Raphael Hug

01.02.2023

Modul 165 Projektarbeit

Jetstream Ski Service

Inhalt

[1. Informieren 2](#_Toc126160333)

[1.1. Ausgangslage 2](#_Toc126160334)

[1.2. Anforderungen 2](#_Toc126160335)

[1.3. Zusätzliche Anforderungen 2](#_Toc126160336)

[1.4. Was muss ich sonst noch wissen 3](#_Toc126160337)

[2. Planen 3](#_Toc126160338)

[2.1. NoSQL Datenbanken 3](#_Toc126160339)

[2.2. Vorgangsweise 3](#_Toc126160340)

[2.3. Zeitplanung und PSP 4](#_Toc126160341)

[3. Entscheiden 6](#_Toc126160342)

[3.1. Welche NoSQL Datenbank 6](#_Toc126160343)

[3.2. Tools und Voraussetzungen/Versionen 7](#_Toc126160344)

[3.3. Aufbau 7](#_Toc126160345)

[4. Realisieren 8](#_Toc126160346)

[4.1. Datenbanken 8](#_Toc126160347)

[4.1.1. Migration 8](#_Toc126160348)

[4.1.2. Schema 8](#_Toc126160349)

[4.1.3. Erstellen 8](#_Toc126160350)

[4.1.4. User erstellen 8](#_Toc126160351)

[4.1.5. Backup 8](#_Toc126160352)

[4.1.6. Restore 8](#_Toc126160353)

[4.2. Model 8](#_Toc126160354)

[4.3. Controller 8](#_Toc126160355)

[4.4. Service 8](#_Toc126160356)

[4.5. JWT 8](#_Toc126160357)

[5. Kontrollieren 8](#_Toc126160358)

[5.1. Checkliste 8](#_Toc126160359)

[5.2. Postmann 8](#_Toc126160360)

[5.3. Frontend WPF 8](#_Toc126160361)

[6. Auswertung 8](#_Toc126160362)

[6.1. Sol ist vergleich 8](#_Toc126160363)

[6.2. Fazit 8](#_Toc126160364)

[6.3. Dokumentation 8](#_Toc126160365)

# Informieren

## Ausgangslage

Die Firma Jetstream-Service führt als KMU in der Wintersaison Skiservicearbeiten durch und hat in den letzten Jahren grosse Investitionen in eine durchgängige digitale Auftragsanmeldung und Verwaltung, bestehend aus einer datenbankbasierender Web-Anmeldung und Auftragsverwaltung getätigt. Aufgrund guter Auftragslage hat sich die Geschäftsführung für eine Diversifizierung mit Neueröffnungen an verschiedenen Standorten entschieden. Die bis anhin eingesetzte relationale Datenbank genügt den damit verbundenen Ansprüchen an Datenverteilung und Skalierung nicht mehr. Um einerseits den neuen Anforderungen gerecht zu werden sowie anderseits Lizenzkosten einzusparen, soll im Backend der Anwendung die Datenbank auf ein NoSQL Datenbanksystem migriert werden.

## Anforderungen

|  |  |
| --- | --- |
| Nr. | Beischreibung |
| A1 | Datenbasis aus relationaler Datenbank vollständig nach NoSQL migriert |
| A2 | Benutzerkonzept mit min. 2 Benutzeranmeldungen mit verschiedenen Berechtigungsstufen implementiert. |
| A3 | Für die Web-API Applikation muss ein eigener Datenbankbenutzerzugang mit eingeschränkter Berechtigung (DML) zur Verfügung gestellt werden |
| A4 | Schema für Datenkonsistenz implementiert |
| A5 | Datenbank Indexe für schnelle Ausführung von Suchabfragen implementiert |
| A6 | Backup und Restore Möglichkeiten umgesetzt (Skript-Dateien) |
| A7 | Vollständige Datenbankmigration mittels Skript-Dateien realisiert |
| A8 | Das Web-API-Projekt (CRUD) komplett auf NoSQL Datenbanksystem migriert |
| A9 | Datenmodell vollständig dokumentiert, inkl. Grafik zum Datenmodell |
| A10 | Einfaches Testprojekt in Postman erstellt. |
| A11 | Das Softwareprojekt ist über ein Git-Repository zu verwalten |
| A12 | Ganzes Projektmanagement muss nach IPERKA dokumentiert sein |

## Zusätzliche Anforderungen

|  |  |
| --- | --- |
| Nr. | Beschreibung |
| AO1 | Automatisiertes Backup-Konzept durchgeführt u. implementiert. |
| AO2 | Komplexe Schema Validierungen umgesetzt (Referenzen, enum, min, max. usw.) |
| AO3 | Datenmigrationsskripte zu den RDBMS nach NoSQL realisiert |
| AO4 | Komplexes Datenmodell mit mehr als 6 Grundtypen (Collection / Labels) implementiert |
| AO5 | Komplexe statistische Auswertungsabfragen realisiert |

## Was muss ich sonst noch wissen

Da wir schon eine API mit MS SQL gemacht haben, wusste ich wie man eine API aufbaut.

Was ich mich noch informieren musste, wie man daten vom SQL zu NoSQL migriert.

Wie man NoSQL in C# für Möglichkeiten gibt, da wir kein Entity Framework wie bei der anderen Projektarbeit haben

# Planen

## NoSQL Datenbanken

Welche NoSQL Datenbank soll ich verwenden. In der Schule haben wir 2 genauer angeschaut:

Dokument: MongoDB

Graph: Neo4j

Ich muss mich für eine der beiden entscheiden, da werde ich beim Entscheiden werde eine ausführliche Recherche machen.

## Vorgangsweise

Bei der Datenbank, wie soll, das Schema aussehen brauche ein Schema und wenn man eins braucht, was sind Pflichtfelder, was soll ein String sein.

Bei dem C# Projekt, wasfür Controllers und Anzahl Models brauche ich und wie mache ich einen JWT-Token mit NoSQL Datenbanken.

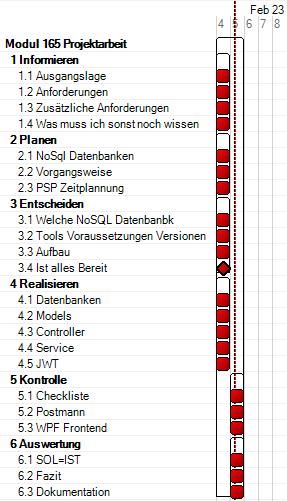
Ich habe mir am Anfang einfach Gedanken gemacht über den ganzen Aufbau des Codes und der Datenbank.

Darauf werde ich beim Aufbau zurückkommen

## Zeitplanung und PSP

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung



# Entscheiden

## Welche NoSQL Datenbank

#### MongoDB:

MongoDB ist eine der beliebtesten Open-Source-No-SQL-Datenbanken. MongoDB ist nicht relational und kann ein dynamisches Schema haben, das es Benutzern ermöglicht, die Daten in MongoDB einzufügen, ohne das Schema zu definieren.

In MongoDB können Benutzer auch das Schema definieren, um Benutzer zu zwingen, Daten mit bestimmten Feldern einzufügen. Es kann problemlos operative, transaktionale und analytische Workload bewältigen.

MongoDB verwendet das BSON-Format (das JSON ähnlich ist), um die Daten darzustellen, und ist gemäss den eigenen Anforderungen leicht skalierbar. Sie können mit MongoDB entweder über die Befehlszeile oder über einen Atlas (eine Benutzeroberfläche auf MongoDB) interagieren.

MongoDB bietet auch eine Unternehmensversion, die mehr Funktionen und Unternehmensunterstützung bietet als die Open-Source-MongoDB-Version.

#### Neo4j:

Neo4j ist eine der beliebtesten Graph Datenbanken. Es ist Open-Source-Software. Eine Graf-Datenbank zeigt die Beziehungen zwischen Daten auf grafische Weise.

Neo4j speichert die Daten in den Knoten, deren Natur gerichtet ist. Es ist eine Grafendatenbank, die auf Grafentheorie basiert, im Gegensatz zu der traditionellen Datenbank, die auf Logik und Mengentheorie basiert.

Neo4j ist eine NoSQL-Datenbank, in der Daten als Knoten (Entitäten) und Kanten (Beziehungen) dargestellt werden und jeder Knoten oder jede Kante eine oder mehrere Eigenschaften (Attribute) haben kann.

#### Meine Wahl:

Das ist MongoDB, weil ich die Syntax in C# kenne und ich für die Daten, wo ich habe, keine grafische Oberfläche brauche. Es hat nicht so viele Referenzen, dass sich Neo4j lohnen würde. Im MongoDB kann ich auch ein Schema erstellen, mit dem bestimmten Enum, wo ich brauche, dass kein Fehler in Datenbank machen könnte.  
Deshalb habe ich mich für MongoDB entschieden.

## Tools und Voraussetzungen/Versionen

#### Was habe ich benutzt:

#### Rechner

Betriebsystemname Microsoft Windows 11 Pro

Version 10.0.22621 Build 22621

Betriebsystemhersteller Microsoft Corporation

Systemname PCHOME

Systemtyp x64-basierter PC

System-SKU SKU

Prozessor AMD Ryzen 5 5600X 6-Core Processor, 3901 MHz, 6 Kern(e), 12 logische(r)

BIOS-Version/-Datum American Megatrends Inc. 4403, 27.04.2022

SMBIOS-Version 3.3

Version des eingebetteten Controllers 255.255

BIOS-Modus UEFI

BaseBoard-Hersteller ASUSTeK COMPUTER INC.

BaseBoard-Produkt ROG STRIX X570-E GAMING

Windows-Verzeichnis C:\Windows

Systemverzeichnis C:\Windows\system32

Hardwareabstraktionsebene Version = "10.0.22621.819"

Benutzername PCHome\raphi

Gesamter physischer Speicher 63.9 GB

#### IDE

Visual Studio 2022 Version 17.4.3

#### SDK

.Net 6

#### NuGet Pakete

Microsoft.AspNetCore.Authentication.JwtBearer Version 6.0.13

MongoDB.Driver Version 2.18.0

Serilog.AspNetCore Version 6.1.0

Swashbuckle.AspNetCore Version 6.2.3

#### MS-SQL

OBCD neuste Version

## Aufbau

Also ich habe mir gedacht zuerst alles zu machen, was mit der Datenbank zu tun, hat Migrieren von MS-SQL zu einer JSON-Datei umgewandelt genauer werde ich beim unter Kapitel Migrieren genauer erläutern.

Danach habe ein Schema erstellt und die migrierte JSON-File noch das Create und Pickup verändern, dass es wirklich ein Datum ist. Ich möchte ich dir das File einlesen. Dann will ich einen User erstellen, der nur auf die bestimmte Datenbank Lese und Schreibrechte hat.

Danach kommt das Backup und den Restore dazu.

Nach der Datenbank will ich ein neues Projekt auf .Net 6 API erstellen und dort das Model und die Controller erstellen und nachdem will ich noch den Service erstellen.

Wen dann alles funktioniert hat, will ich dann einem JWT-Model, Controller und der Service erstellen.

Natürlich im Main Programm noch alles einbauen.

Nachdem will ich noch einen Logger zu machen, wo alle Störungen in einem File speichert und das wird täglich aktualisiert.

Jetzt kommen wir zum Testen, ich will es mit Postman Testen und auch noch mit meiner WPF Frontend.

# Realisieren

## Datenbanken

## Migration

Ich habe von der Datenbank Ski Service mit Inner Join mit 3 Tabellen eine View gemacht.

Die View habe ich dann in einer JSON-Datei gespeichert. Wen man mit OBCD auf den Server hat, kann es Exportieren in einem, sonst kann man einfach das File Kopieren und dann ein JSON-File erstellen und einfügen

## Schema

Beim Schema habe ich mir Gedanken gemacht, was brauche, ich also habe alles aus Kommentar als Pflichtfelder angeben, aber beim Kommentar habe ich eingeben, dass das es einen String muss sein.

Und habe auch bei jeden andere Feld String als Pflicht ausser bei Create und Pickup date.

Dort habe ich ein Datum als Pflicht angeben.

Und ein enum für Service, Status, Priority, das ist beim Script beschrieben, was meine enum sind.

## Erstellen

Ich habe dann das JSON-File von der Datenbank mit einem Mongoimport erstellt und habe auch noch ein Beispiel gemacht, wie es auch ohne Mongoimport machen kann.

## User erstellen

Ich habe dort einen User erstellt, der die rechte auf der Datenbank mit dem Collection Registration und User Lese und Schreibrechte hat.

## Backup

Ich habe dort Atlas erstellt dort macht es jede 72 stunden selbstständig gemacht.

Aber da es eine Anforderung ist, habe ich einfach ein Script geschrieben und die Daten auf dem Laufwerk D (Daten) gespeichert.

## Restore

Dort habe ich das Backup File mit drop wiedergeholt.

## Model

Ich habe dort ein Model gemacht w o alles, wo die Datenbank braucht, drinsteht.

Und mit [JsonPropertyName("")] ist mein DTO dort sieht es drin was man eingeben muss beim Update oder beim Post.

Was habe ich alles drin:

String \_id

String name

String Email

String Phone

String Service

String Status

String Priority

String Komentar

Date Pickup

Date Create

## Controller

In dem Controller habe ich eine HTTP-Methode mit den Get, Get(id), Post, Update(id) und Delete.

Jetzt kommt dann den Service, bevor ich dort weiter machen kann, aber ich muss einfach auf den Service verweisen. Und natürlich im Nachhinein das Interface von Service Verweisen.

Was bei mir noch speziell ist beim Löschen wird der Status auf Gelöscht verwiesen und wird dann ein Update vom Service gemacht wird. Bei mir wird nichts gelöscht, sondern nur Status = gelöscht.

## Service

Beim Service habe ich ein Interface erstellt und dann eine Klasse, die auf dem Interface vererbt.

Dort habe alle MongoDB Statements gemacht, was noch Besonderes ist, ich habe dort GetALL einfach noch die alle ausser, die wo den Status gelöscht sind also habe ich dort ein Index in der Datenbank gemacht.

## JWT

Ich habe dort ein Model gemacht, einen Service mit Interface und natürlich auch ein Controller.

Ich musste dann dort im Main Programm noch müssen Verbesserungen zu machen.

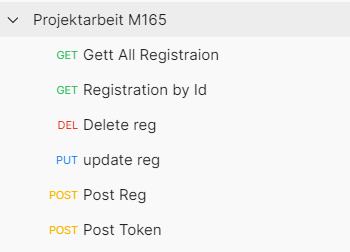
# Kontrollieren

## Checkliste

|  |  |  |
| --- | --- | --- |
| Nr. | Beischreibung |  |
| A1 | Datenbasis aus relationaler Datenbank vollständig nach NoSQL migriert | x |
| A2 | Benutzerkonzept mit min. 2 Benutzeranmeldungen mit verschiedenen Berechtigungsstufen implementiert. | x |
| A3 | Für die Web-API Applikation muss ein eigener Datenbankbenutzerzugang mit eingeschränkter Berechtigung (DML) zur Verfügung gestellt werden | x |
| A4 | Schema für Datenkonsistenz implementiert | x |
| A5 | Datenbank Indexe für schnelle Ausführung von Suchabfragen implementiert | x |
| A6 | Backup und Restore Möglichkeiten umgesetzt (Skript-Dateien) | x |
| A7 | Vollständige Datenbankmigration mittels Skript-Dateien realisiert | x |
| A8 | Das Web-API-Projekt (CRUD) komplett auf NoSQL Datenbanksystem migriert | x |
| A9 | Datenmodell vollständig dokumentiert, inkl. Grafik zum Datenmodell | x |
| A10 | Einfaches Testprojekt in Postman erstellt. | x |
| A11 | Das Softwareprojekt ist über ein Git-Repository zu verwalten | x |
| A12 | Ganzes Projektmanagement muss nach IPERKA dokumentiert sein | x |

|  |  |  |
| --- | --- | --- |
| Nr. | Beschreibung |  |
| AO1 | Automatisiertes Backup-Konzept durchgeführt u. implementiert. | Atlas |
| AO2 | Komplexe Schema Validierungen umgesetzt (Referenzen, enum, min, max. usw.) | x |
| AO3 | Datenmigrationsskripte zu den RDBMS nach NoSQL realisiert |  |
| AO4 | Komplexes Datenmodell mit mehr als 6 Grundtypen (Collection / Labels) implementiert |  |
| AO5 | Komplexe statistische Auswertungsabfragen realisiert |  |

## Postman



Dort habe einfach getestet und war erfolgreich

## Frontend WPF

Ein Bild, das Tisch enthält.

Automatisch generierte Beschreibung

Auch dort habe ich gesteht und auch hatte keine Probleme

# Auswertung

## Sol ist vergleich

Geplante Stunden ist 28h, aber ich schaffte das in 16h war kurzfristiger, also ich geplant habe.

Weil die Vorlage, die wir haben, hervorragend war, ich musste halt alles ein wenig verbessern.

## Fazit

Ich finde es spannend, wie man so einfach MongoDB mit C# machen kann.

Und was die grösste Schwierigkeit war der JWT-Token, ich wusste einfach nicht mehr, was alles braucht, aber es war eine kleine Auffrischung.

## Dokumentation

Ich muss jetzt nur noch meine Dokumentation fertigstellen und auch meine PowerPoint.

Ich weiss manchmal nicht ganz was in der Dokumentation soll, ich habe dort noch Probleme.