Automatisierte Tests und bessere Dokumentation

# 7. Fazit

Die Migration der Ski-Service Auftragsverwaltung auf MongoDB war erfolgreich. Alle Anforderungen wurden erfüllt, und durch Indexierungen und Backup-Skripte wurde die Stabilität verbessert. Die Web-API ermöglicht nun zuverlässige CRUD-Operationen für Serviceaufträge.

Ausserdem konnten wir Yannicks Code korrigieren, was länger dauerte als erwartet.