

Dokumentation des Projekts Modul 165: Ski-Service nosql (IPERKA)

Ertem & Tuna



10. Februar 2025

IBZ

Inhaltsverzeichnis

[1. Information (I) 2](#_Toc191402170)

[1.1 Projektdefinition 2](#_Toc191402171)

[1.2 Zielsetzung 3](#_Toc191402172)

[2. Planung (P) 3](#_Toc191402173)

[2.1 Arbeitspakete 3](#_Toc191402174)

[2.2 Ressourcen 4](#_Toc191402175)

[2.3 GANTT 5](#_Toc191402176)

[3. Entscheidung (E) 5](#_Toc191402177)

[3.1 Auswahl der Werkzeuge 5](#_Toc191402178)

[3.2 Begründung der Technologien 6](#_Toc191402179)

[3.3 Beispiel-Bilder 6](#_Toc191402180)

[4. Realisierung (R) 6](#_Toc191402181)

[4.1 Implementierung 6](#_Toc191402182)

[4.2 Besonderheiten und Herausforderungen 7](#_Toc191402183)

[4.3 Screenshot vom Prozess 8](#_Toc191402184)

[5. Kontrolle (K) 8](#_Toc191402185)

[5.1 Testprotokoll 8](#_Toc191402186)

[6. Auswertung (A) 8](#_Toc191402187)

[6.1 Fazit/Reflexion 8](#_Toc191402188)

[6.2 Wichtige Punkte 9](#_Toc191402189)

[7. Arbeitsaufteilung 9](#_Toc191402190)

[8. Code-Struktur 9](#_Toc191402191)

## 1. Information (I)

# 1.1 Projektdefinition

Die Firma Jetstream-Service führt als KMU in der Wintersaison verschiedene Ski-Servicearbeiten durch. Das bestehende Auftragsmanagement basierte bislang auf einer **relationalen Datenbank**, die jedoch mit steigender Standortzahl und dem erhöhten Datenvolumen an ihre Grenzen stößt.  
Ziel des Projekts ist die **Migration** des Backend-Teils des Auftragsmanagements auf eine **NoSQL-Datenbank** (MongoDB oder Neo4j) inklusive Anbindung an eine bereits bestehende Web-API. Dabei sollen die bisherigen Daten vollständig übernommen und Erweiterungen wie z. B. neue Stati (Offen, In Arbeit, Abgeschlossen) und Optionen zur Bearbeitung ermöglicht werden.

# 1.2 Zielsetzung

* **NoSQL-Datenbankdesign** und vollständige Übertragung (Migration) aller relevanten Daten aus dem bestehenden SQL-System.
* **Benutzerkonzept** mit mindestens zwei Benutzerrollen (z.B. Admin und Mitarbeiter), die unterschiedliche Berechtigungen besitzen.
* **Eingeschränkte Datenbankberechtigungen** für den Web-API-Benutzer (nur DML-Operationen).
* **Schema für Datenkonsistenz** (z. B. Validierung, Referenzen) und sinnvolle **Index-Strukturen** für performante Abfragen.
* **Backup & Restore**-Möglichkeiten sowie Skripte für die **vollständige Datenbankmigration**.
* **Web-API** (CRUD-Funktionen) auf die NoSQL-Datenbank anpassen.
* **Dokumentation** nach IPERKA, plus **Testprojekt** (z. B. Postman) zur Sicherstellung der Funktionsfähigkeit.
* **Git-Repository** zur Versionskontrolle und Projektverwaltung.

## 2. Planung (P)

# 2.1 Arbeitspakete

1. **Analyse & Entwurf (Datenbankdesign)**

* Auswahl des NoSQL-Systems (MongoDB oder Neo4j)
* Entwurf der Collections / Nodes und Beziehungen
* Festlegung der Felder (z. B. Kundenname, Dienstleistung, Status, Priorität)

1. **Migration & Skripte**

* Schreiben von Skripten zur Übertragung der Daten aus dem relationalen SQL-System
* Validierung der migrierten Daten
* Implementierung von Backup- und Restore-Prozessen

1. **Implementierung Web-API**

* Anpassung der bestehenden CRUD-Operationen
* Sicherstellung von Berechtigungen (Benutzerrollen)
* Einrichten spezieller DB-Zugänge mit eingeschränkten Rechten

1. **Test & Dokumentation**

* Erstellen einer Postman-Collection zur Validierung der API-Endpunkte
* Dokumentation (IPERKA)
* Vorbereitung der Kurzpräsentation mit Live-Demo

1. **Optionale Anforderungen** (zwei davon wählen)

* Automatisiertes Backup-Konzept (AO1)
* Komplexe Validierungen (AO2)

# 2.2 Ressourcen

* **Technologien & Werkzeuge**
* MongoDB oder Neo4j als Datenbank
* Web-API in Java, C# oder Python
* Postman für API-Tests
* Git für Versionsverwaltung
* **Personal**: 2 Entwickler bzw. kleines Team
* **Zeitbudget**: 12 Stunden
* **Hardware**: Server / Entwicklungsrechner mit installierter NoSQL-Datenbank

# 2.3 GANTT

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

KI-generierte Inhalte können fehlerhaft sein.

## 3. Entscheidung (E)

# 3.1 Auswahl der Werkzeuge

**1. NoSQL-Datenbank**:

* **MongoDB** (Dokumentenorientiert) oder **Neo4j** (Graph-basiert). MongoDB eignet sich gut für flexible Dokumentstrukturen und einfache Migrationen, Neo4j hingegen für Beziehungen in Graph-Form.

**2. Programmiersprache / Framework** für Web-API:

* Java (Spring Boot), C# (ASP.NET Core) oder Python (FastAPI, Flask, Django).

**3. Postman** als Testwerkzeug:

* Zur Automatisierung und Dokumentation aller API-Abfragen.

**4.**  **Git** (GitHub / GitLab / Bitbucket) für Versionsverwaltung.

# 3.2 Begründung der Technologien

* **NoSQL (MongoDB / Neo4j)**: Bessere Skalierbarkeit, horizontale Verteilung und Lizenzkostenersparnis.
* **Bekannte Webframeworks**: Hohe Community-Unterstützung und etablierte Best Practices.
* **Postman**: Standard-Tool für API-Test und einfache Integrationsprüfung.

# 3.3 Beispiel-Bilder



## 4. Realisierung (R)

# 4.1 Implementierung

**1. Datenbankdesign**

* Definition der Collections (z. B. orders, customers, users) in MongoDB oder der Nodes/Edges in Neo4j.
* Erstellung der notwendigen Indizes zur Performancesteigerung (z. B. Index auf email, status, priority).

**2. Datenmigration**

* Entwicklung und Ausführung von Skripten (z. B. mittels Python, Node.js oder MongoDB Database Tools) für den Transfer der Daten von SQL nach NoSQL.
* Überprüfung der Datenintegrität und Vollständigkeit.

**3. Migration & Anpassung der Web-API**

* Ersetzen der bisherigen SQL-Queries durch NoSQL-spezifische Zugriffe.
* Sicherstellung korrekter CRUD-Funktionen (Create, Read, Update, Delete) für Ski-Serviceaufträge.
* Integration von **Benutzerrollen** (Admin, Mitarbeiter) und tokenbasierte Authentifizierung (optional).

**4. Backup und Restore**

* Erstellung von Skripten oder Konfigurationsdateien, die regelmäßige Backups ermöglichen (z. B. via mongodump / neo4j-admin).
* Dokumentation der Restore-Schritte.

**5. Optionale Anforderungen**

* Z. B. **AO2**: Erweiterte Schema-Validierungen in MongoDB mittels JSON-Schema (z. B. enum für Status, min und max für Werte).
* Automatisiertes Backup-Konzept (AO1) via Cron-Jobs o. Ä.

# 4.2 Besonderheiten und Herausforderungen

* Umgang mit **verschiedenen Stati** (Offen, InArbeit, Abgeschlossen).
* **Skalierbarkeit**: Verteilen der Datenbank auf mehrere Instanzen.
* Einhaltung der **Datenkonsistenz** trotz der flexibleren Strukturen von NoSQL.
* Anpassung des **Benutzerkonzepts** (Rollen und Rechte).

# 4.3 Screenshot vom Prozess

Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

KI-generierte Inhalte können fehlerhaft sein.

## 5. Kontrolle (K)

# 5.1 Testprotokoll

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test-ID** | **Funktion** | **Beschreibung** | **Ergebnis** | **Status** |
| 001 | Datenbankverbindung | Verbindung zur NoSQL-DB (Login, Lese-/Schreibtest) | OK | Bestanden |
| 002 | Datenmigration | Vollständige Übernahme der Daten (SQL -> NoSQL) | OK | Bestanden |
| 003 | CRUD-Aufträge | Erstellen, Lesen, Updaten, Löschen eines Auftrags | OK | Bestanden |
| 004 | Benutzerrollen (Admin, Mitarbeiter) | Rechteverteilung überprüft | OK | Bestanden |
| 005 | Backup & Restore | Skript testweise ausgeführt, Daten restauriert | OK | Bestanden |

## 6. Auswertung (A)

# 6.1 Fazit/Reflexion

Das Projekt **Ski-Service Auftragsverwaltung** wurde erfolgreich auf eine NoSQL-Datenbank migriert. Alle wesentlichen Anforderungen (A1–A10) sowie zwei optionale Anforderungen wurden umgesetzt. Die Datenmigration verlief reibungslos und alle CRUD-Operationen sind funktional. Durch die Verwendung von **Indizes**, **Validierungsregeln** und **Backup-/Restore-Skripten** ist die Anwendung zukunftssicher und skalierbar.

# 6.2 Wichtige Punkte

* **Skalierbarkeit** & **Performance** profitieren von der NoSQL-Lösung.
* **Automatisiertes Backup** erhöht die Ausfallsicherheit.
* In Zukunft wäre es denkbar, weitere **statistische Auswertungen** (z. B. über Auftragshäufigkeit pro Saison, Standorte, etc.) zu integrieren oder ein **komplexeres Rollen-/Rechtemanagement** einzuführen.

## 7. Arbeitsaufteilung

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Aufteilung der Aufträge im Team | | |  | |
| Ertem: | Gemeinsam: | Tuna: | |
| Dokumentation | Präsentieren | Datenbankdesign | |
| Test-Protokoll | Fazit/Reflexion | Funktionen | |
| GANTT | Powerpoint | Datenbankkonfiguration | |
| Code | Ausblick | Testing | |
| Arbeitsaufteilung | Live-Demo | Optionale Anforderungen | |
|  | Kommunizieren | Backup & Restore | |
|  | Kontrolle |  | |

## 8. Code-Struktur

Ein Bild, das Text, Screenshot, Software, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.