ipso! Bildung seit 1876

2/2/2024

NoSQL Ski Service Manager

Modul 165

Arda Dursun
IPSO BILDUNG AG

ipso! Bildung seit 1876

Inhaltsverzeichnis

1 Einleitung

1.1 Projektübersicht

Das Projekt "NoSQL Ski Service Manager" ist eine moderne Backend-Anwendung, entwickelt, um ein bestehendes Skiservice-Auftragsverwaltungssystem von einer relationalen zu einer NoSQL-Datenbank zu migrieren. Diese Umstellung ermöglicht es dem System, den wachsenden Anforderungen an Flexibilität, Datenverteilung und Skalierbarkeit gerecht zu werden. Ziel des Projekts ist es, eine effiziente und kosteneffektive Lösung zu implementieren, die sich nahtlos in bestehende Frontend-Systeme integrieren lässt und eine robuste API-Schnittstelle für die Verwaltung von Serviceaufträgen bietet. Die Kernkomponenten umfassen das Datenbankdesign, die Datenmigration, die Entwicklung der WebAPI und umfassende Tests zur Sicherstellung der Funktionalität und Leistung.

1.1.1 Projekthighlights

Bei der ersten Ausführung des "NoSQL Ski Service Manager"-Projekts werden alle notwendigen Daten automatisch geseedet und Schemas sowie Indizes erstellt, was eine einsatzbereite Datenbankumgebung gewährleistet. Die Installation des .NET 8.0 SDK und des MongoDB Servers (Version 7.0 oder höher) ist erforderlich. Postman für API-Tests ist optional. Die Architektur nutzt generische Controller und Services zur Vereinfachung der CRUD-Operationen, unterstützt durch AutoMapper für effiziente Objektzuweisungen. Spezielle Anforderungen, wie Backup und Restore, sind durch JavaScript-Skripte abgedeckt, die meisten Setup-Prozesse sind jedoch automatisiert, um den Einstieg zu erleichtern.

1.2 Zielsetzung

Das primäre Ziel des "NoSQL Ski Service Manager" ist die Entwicklung eines leistungsfähigen Backend-Systems zur Verwaltung von Skiservice-Aufträgen mit einer NoSQL-Datenbank. Dabei sollen die Vorteile von NoSQL, wie hohe Skalierbarkeit und Flexibilität in der Datenstrukturierung, genutzt werden, um eine verbesserte Performance und Wartbarkeit gegenüber dem vorherigen relationalen System zu erreichen. Der Nutzen des Projekts erstreckt sich auf eine vereinfachte Datenverwaltung, eine gesteigerte Anfrageeffizienz und eine zukunftssichere Plattform, die eine schnelle Anpassung an neue Geschäftsanforderungen ermöglicht. Darüber hinaus zielt das Projekt darauf ab, durch die Implementierung eines intuitiven API-Designs die Integration in bestehende und neue Frontend-Systeme zu erleichtern, was eine breite Nutzerakzeptanz sicherstellen soll.

1.3 Zielgruppe

Der "NoSQL Ski Service Manager" richtet sich als Schulprojekt primär an den Dozenten der Lehre für Applikationsentwickler EFZ. Das Projekt dient als Nachweis der erlernten Kompetenzen in der Entwicklung und Verwaltung von NoSQL-basierten Anwendungen. Als sekundäre Zielgruppe sind zukünftige Softwareentwickler und Studierende anzusehen, die anhand dieses Projekts praktische Anwendungsbeispiele und moderne Entwicklungspraktiken kennenlernen sollen. Das Projekt soll als Lehrmittel dienen und den Studierenden ermöglichen, tiefergehende Einblicke in die Arbeit mit nicht-relationalen Datenbanken und die Implementierung von Backend-Systemen zu gewinnen.

2 Informieren

2.1 Technologieüberblick

Der "NoSQL Ski Service Manager" stützt sich auf eine Kombination aus modernen Technologien und Frameworks, um eine hochperformante und flexible Backend-Lösung zu gewährleisten. Kern der Anwendung ist die Nutzung von MongoDB, einer leistungsfähigen NoSQL-Datenbank, die eine hohe Skalierbarkeit und eine flexible Datenstrukturierung ermöglicht. Die Anwendung wird in .NET 8.0 entwickelt und bietet durch den Einsatz von RESTful API-Konzepten eine klare Schnittstelle für Frontend-Anwendungen. Für die Objektmappung wird AutoMapper verwendet, und für Authentifizierungszwecke JWT (JSON Web Tokens). Tests werden mit Postman durchgeführt, um sicherzustellen, dass alle Endpoints funktionieren. Serilog dient als Logging-Framework zur Überwachung und Fehlerbehebung, während Swashbuckle die Erstellung von Swagger-Dokumentation für die API erleichtert.

2.2 Anforderungsüberblick

Das Projekt "NoSQL Ski Service Manager" wurde konzipiert, um die Anforderungen der Datenmigration von einem relationalen zu einem NoSQL-Datenbanksystem zu erfüllen. Im Zentrum stehen dabei die Implementierung eines Benutzerkonzepts mit verschiedenen Berechtigungsstufen, die Sicherstellung der Datenkonsistenz durch Schemata und die Gewährleistung schneller Suchabfragen mittels Datenbankindizes. Die Anwendung ermöglicht Backup und Restore über Skript-Dateien und stellt sicher, dass CRUD-Operationen auf das NoSQL-System übertragen werden. Darüber hinaus wird das Datenmodell detailliert dokumentiert. Optional wurden ein automatisiertes Backup-Konzept und komplexe Schema-Validierungen integriert, und das Projektmanagement folgt dem IPERKA-Ansatz.

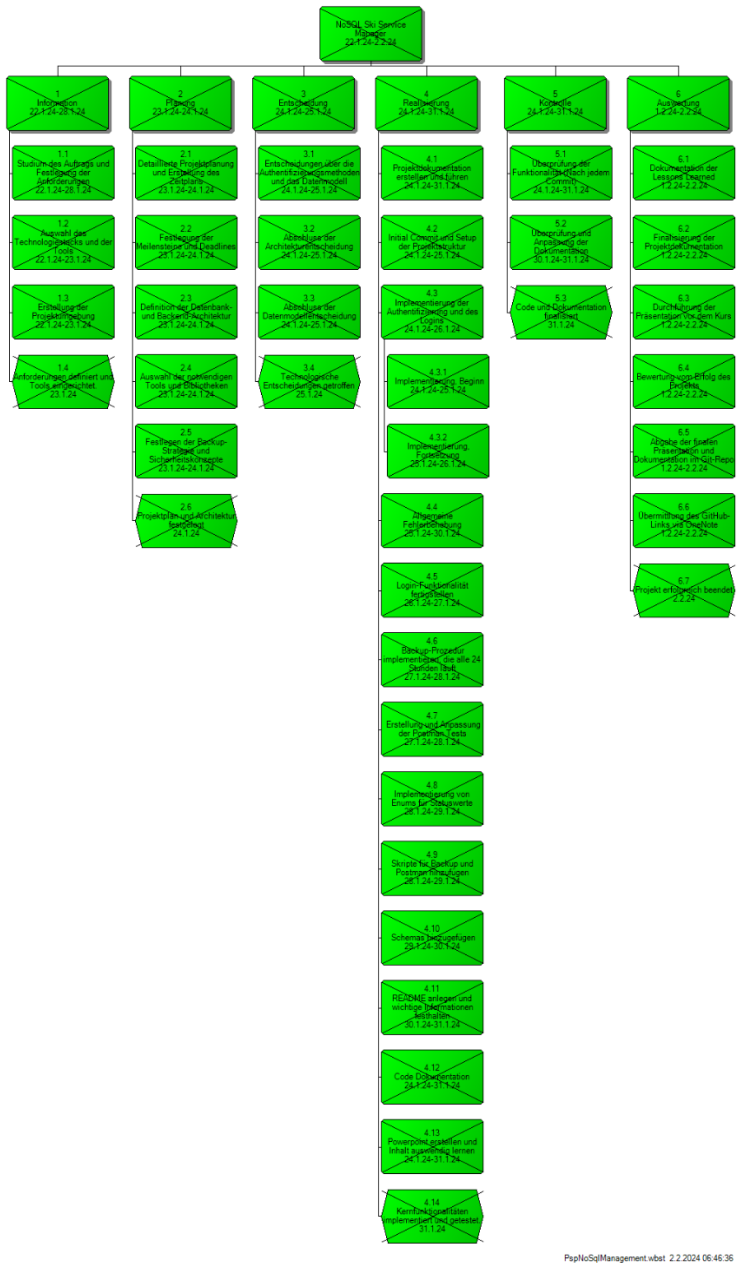
2.3 Risikoanalyse

Risikoanalyse NoSQL Ski Service Manager

Risikotyp	Nr.	Wahrsch.	Auswirk.	Ampel	Beschreibung	Behandlung und Kontrolle
Risikomanagement NoSQL Projekt						
Datenverlust	1	3	4	12	Kann durch unzureichende Backup-Verfahren oder Fehler beim Handling der Datenbank entstehen. Schaden wäre der Verlust von wertvollen Daten und Zeit für die Wiederherstellung oder Rekonstruktion.	Regelmässige Backups, Datenvalidierung
Sicherheitsrisiken	2	2	4	8	Mögliche Verletzungen der Datensicherheit durch Hacking oder interne Lecks.	Sicherheitsprotokolle, Zugriffsrechteverwaltung
Technische Defekte	3	2	3	6	Risiko von Ausfällen durch Hardware- oder Softwarefehler.	Redundanzplan, regelmässige Wartung
Performance-Probleme	4	3	3	9	Risiko von Leistungsengpässen, die Benutzererfahrungen beeinträchtigen können.	Performance-Monitoring, Skalierungsoptionen
Fehlende Dokumentation	5	3	2	6	Das Risiko, dass mangelnde Dokumentation zu Verwirrung und Fehlern führt.	Dokumentationsrichtlinien, regelmässige Reviews
Projektverzögerungen	6	3	3	9	Risiken, die zu Verzögerungen im Projektzeitplan führen	Zeitmanagement, Meilenstein-Planung
Veraltete Technologie	7	1	2	2	Die Gefahr, dass veraltete Technologien die Effizienz und Sicherheit beeinträchtigen	Technologie-Review-Zyklen, Weiterbildung
Unzureichendes Testing	8	3	3	9	Das Risiko, dass durch unzureichende Tests Fehler in der Produktion auftreten.	Umfassende Testpläne, automatisierte Tests
Anforderungsänderungen	9	0	2	0	Risiko von Projektkomplikationen durch sich ändernde Anforderungen.	Change-Management-Prozesse, agile Methodik
Kommunikationsprobleme	10	0	3	0	Risiko von Missverständnissen und Fehlern aufgrund schlechter Kommunikation.	Klare Kommunikationskanäle, regelmässige Meetings (Trello, Miro usw.)

In meiner Risikoanalyse für das NoSQL Ski Service Manager-Projekt habe ich verschiedene Risiken identifiziert, die das Projekt beeinträchtigen könnten. Ich habe Wahrscheinlichkeiten und mögliche Auswirkungen dieser Risiken bewertet, um eine Prioritätenliste zu erstellen und entsprechende Massnahmen zur Minderung zu ergreifen. Zu meinen proaktiven Schritten gehören die Durchführung regelmässiger Backups, die Implementierung von Sicherheitsprotokollen, das Aufsetzen eines Redundanzplans und die kontinuierliche Wartung der technischen Infrastruktur. Weiterhin habe ich durch Performance-Monitoring und Skalierungsoptionen das Risiko von Leistungseinbussen minimiert und durch Einführung von Dokumentationsrichtlinien und regelmässigen Reviews die Dokumentationsqualität verbessert. Diese Massnahmen tragen dazu bei, das Projekt gegenüber unvorhergesehenen Ereignissen zu stärken und die Projektziele effektiv zu erreichen.

3 Planen



In der Projektplanungsphase, die im Rahmen des IPERKA-Modells durchgeführt wurde, habe ich einen umfassenden Planungsprozess eingeleitet, der mit der Analyse der Projektanforderungen begann. Diese Analyse beinhaltete das Studium des Auftrags und die Festlegung der technischen und funktionalen Anforderungen der Backend-Anwendung. Die Technologieauswahl wurde getroffen, wobei MongoDB aufgrund seiner Skalierbarkeit und Flexibilität als Datenbanksystem ausgewählt wurde. Der Einsatz von .NET 8.0 sowie die Integration von JWT für Authentifizierungszwecke wurden beschlossen. Zur Unterstützung des Entwicklungsprozesses wurden Tools wie Swagger für die API-Dokumentation und Postman für das Testen der Endpunkte eingeplant. Ein detaillierter Zeitplan wurde erstellt, der alle wesentlichen Aktivitäten und Meilensteine umfasst. Dieser Zeitplan diente als Richtlinie für die Realisierungsphase und gewährleistete, dass alle Teile des Projekts fristgerecht fertiggestellt werden.

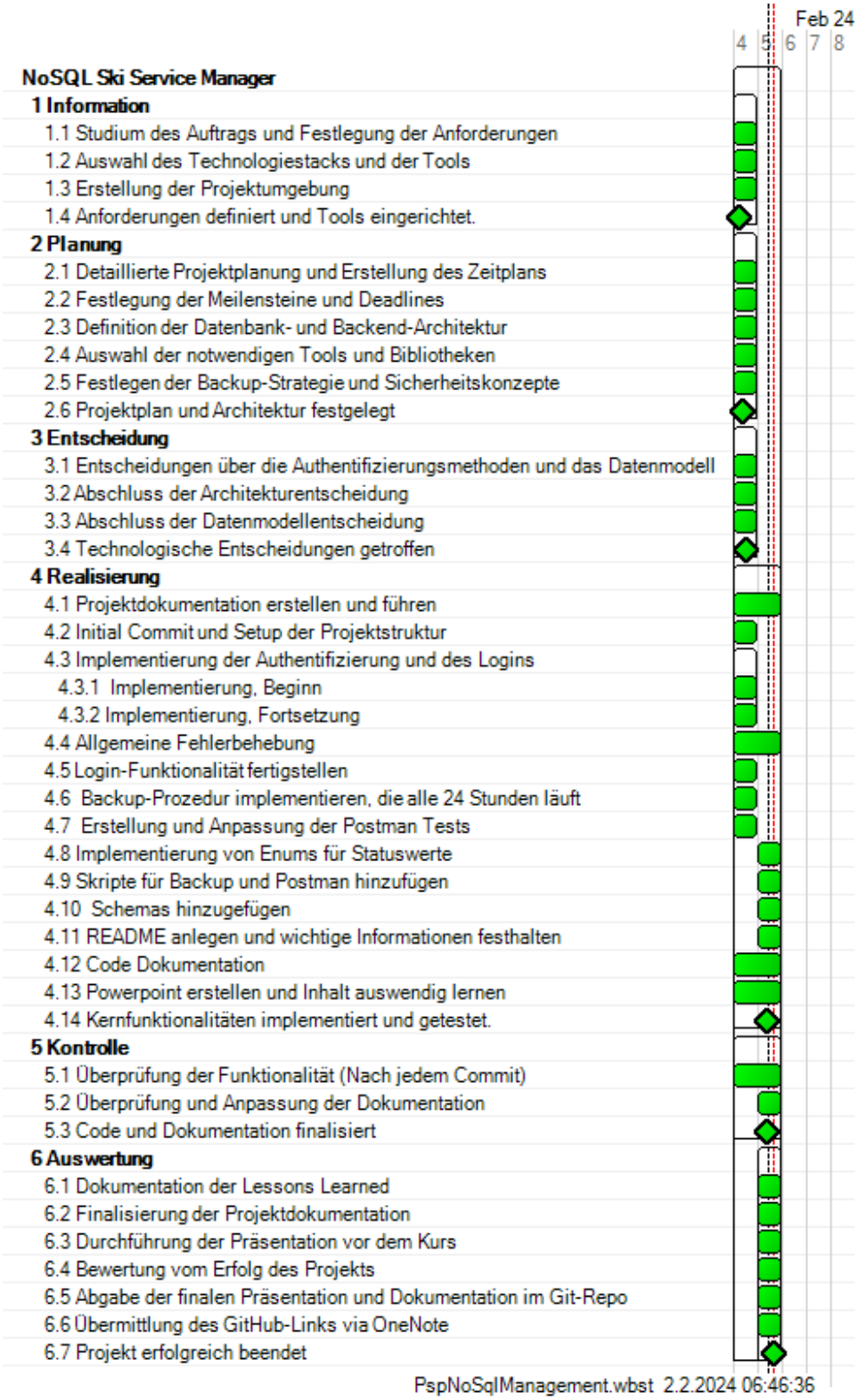
Die Ressourcenplanung berücksichtigte die verfügbaren Kapazitäten und Expertisen. Es wurde sichergestellt, dass ausreichend Zeit für die Implementierung, das Testing und die Dokumentation zur Verfügung stand. Die Kostenplanung erfolgte sorgfältig, um die Wirtschaftlichkeit des Projekts zu gewährleisten.

Insgesamt sorgte die sorgfältige Planung dafür, dass das Projekt auf einem soliden Fundament aufgebaut wurde, was die Grundlage für den erfolgreichen Abschluss des "NoSQL Ski Service Manager"-Projekts bildete.

3.1 Ressourcenplanung

	PSP-Nr.	Beschreibung	Verantwortlich	Plan Arbeit h	Ist-Arbeit bisher h	Rest-Arbeit h	Ausblick Arbeit h	Abw. %	Plan ext. Kosten CHF	Plan Gesamtkosten CHF
▸		**NoSQL Ski Service Manager**		**70.25**	**82.00**	**0.00**	**82.00**	**+17%**	**0**	**5'620**
	1	**Information**		**3.00**	**3.50**	**0.00**	**3.50**	**+17%**	**0**	**240**
	1.1	Studium des Auftrags und Festlegung der Anforderungen		1.00	1.00	0.00	1.00	+0%	0	80
	1.2	Auswahl des Technologiestacks und der Tools		1.00	1.00	0.00	1.00	+0%	0	80
	1.3	Erstellung der Projektumgebung		1.00	1.50	0.00	1.50	+50%	0	80
	1.4	Anforderungen definiert und Tools eingerichtet.		0.00	0.00	0.00	0.00	+0%	0	0
	2	**Planung**		**8.50**	**8.75**	**0.00**	**8.75**	**+3%**	**0**	**680**
	2.1	Detaillierte Projektplanung und Erstellung des Zeitplans		1.00	3.00	0.00	3.00	+200%	0	80
	2.2	Festlegung der Meilensteine und Deadlines		1.00	1.00	0.00	1.00	+0%	0	80
	2.3	Definition der Datenbank- und Backend-Architektur		2.00	3.00	0.00	3.00	+50%	0	160
	2.4	Auswahl der notwendigen Tools und Bibliotheken		2.50	0.75	0.00	0.75	-70%	0	200
	2.5	Festlegen der Backup-Strategie und Sicherheitskonzepte		2.00	1.00	0.00	1.00	-50%	0	160
	2.6	Projektplan und Architektur festgelegt		0.00	0.00	0.00	0.00	+0%	0	0
	3	**Entscheidung**		**4.00**	**4.00**	**0.00**	**4.00**	**+0%**	**0**	**320**
	3.1	Entscheidungen über die Authentifizierungsmethoden und das Datenmodell		2.00	1.50	0.00	1.50	-25%	0	160
	3.2	Abschluss der Architekturentscheidung		1.00	0.50	0.00	0.50	-50%	0	80
	3.3	Abschluss der Datenmodellentscheidung		1.00	2.00	0.00	2.00	+100%	0	80
	3.4	Technologische Entscheidungen getroffen		0.00	0.00	0.00	0.00	+0%	0	0
	4	**Realisierung**		**37.50**	**46.25**	**0.00**	**46.25**	**+23%**	**0**	**3'000**
	4.1	Projektdokumentation erstellen und führen		8.00	7.00	0.00	7.00	-13%	0	640
	4.2	Initial Commit und Setup der Projektstruktur		2.00	3.00	0.00	3.00	+50%	0	160
	4.3	Implementierung der Authentifizierung und des Logins		3.00	6.00	0.00	6.00	+100%	0	240
	4.3.1	Implementierung, Beginn		1.00	3.00	0.00	3.00	+200%	0	80
	4.3.2	Implementierung, Fortsetzung		2.00	3.00	0.00	3.00	+50%	0	160
	4.4	Allgemeine Fehlerbehebung		5.00	9.00	0.00	9.00	+80%	0	400
	4.5	Login-Funktionalität fertigstellen		5.00	5.00	0.00	5.00	+0%	0	400
	4.6	Backup-Prozedur implementieren, die alle 24 Stunden läuft		1.00	1.00	0.00	1.00	+0%	0	80
	4.7	Erstellung und Anpassung der Postman Tests		1.00	1.00	0.00	1.00	+0%	0	80
	4.8	Implementierung von Enums für Statuswerte		2.00	2.00	0.00	2.00	+0%	0	160
	4.9	Skripte für Backup und Postman hinzufügen		1.00	1.00	0.00	1.00	+0%	0	80
	4.10	Schemas hinzugefügen		3.00	5.00	0.00	5.00	+67%	0	240
	4.11	README anlegen und wichtige Informationen festhalten		3.00	2.75	0.00	2.75	-8%	0	240
	4.12	Code Dokumentation		2.50	2.50	0.00	2.50	+0%	0	200
	4.13	Powerpoint erstellen und Inhalt auswendig lernen		1.00	1.00	0.00	1.00	+0%	0	80
	4.14	Kernfunktionalitäten implementiert und getestet.		0.00	0.00	0.00	0.00	+0%	0	0
	5	**Kontrolle**		**12.00**	**13.00**	**0.00**	**13.00**	**+8%**	**0**	**960**
	5.1	Überprüfung der Funktionalität (Nach jedem Commit)		10.00	9.00	0.00	9.00	-10%	0	800
	5.2	Überprüfung und Anpassung der Dokumentation		2.00	4.00	0.00	4.00	+100%	0	160
	5.3	Code und Dokumentation finalisiert		0.00	0.00	0.00	0.00	+0%	0	0
	6	**Auswertung**		**5.25**	**6.50**	**0.00**	**6.50**	**+24%**	**0**	**420**
	6.1	Dokumentation der Lessons Learned		1.00	1.50	0.00	1.50	+50%	0	80
	6.2	Finalisierung der Projektdokumentation		2.00	3.00	0.00	3.00	+50%	0	160
	6.3	Durchführung der Präsentation vor dem Kurs		0.50	0.50	0.00	0.50	+0%	0	40
	6.4	Bewertung vom Erfolg des Projekts		1.00	1.00	0.00	1.00	+0%	0	80
	6.5	Abgabe der finalen Präsentation und Dokumentation im Git-Repo		0.50	0.25	0.00	0.25	-50%	0	40
	6.6	Übermittlung des GitHub-Links via OneNote		0.25	0.25	0.00	0.25	+0%	0	20
	6.7	Projekt erfolgreich beendet		0.00	0.00	0.00	0.00	+0%	0	0

3.2 Zeitplanung (GANTT)



4 Entscheiden

4.1 Entscheidungsfindung

Im Zuge der Entscheidungsfindung habe ich mich eingehend mit den technologischen Optionen auseinandergesetzt, um die beste Basis für unser Projekt zu schaffen. Unter Berücksichtigung der projektspezifischen Anforderungen und zukünftigen Skalierbarkeit entschied ich mich für MongoDB als datenflexibles NoSQL-Datenbanksystem. .NET 8.0 wurde als Entwicklungsplattform ausgewählt, da es optimale Unterstützung für moderne, RESTful Web-APIs bietet und sich gut in das bestehende Systemgefüge integrieren lässt. Diese Entscheidungen waren entscheidend für die Festlegung der Projektarchitektur und bildeten die Grundlage für die weitere Umsetzung des "NoSQL Ski Service Managers". Dabei wurde stets darauf geachtet, dass die gewählten Technologien und Strukturen nicht nur den momentanen Bedarf decken, sondern auch zukünftige Anpassungen und Erweiterungen ermöglichen.

4.2 Auswahl der Technologien

Kategorie	Technologie/Komponente	Version
Entwicklungsumgebung	.NET Framework	8.0
Mapping-Tool	AutoMapper.Extensions.Microsoft.DependencyInjection	12.0.1
Datenbank	MongoDB.Bson	2.23.1
Datenbank	MongoDB.Driver	2.23.1
Datenbank	Draw.io	v22.1.21
Datenbankserver	MongoDB Server	7.0.5
Logging	Serilog	3.1.1
Logging	Serilog.AspNetCore	7.0.0
API-Dokumentation	Swashbuckle.AspNetCore	6.5.0
Authentifizierung	Microsoft.AspNetCore.Authentication.JwtBearer	8.0.1
API-Entwicklung	Microsoft.AspNetCore.OpenApi	8.0.1
Dokumentation	Microsoft Office Word	V2312 Build 16.0.17126.20132
Tabellenkalkulation	Microsoft Office Excel	V2312 Build 16.0.17126.20132
Projektstrukturplan	WBSTool	v1.0
API-Tests	Postman	v10.22

Im Rahmen der Technologieauswahl für das Projekt "NoSQL Ski Service Manager" habe ich verschiedene Optionen geprüft. Es war mir wichtig, eine Technologie zu wählen, die sowohl eine effiziente Entwicklung als auch eine zukunftssichere Skalierbarkeit ermöglicht. Nach sorgfältiger Abwägung habe ich mich für MongoDB entschieden, da es flexible Datenspeicherung und hervorragende Performance bietet. Für die Backend-Entwicklung fiel die Wahl auf .NET 8.0, da es reichhaltige Funktionalitäten für die Erstellung von Web-APIs zur Verfügung stellt und eine starke Community-Unterstützung bietet. Diese Technologien bieten die gewünschte Balance aus Stabilität, Funktionsumfang und Innovationspotential für eine nachhaltige Projektentwicklung.

4.3 Festlegung der Architektur

In der Phase der Architekturfestlegung des "NoSQL Ski Service Manager"-Projekts habe ich mich auf eine mehrschichtige Backend-Architektur konzentriert, die sich durch Modularität und Skalierbarkeit auszeichnet. Die Entscheidung für MongoDB als NoSQL-Datenbank war zentral, da sie eine flexible und erweiterbare Datenstruktur bietet, die für die dynamischen Anforderungen des Skiservice-Managements geeignet ist.

4.3.1 API-Struktur

Die Backend-Logik wurde in .NET 8.0 entwickelt und bot eine RESTful API für die Frontend-Interaktion. Hierbei wurde besonderer Wert auf eine klare Trennung von Concerns gelegt. Controller, Services und Repositories wurden strukturiert, um eine klare Verantwortungstrennung zu gewährleisten.

4.3.2 Sicherheit und Authentifizierung

In diesem Projekt wurde JWT (JSON Web Tokens) für die Authentifizierung bestimmter API-Endpunkte, insbesondere für die Freischaltung von Benutzerkonten, implementiert. Diese Massnahme dient dem Schutz vor unbefugtem Zugriff und gewährleistet eine sichere Handhabung sensibler Funktionen. Die sorgfältige Beschränkung der Authentifizierung auf spezifische Bereiche bietet Flexibilität, ohne die Sicherheit zu beeinträchtigen, und lässt Raum für zukünftige Erweiterungen der Sicherheitsmechanismen. Dadurch wird sichergestellt, dass nur autorisierte Nutzer kritische Aktionen ausführen können, was die Integrität des Systems stärkt.

4.3.3 Performance und Skalierung

Durch die Verwendung von MongoDB und dessen Indizierungsfunktionen konnte eine hohe Performance bei Abfragen sichergestellt werden. Die Architektur wurde mit Blick auf zukünftige Skalierbarkeit entworfen, sodass bei steigenden Anforderungen eine einfache Erweiterung möglich ist.

4.3.4 Logging und Monitoring

Mit Serilog wurde ein robustes Logging-System implementiert, das eine kontinuierliche Überwachung und Fehlerbehebung der Anwendung ermöglicht. Dies war entscheidend, um die Zuverlässigkeit des Systems zu gewährleisten.

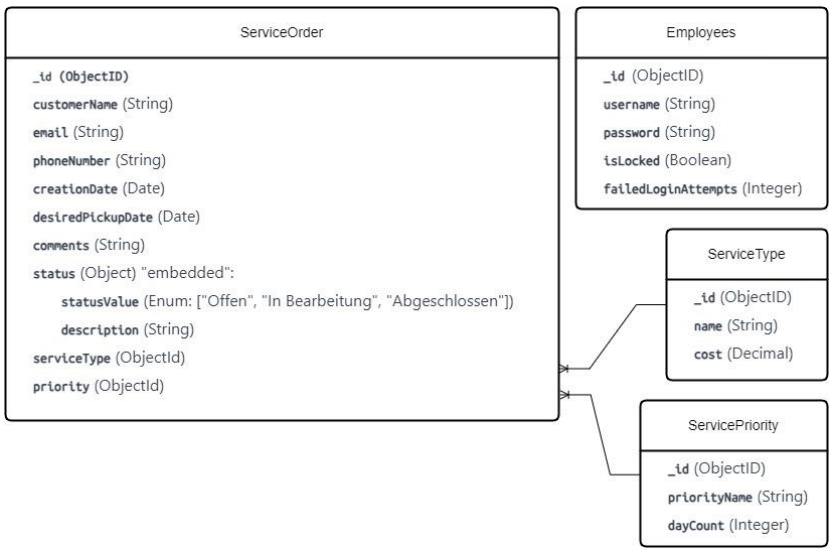
Diese architektonischen Entscheidungen bildeten das Fundament für ein leistungsfähiges, sicheres und wartbares Backend-System, das die Anforderungen des "NoSQL Ski Service Manager"-Projekts vollständig erfüllt.

5 Realisieren

5.1 Implementierung der Anwendung

Die Realisierungsphase des "NoSQL Ski Service Manager"-Projekts umfasste die umfassende Implementierung der Backend-Anwendung. Diese Phase begann mit dem Aufbau der Projektstruktur in .NET 8.0, gefolgt von der Entwicklung der Datenbankmodelle und der Integration von MongoDB. Schlüsselaspekte wie CRUD-Operationen für Serviceaufträge und Benutzerverwaltung wurden implementiert. Besondere Aufmerksamkeit galt der Sicherstellung der Datenintegrität und der Performance-Optimierung. Parallel dazu wurde das Authentifizierungssystem mittels JWT umgesetzt, was eine sichere Benutzerverwaltung ermöglichte. Ein wichtiger Bestandteil war auch die Einrichtung des Logging-Systems mit Serilog, um eine effektive Überwachung und Fehlerbehandlung zu gewährleisten. Abschliessend wurde die API mit Swagger dokumentiert, um eine klare und verständliche Schnittstelle für Frontend-Entwickler bereitzustellen. Diese Phase war geprägt von kontinuierlichem Testing und Anpassungen, um eine robuste und zuverlässige Anwendung zu gewährleisten.

5.2 Datenbankdesign



Für das Projekt "NoSQL Ski Service Manager" wurde ein durchdachtes Datenbankdesign entwickelt, das die Flexibilität und Skalierbarkeit von NoSQL nutzt, um eine effiziente Verwaltung der Serviceaufträge zu gewährleisten. Die Datenbankstruktur basiert auf vier Hauptkollektionen: **ServiceOrder**, **Employees**, **ServiceType** und **ServicePriority**.

- **ServiceOrder**: Diese Kollektion ist das Herzstück des Systems und speichert alle Informationen zu den Serviceaufträgen. Jeder Auftrag umfasst Kundendaten, Daten zu Service und Priorität sowie den Status, der als eingebettetes Objekt mit Werten wie "Offen", "In Bearbeitung" und "Abgeschlossen" realisiert ist.

- **Employees**: Hier werden die Mitarbeiterdaten gespeichert, einschliesslich Benutzername, Passwort und Statusinformationen wie isLocked und failedLoginAttempts. Diese Kollektion ist essenziell für die Authentifizierung und Autorisierung im System.

- **ServiceType**: Diese Kollektion definiert die verschiedenen Arten von Dienstleistungen, die angeboten werden, und enthält Informationen wie Bezeichnung und Kosten.

- **ServicePriority**: Hier werden die Prioritätsstufen für die Aufträge festgelegt, mit Namen und einer Anzahl von Tagen, die die Dringlichkeit des Auftrags bestimmen

5.3 Entwicklung der Backend-Logik

Im Rahmen des "NoSQL Ski Service Manager"-Projekts umfasste die Entwicklung der Backend-Logik die Implementierung der Geschäftslogik auf Serverseite, die für die Handhabung der Anfragen an die API zuständig ist. Der Fokus lag auf der Schaffung einer robusten, effizienten und sicheren Anwendung, die die Verwaltung von Skiservice-Aufträgen ermöglicht.

Die Kernaspekte der Backend-Entwicklung beinhalteten:

5.4 Schnittstellenimplementierung

- Modellierung und Implementierung der Geschäftslogik: Hierzu zählten die Erstellung von Services und Controllern, die Verarbeitung der Geschäftsregeln und die Abwicklung der CRUD-Operationen (Create, Read, Update, Delete) für die einzelnen Kollektionen in der Datenbank.

- Authentifizierung und Sicherheit: Die Implementierung von Authentifizierungsmechanismen mittels JWT (JSON Web Tokens) und die Einrichtung von Zugriffsrechten sorgten für eine sichere Umgebung, in der nur autorisierte Aktionen durchgeführt werden können.

- Fehlerbehandlung und Validierung: Es wurde sichergestellt, dass Eingabedaten validiert und Fehler ordnungsgemäss behandelt werden, um die Stabilität und Zuverlässigkeit des Backends zu gewährleisten.

- Integration von Middleware: Für eine effektive Fehlerprotokollierung und Überwachung des Betriebszustands des Backends wurden entsprechende Middleware-Komponenten eingesetzt.

- API-Design und -Dokumentation: Durch die Nutzung von Swagger wurde die API sorgfältig dokumentiert, um eine klare und verständliche Schnittstelle für die Frontend-Entwicklung und andere potenzielle API-Konsumenten bereitzustellen.

- Testentwicklung: Um die Korrektheit der Backend-Logik zu garantieren, wurden umfassende Unit- und Integrationstests geschrieben und durchgeführt, die eine kontinuierliche Qualitätssicherung während der Entwicklungsphase ermöglichten.

5.5 Fazit Realisierung

Obwohl ich vor Beginn meiner Ausbildung im zweiten Lehrjahr keine Vorkenntnisse im Programmieren hatte, habe ich mich mit Eifer und Engagement den Herausforderungen gestellt. Ich habe viel Zeit und Mühe investiert, um die Grundlagen des Codierens zu erlernen und mich dann weiterzubilden, um die Best Practices und aktuellen Industriestandards zu verstehen. Dieser Prozess hat es mir ermöglicht, ein Projekt zu entwickeln, das nicht nur die technischen Anforderungen erfüllt, sondern auch die Qualität und Wartbarkeit des Codes berücksichtigt, was für professionelle Softwareentwicklung entscheidend ist.

6 Kontrollieren

Im Rahmen der Qualitätssicherung des Projekts "NoSQL Ski Service Manager" wurden massgebliche Schritte unternommen, um die Funktionsfähigkeit und Zuverlässigkeit der entwickelten Anwendung zu gewährleisten. Die Anwendung und Funktionsweise der RESTful API wurden mittels manuell ausgeführter Postman-Tests eingehend geprüft, wobei die Ergebnisse dieser Tests sorgfältig dokumentiert wurden, um etwaige Fehler und Unstimmigkeiten zu identifizieren und zu beheben.

Statt einer CI/CD-Pipeline und automatisierter Unit-Tests lag der Fokus auf manuellen Kontrollen und der direkten Überprüfung durch die Ausführung von Postman-Tests. Diese Tests ermöglichten es, das Zusammenspiel der Backend-Logik mit der Datenbank und den Endpunkten der API umfassend zu testen. Zudem wurde der Quellcode regelmässig einer manuellen Überprüfung unterzogen, um die Codequalität sicherzustellen und die Einhaltung von Best Practices zu überwachen.

Die Dokumentation der API-Endpunkte erfolgte durch den Einsatz von Swagger, welches die Übersichtlichkeit und Verständlichkeit der verschiedenen Funktionen der Anwendung verbesserte und gleichzeitig als praktisches Werkzeug für die Testdurchführung diente. Diese Strategien trugen dazu bei, eine robuste und gut getestete Anwendung zu entwickeln, die den Anforderungen des Projekts gerecht wird.

6.1 Qualitätssicherung

Die Qualitätssicherung meines Projekts "NoSQL Ski Service Manager" konzentrierte sich auf manuelle Tests und Code-Reviews, um eine hohe Codequalität ohne den Einsatz von CI/CD-Pipelines oder Unit-Testing-Frameworks wie xUnit zu gewährleisten. Hierfür erstellte ich detaillierte Testfälle, die ich in Postman durchlief, um sicherzustellen, dass alle API-Endpunkte wie erwartet funktionieren. Die Code-Reviews führte ich selbst durch, unterstützt durch Peer-Feedback, wo immer möglich. Die API-Dokumentation wurde mit Swagger erstellt, was die Übersichtlichkeit und Nachvollziehbarkeit der verschiedenen API-Funktionen verbesserte und als Grundlage für die manuellen Tests diente. Diese Herangehensweise ermöglichte eine sorgfältige Überprüfung aller Funktionen und die Behebung von Fehlern während der Entwicklung.

6.2 Testszenarien und Testfällen

Für die Erstellung der Testszenarien und Testfälle im Rahmen des "NoSQL Ski Service Manager"-Projekts wurde ein methodischer Ansatz verfolgt. Zunächst wurden die Kernfunktionen der Anwendung analysiert und kritische Benutzerabläufe identifiziert. Basierend darauf wurden Testszenarien entwickelt, die sowohl Standard- als auch Ausnahmefälle abdecken.

Für jeden API-Endpunkt wurden mehrere Testfälle entworfen, um die Funktionalität systematisch zu überprüfen. Dies umfasste das Testen von erfolgreichen Abläufen sowie das simulierte Nachstellen von Fehlern, um die Robustheit des Systems zu prüfen.

Die Testfälle wurden in Postman dokumentiert und mit der entsprechenden Datenbankvalidierung und -verarbeitung kombiniert. Bei jedem Commit wurden diese Tests manuell durchgeführt, um sofortige Rückmeldungen zu erhalten und mögliche Regressionen oder neue Bugs schnell identifizieren zu können.

Diese Vorgehensweise gewährleistete, dass alle Aspekte der Anwendung gründlich getestet wurden und dass die Implementierung den festgelegten Anforderungen entspricht. Durch die kontinuierliche Wiederholung und Verfeinerung der Testfälle wurde die Zuverlässigkeit der Anwendung sichergestellt.

6.3 Performance-Optimierung

Die Performance-Optimierung im "NoSQL Ski Service Manager"-Projekt konzentrierte sich auf die Effizienz der Datenbankoperationen. Durch die sorgfältige Erstellung von Indizes für die MongoDB-Collections wurden die Abfragezeiten minimiert. Zusätzlich zur Indizierung wurden Abfragen und Aggregationen optimiert, um eine schnelle Verarbeitung grosser Datenmengen zu gewährleisten. Es wurde Wert daraufgelegt, dass die Operationen im Backend ressourceneffizient gestaltet sind, um die Serverlast zu minimieren und eine schnelle Antwortzeit der API zu garantieren. Diese Massnahmen trugen dazu bei, das System ohne die Implementierung komplexerer Optimierungsstrategien wie Caching oder Lasttests effizient zu gestalten.

6.4 Sicherheitsüberprüfungen

Im Rahmen des "NoSQL Ski Service Manager"-Projekts wurden umfangreiche Sicherheitsüberprüfungen durchgeführt, um die Integrität und Vertraulichkeit der Daten sowie die Sicherheit der Backend-Architektur zu gewährleisten. Die Implementierung von Sicherheitsprotokollen, wie beispielsweise die Verwendung von JSON Web Tokens (JWT) für die Authentifizierung, stellte sicher, dass nur autorisierte Benutzer Zugang zu bestimmten Endpunkten haben. Die Sicherheitsüberprüfungen umfassten auch die Validierung der Eingabedaten, um SQL-Injection und andere gängige Webangriffe zu verhindern. Sorgfältige Code-Reviews und automatisierte Sicherheitstests wurden regelmässig durchgeführt, um Schwachstellen frühzeitig zu erkennen und zu beheben. Diese Massnahmen bildeten eine mehrschichtige Verteidigung gegen potenzielle Sicherheitsbedrohungen und trugen dazu bei, das Backend-System robust gegenüber externen Angriffen zu gestalten.

7 Auswerten

7.1 Projektbewertung

Die Projektbewertung des "NoSQL Ski Service Manager" Projekts fiel überwiegend positiv aus. Trotz einiger Herausforderungen und Lernkurven, insbesondere im Bereich der NoSQL-Datenbanktechnologie und Backend-Entwicklung, konnte das Projekt erfolgreich abgeschlossen werden. Die Umsetzung der Kernfunktionalitäten, wie Datenmigration, CRUD-Operationen und API-Entwicklung, erfüllte die gestellten Anforderungen. Besonders hervorzuheben ist der erfolgreiche Einsatz von JSON Web Tokens zur Authentifizierung und die Einhaltung von Sicherheitsstandards. Das Projekt demonstrierte effektiv die Anwendung von NoSQL-Technologien in der Backend-Entwicklung und trug zur Weiterentwicklung meiner Fähigkeiten bei. Insgesamt bot das Projekt eine wertvolle Erfahrung in der Anwendung moderner Softwareentwicklungspraktiken und -technologien.

7.2 Lessons Learned

Während der Realisierung des "NoSQL Ski Service Manager" Projekts habe ich eine Vielzahl wertvoller Erfahrungen und Lektionen gelernt. Zunächst wurde mir die Wichtigkeit einer sorgfältigen Planung und Recherche in der Frühphase eines Projekts bewusst, was zur Auswahl geeigneter Technologien und Werkzeuge führte. Die Auseinandersetzung mit NoSQL-Datenbanken und MongoDB erweiterte mein Verständnis für nicht-relationale Datenstrukturen und deren Vorteile in bestimmten Anwendungsfällen.

Die Arbeit mit .NET 8.0 und der Entwicklung einer RESTful API bot eine solide Grundlage für das Verständnis von Backend-Entwicklung. Die Implementierung von Authentifizierung mit JWT zeigte die Bedeutung von Sicherheit in Web-Anwendungen. Das Erstellen von Skripten für Backup und Restore sowie das Handling von Datenbankindizes verbesserte meine Fähigkeiten im Bereich Datenmanagement. Die Herausforderung, ein Projekt mit begrenztem Vorwissen und Erfahrung zu leiten, lehrte mich, proaktiv nach Lösungen zu suchen und mich intensiv in neue Technologien einzuarbeiten. Die Bedeutung von Dokumentation und qualitätssichernden Massnahmen wie Postman-Tests wurde deutlich.

Zusammenfassend war das Projekt eine umfassende Lernerfahrung, die meine Fähigkeiten in Softwareentwicklung und Projektmanagement erheblich erweiterte und mir das Vertrauen gab, zukünftige Projekte mit komplexen Anforderungen zu meistern.

7.3 Ausblick und Weiterentwicklung

Für den Ausblick und die Weiterentwicklung des "NoSQL Ski Service Manager" Projekts plane ich, das Backend auf einem MongoDB Cloud-Host zu implementieren. Dieser Schritt bietet die Möglichkeit, das System in einer realen Umgebung zu testen und zugleich die Skalierbarkeit und Verfügbarkeit zu erhöhen. Eine wesentliche Anpassung wird dabei die Sicherheit betreffen, insbesondere die sichere Handhabung des Zugriffs auf die Cloud-Datenbank. Hierzu gehören die Implementierung stärkerer Authentifizierungsmechanismen und die Überprüfung der Netzwerksicherheit, um sicherzustellen, dass der Datenzugriff sowohl sicher als auch effizient ist.

Darüber hinaus ist geplant, die Funktionalität des Systems weiter zu verbessern und zu erweitern. Dies könnte die Integration zusätzlicher Features oder die Optimierung bestehender Prozesse beinhalten, um die Benutzererfahrung zu verbessern und die Effizienz des Systems zu steigern. Durch kontinuierliche Wartung und Anpassung an die sich ändernden Anforderungen und Technologien werde ich sicherstellen, dass der "NoSQL Ski Service Manager" auch zukünftig leistungsstark und relevant bleibt.

8 Fazit

Das Fazit des Projekts "NoSQL Ski Service Manager" reflektiert einen bedeutenden Lern- und Entwicklungsprozess. Trotz der Herausforderungen, die mit der Migration von einer relationalen zu einer NoSQL-Datenbank einhergingen, konnte ein funktionsfähiges und leistungsfähiges Backend-System erfolgreich implementiert werden. Die Erfahrung, die durch die Arbeit an diesem Projekt gesammelt wurde, ist immens, insbesondere im Hinblick auf die Verwendung moderner Technologien und Entwicklungsmethoden. Die erfolgreiche Umsetzung des Projekts demonstriert nicht nur meine Fähigkeiten als Entwickler, sondern stellt auch eine solide Basis für zukünftige Projekte dar. Dieses Projekt hat wesentlich dazu beigetragen, mein Verständnis für Datenbanktechnologien zu vertiefen und meine Kompetenzen in der Softwareentwicklung zu stärken. Mir ist bewusst, dass der Einsatz von .exe-Tools zur Automatisierung in einer produktiven Umgebung nicht der Standardpraxis entspricht. Diese Entscheidung wurde jedoch bewusst getroffen, um die Installation und Inbetriebnahme für den Dozenten und andere Nutzer so einfach wie möglich zu gestalten.

9 Anhänge

9.1 Quellcodeausschnitte

9.1.1 GenericController

```
[HttpPost]
public virtual async Task<IActionResult> Create(TCreateDto createDto)
{
    var createdItem = await _service.CreateAsync(createDto);
    return CreatedAtAction(nameof(Get), new { id = createdItem.Id }, createdItem);
}
```

Der GenericController nutzt die Vorteile der Generizität in C#, um wiederverwendbare CRUD-Operationen für verschiedene Entitätsklassen bereitzustellen. Das bedeutet, dass für jede spezifische Entität wie ServiceOrder oder Employee keine separaten Controller-Methoden implementiert werden müssen. Die Implementierung der Create-Methode im generischen Controller zeigt, wie ein neues Objekt basierend auf dem übergebenen DTO (Data Transfer Object) erstellt und gespeichert wird, wobei die spezifische Logik in den entsprechenden Service-Klassen definiert wird.

9.1.2 ServiceOrderService (Erweiterung von GenericService):

```
public override async Task<OrderResponseDto> CreateAsync(CreateServiceOrderRequestDto
createDto)
{
    // Überschriebene Logik für ServiceOrder CreateAsync
}
```

Der ServiceOrderService erweitert den GenericService um spezielle Geschäftslogik für das Erstellen von Serviceaufträgen. Dieser Codeausschnitt zeigt, wie die Methode CreateAsync der Basisklasse überschrieben wird, um spezifische Anforderungen wie das Überprüfen und Verarbeiten von Service-Typen und Prioritäten zu behandeln. Dieser Ansatz ermöglicht eine klare Trennung zwischen generischen CRUD-Operationen und geschäftsspezifischer Logik.

9.1.3 ServiceOrderController

```
public class ServiceOrderController : GenericController<ServiceOrder,
CreateServiceOrderRequestDto, UpdateServiceOrderRequestDto, OrderResponseDto>
{
    private readonly ServiceOrderService _serviceOrderService;

    public ServiceOrderController(
        GenericService<ServiceOrder, CreateServiceOrderRequestDto,
UpdateServiceOrderRequestDto, OrderResponseDto> genericService,
        ServiceOrderService serviceOrderService) : base(genericService)
    {
        _serviceOrderService = serviceOrderService;
    }

    [HttpPost]
    public override async Task<IActionResult> Create(CreateServiceOrderRequestDto
createDto)
    {
        // Überschriebene Logik für ServiceOrder CreateAsync
    }

}
```

Der ServiceOrderController ist ein spezialisierter Controller, der von GenericController erbt. In diesem Codeausschnitt wird die Create-Methode explizit für die ServiceOrder-Entität überschrieben, um eine spezifische Geschäftslogik zu implementieren. Obwohl nur diese Methode sichtbar ist, erbt der Controller automatisch die anderen CRUD-Operationen (Read, Update, Delete) von GenericController.

Diese Erbschaft führt dazu, dass in Swagger-UI alle CRUD-Operationen für ServiceOrder angezeigt werden, obwohl im ServiceOrderController-Code nur die Create-Operation definiert ist. Dies ermöglicht es, den Code für gemeinsame Operationen zentral zu verwalten und gleichzeitig für spezifische Entitäten anzupassen, was die Wiederverwendbarkeit des Codes und die Wartbarkeit der Anwendung erhöht.

9.2 API-Dokumentation

9.2.1 GenericController Endpunkte:

- **GET /api/{controller}:** Listet alle Einträge der entsprechenden Entität.
- **GET /api/{controller}/{id}:** Ruft eine spezifische Entität ab.
- **POST /api/{controller}:** Erstellt eine neue Entität.
- **PUT /api/{controller}/{id}:** Aktualisiert eine bestehende Entität.
- **DELETE /api/{controller}/{id}:** Löscht eine Entität.

9.2.2 ServiceOrderController Endpunkte:

- **POST /api/ServiceOrder:** Erstellt eine neue ServiceOrder.
- Zusätzlich erben alle Endpunkte von GenericController.

9.2.3 EmployeeController Endpunkte:

- **POST /api/Employee/login:** Authentifiziert einen Mitarbeiter.
- **POST /api/Employee/unlock/{username}:** Entsperren eines Mitarbeiterkontos.
- Zusätzlich erben alle Endpunkte von GenericController.

Die Endpunkte bieten eine vollständige CRUD-Funktionalität für ServiceOrders und Employees. Die Nutzung von generischen Controllern ermöglicht eine effiziente und konsistente Behandlung aller Entitäten. Swagger-UI bietet eine interaktive Dokumentation, die es Entwicklern erleichtert, die API zu verstehen und zu verwenden.

9.3 Postman Collection

Die Postman Collection "JetStreamAPI Tests" enthält eine Reihe von Tests für die ServiceOrder API Ihres Projekts. Diese Tests decken die grundlegenden CRUD-Operationen (Create, Read, Update, Delete) ab. Jeder Test besteht aus einem Request, der an die API gesendet wird, und einem dazugehörigen Testskript, das die erwarteten Ergebnisse überprüft.

Zum Beispiel gibt es Tests für das Erstellen von ServiceOrders, das Abrufen aller ServiceOrders oder eines einzelnen ServiceOrders anhand seiner ID, das Aktualisieren und das Löschen von ServiceOrders. Zusätzlich enthält die Collection Tests für den Employee-Teil der API, einschliesslich des Anlegens von Mitarbeitern, des Logins, des Entsperrens von gesperrten Benutzerkonten sowie des Abrufens und Löschens von Mitarbeiterdaten.

Jeder Test in der Collection prüft spezifische Aspekte der API-Anforderungen und -Antworten, wie den Statuscode der Antwort, die Struktur der Antwortdaten und das Vorhandensein spezifischer Schlüssel in den JSON-Antworten.

Diese Postman Collection dient somit als wertvolles Werkzeug für die Überprüfung und Sicherstellung der Funktionsweise und Zuverlässigkeit Ihrer API.

9.4 Benutzerkonzept

Die Benutzerauthentifizierung und Zugriffsberechtigungen sind wesentliche Aspekte der Anwendungssicherheit und -verwaltung. Diese Dokumentation beschreibt die Konfiguration von Benutzern und deren Zugriffsrechten in den verschiedenen Umgebungen der Anwendung.

9.4.1 Entwicklungsumgebung

In der Entwicklungsumgebung werden Entwicklungsarbeiten und Tests durchgeführt. Hierbei wird der Benutzer JetStreamApiMaster verwendet, um auf die MongoDB-Datenbank JetStreamAPI zuzugreifen.

Benutzerkonfiguration:

- **Benutzername:** JetStreamApiMaster
- **Passwort:** SavePassword1234
- **Rolle:** dbOwner
- **Datenbank:** JetStreamAPI

Der Benutzer JetStreamApiMaster verfügt über die dbOwner-Rolle, was ihm umfassende Verwaltungsrechte für die Datenbank JetStreamAPI gewährt. Dies ermöglicht das Erstellen, Aktualisieren und Löschen von Daten sowie die Verwaltung der Datenbank selbst.

9.4.2 Produktionsumgebung

In der Produktionsumgebung wird die Anwendung live betrieben und öffentlich zugänglich gemacht. Hierbei wird der Benutzer BackendUser verwendet, um auf die MongoDB-Datenbank JetStreamAPI zuzugreifen.

Benutzerkonfiguration:

- **Benutzername:** BackendUser
- **Passwort:** BackendUserPassword123
- **Rolle:** readWrite
- **Datenbank:** JetStreamAPI

Der Benutzer BackendUser verfügt über die readWrite-Rolle, was ihm Lese- und Schreibzugriff auf die Datenbank JetStreamAPI gewährt. Er kann Daten abfragen und aktualisieren, jedoch nicht die Verwaltungsaufgaben wie das Löschen der Datenbank ausführen.

Wichtig:
In beiden Umgebungen, der Entwicklungsumgebung und der Produktionsumgebung, müssen die Sicherheitseinstellungen des MongoDB-Servers so konfiguriert sein, dass die Benutzerauthentifizierung aktiviert ist, damit die in dieser Dokumentation beschriebenen Benutzer erfolgreich auf die Datenbank zugreifen können. Andernfalls haben auch diese Benutzer alle Berechtigungen.

10 Literaturverzeichnis

- **MongoDB, Inc.** "MongoDB Documentation." https://docs.mongodb.com/
- **Jimmy Bogard.** "AutoMapper Documentation." https://docs.automapper.org/en/stable/
- **Microsoft.** ".NET Documentation." https://docs.microsoft.com/en-us/dotnet/
- **OpenAPI Initiative.** "Swagger OpenAPI." https://swagger.io/specification/
- **Postman.** "Postman Learning Center." https://learning.postman.com/
- **Microsoft.** "ASP.NET Core Documentation." https://docs.microsoft.com/en-us/aspnet/core/
- **Serilog.** "Serilog GitHub Repository." https://github.com/serilog/serilog
- **Microsoft.** "ASP.NET Core Authentication." https://docs.microsoft.com/en-us/aspnet/core/security/authentication/

11 Glossar

- **NoSQL:** Nicht-relationale Datenbanksysteme, die für grosse Mengen verteilter Daten konzipiert sind.
- **MongoDB:** Ein dokumentenorientiertes NoSQL-Datenbanksystem.
- **CRUD:** Akronym für Create, Read, Update, Delete – die vier grundlegenden Funktionen, die in Datenbankanwendungen implementiert sind.
- **API:** Application Programming Interface, eine Schnittstelle, die es Programmen ermöglicht, mit anderen Programmen zu kommunizieren.
- **JWT:** JSON Web Tokens, eine Methode für die sichere Übertragung von Informationen zwischen Parteien als JSON-Objekte.
- **Swagger:** Ein Tool zur Visualisierung und Dokumentation von RESTful APIs.
- **Postman:** Eine Plattform für API-Entwicklung, die das Erstellen, Teilen, Testen und Dokumentieren von APIs vereinfacht.
- **.NET:** Eine von Microsoft entwickelte Plattform für die Entwicklung von Anwendungen.
- **Automapper:** Ein Objekt-Objekt-Mapper, der automatisch Objekte eines Typs auf einen anderen abbildet.
- **CI/CD:** Continuous Integration und Continuous Deployment, Praktiken in der Softwareentwicklung, die das Zusammenführen von Codeänderungen und das automatische Ausliefern von Anwendungen erleichtern.

12 Zusätzliche Anforderungen auf denen der Fokus liegt

- AO1 Automatisiertes Backup-Konzept durchgeführt u. implementiert.
- AO2 Komplexe Schema Validierungen umgesetzt (Referenzen, enum, min, max. usw)
- (AO5 Komplexe statistische Auswertungsabfragen realisiert)

AO2 & AO5 befinden sich beide im MongoDbScripts Ordner.
AO1 befindet sich in meiner `MongoBackupManager` Klasse

13 Versionsverlauf

Version	Datum	Beschreibung
V0.01	24.01.2024 20:13	Dokument mit Deckblatt erstellt
V0.02	26.01.2024 10:04	Gliederungspunkte erstellt
V0.03	30.01.2024 21:54	«Einführen und Informieren» erstellt
V0.04	31.01.2024 12:10	«Planen & Entscheiden» dokumentiert
V0.05	31.01.2024 22:56	«Realisieren & Auswerten» beendet
V0.06	01.02.2024 09:53	Relevante Quellcode Dokumentationen erstellt
V0.07	01.02.2024 20:14	API Dokumentation erstellt
V0.08	01.02.2024 23:14	API Dokumentation angepasst und «Anhänge» fertiggestellt. Literaturverzeichnis und Glossar ebenfalls fertiggestellt
V0.09	02.02.2024 06:48	Versionsverlauf fertiggestellt
V0.10	02.02.2024 19:29	Zusätzliche Anforderungen erwähnt