



1.2.2023

# Dokumentation: Ski-Service Auftragverwaltung/NoSql

Modul 165 –NoSQL-Datenbanken einsetzen



David Mangold  
IBZ

## Inhalt

Einleitung.....	4
Verwendet Tools & Nuget: .....	5
Dokumentation nach dem IPERKA-Modell.....	6
Informieren.....	6
Projekt Informieren .....	6
Fragen aufschreiben .....	6
Planen.....	7
PSP und GANT Plan erstellen.....	7
Databank Aufbau.....	8
API Aufbau .....	8
Migration der DB .....	8
Entscheiden .....	9
Entscheid der Datenbank .....	9
Entscheid der Datenbankschema.....	9
Client Collection: .....	9
Mitarbeiter Collection:.....	9
API Aufbau .....	10
Migration der Daten.....	10
Realisieren .....	10
API Realisieren.....	11
Projekterstellung: .....	11
Verbindung zur MongoDB herstellen.....	11
Modelldefinition.....	11
Service-Schicht: .....	11
Controller-Schicht:.....	11
Authentifikation und Autorisierung: .....	11
Testen und Debuggen .....	11
Migration der Daten/ Backup und Restore .....	12
Migration .....	12
Backup: .....	12
Backup wiederherstellen.....	12
Postman Request erstellen .....	12
Neue Collection erstellen .....	12
Neuen Request erstellen .....	12
Request-Methoden auswählen .....	12

URL eingeben: Geben .....	12
Request-Header einstellen (optional): .....	12
Request-Body (optional): .....	12
Ergebnis prüfen: .....	12
Kontrollieren.....	13
API mit Postman Testen .....	13
Alle HTTP Methode Testen.....	13
Anforderungen erfüllt .....	14
Auswerten .....	14
Schlussfolgerung.....	14
Zukunft dieser API .....	14
Fazit .....	14
Quellenverwies.....	15

## Abbildungsverzeichnis

Abbildung 1 GANT und PSP .....	7
Abbildung 2: Neue Datenbank Verbindung.....	11
Abbildung 3: Models Ordner .....	11
Abbildung 4: Services Ordner .....	11
Abbildung 5: Controllers Ordner .....	11
Abbildung 6: Postman Fenster .....	13

## Abkürzungen

<u>Abkürzung</u>	<u>Beschreibung</u>
CRUD	Creat, Read, Update, Delete
MC	Model Control
API	Applikation Programming Interface
JWT	JSON Web Token
SQL	„Structured Query Language

## Einleitung

Der Zweck der Projektarbeit "Ski-Service Auftragsverwaltung" ist die Erstellung einer neuen API, die sich an die vorhergehenden Projekte "Ski-Service Anmeldung", "Ski-Service Management" und "Ski-Service.NET App" aus den Modulen 294, 295 und 322 anlehnt. Diese API soll als Auftragsmanagement für die Mitarbeiter fungieren und eine Verbindung zwischen der Kunden-Webseite und der Datenbank herstellen. Mit Hilfe von MongoDB wird die Datenbank verwaltet.

### Dieses Projekt sollte Folgende Punkte beinhalten:

Das Teilprojekt umfasst ausschliesslich den Backendteil und umfasst folgende Aufträge, welche nach IPERKA durchzuführen sind:

- Datenbankdesign und Implementierung (NoSQL)
- Datenmigration (SQL → NoSQL)
- Migration WebAPI Projekt (siehe Modul 295)
- Testprojekt / Testplan
- Realisierung der kompletten Anwendung, gemäss den Anforderungen
- Durchführung Integrationstest mit bestehenden Frontend Lösung.

### Allgemeine Anforderungen

Das Auftragsmanagement muss folgende Funktionen zur Verfügung stellen:

- Login mit Benutzername und Passwort
- Anstehende Serviceaufträge anzeigen (Liste)
- Bestehende Serviceaufträge mutieren. Dazu stehen folgende Status zu Verfügung: Offen, InArbeit und abgeschlossen
- Aufträge löschen (ggf. bei Stornierung)

Die Informationen zur Online-Anmeldung, welche bereits realisiert wurde, müssen ggf. bei Bedarf wie folgt ergänzt werden.

- Kundenname
- E-Mail
- Telefon
- Priorität
- Dienstleistung

Die Firma bietet folgende Dienstleistungen (Angebot) an:



- Kleiner Service
- Grosse Service
- Renn ski-Service
- Bindung montieren und einstellen
- Fell zuschneiden
- Heisswachsen

## Verwendet Tools & Nuget:

Als Texteditor habe ich Visual Studio Code genutzt und das ASP.NET Webframework implementiert. Die Verbindung mit der Datenbank wurde mithilfe von MongoDB realisiert. POSTMAN wurde verwendet, um die APIs zu entwerfen, erstellen, testen und iterieren. Um das Projekt zu verwalten, wurde ein GitHub-Repository erstellt und täglich alle Änderungen am Projekt hochgeladen.

Um mit eine Verbindung mit der Db zu erstellen brauchte ich Folgende Nugets:

The image shows two screenshots of the NuGet package manager interface. The top screenshot is for the 'MongoDB.Driver' package. It shows a table with two rows: 'Project' and 'SkiServiceNoSQL', both with checked checkboxes. Below the table, the 'Installed' status is 'not installed' and the 'Version' is 'Latest stable 2.18.0'. The bottom screenshot is for the 'Swashbuckle.AspNetCore' package. It shows a table with two rows: 'Project' and 'SkiServiceNoSQL'. The 'SkiServiceNoSQL' row has a checked checkbox, a version of '6.5.0', and an 'Installed' status of '6.5.0'. Below the table, the 'Installed' status is '6.5.0' and the 'Version' is 'Latest stable 6.5.0'.



**MongoDB.Driver**  

Versions - 0

<input checked="" type="checkbox"/>	Project	Version	Installed
<input checked="" type="checkbox"/>	SkiServiceNoSQL		

**Installed:** not installed **Uninstall**

**Version:** Latest stable 2.18.0 **Install**

**Swashbuckle.AspNetCore**  

Versions - 1

<input type="checkbox"/>	Project	Version	Installed
<input type="checkbox"/>	SkiServiceNoSQL	6.5.0	6.5.0

**Installed:** 6.5.0 **Uninstall**

**Version:** Latest stable 6.5.0 **Install**

## Dokumentation nach dem IPERKA-Modell

Die Dokumentation erfolgt unterteilt nach den sechs Phasen des IPERKA-Modells.

### Informieren

In der Informationsphase habe ich mich über die Anforderungen und Bedürfnisse für das Projekt informiert und bin auf die Datenbank MongoDB gestoßen. Um mehr über sie zu erfahren, habe ich mich auch im Internet weitergehend informiert.

### Projekt Informieren

Am ersten Tag des Moduls wurde uns das Projekt vorgestellt. Da wir einen weiterführenden Auftrag ausführen mussten, begann ich sofort mit der Recherche, wie man Daten auf eine andere Art und Weise als in Tabellen speichern kann. Ich beschäftigte mich mit MongoDB und nutzte sowohl die Informationen, die wir in der Schule erhielten, als auch das Internet (die Links finden sich im [Quellenverzeichnis](#)). Ein grosser Teil meiner Recherche war es, das Konzept von NoSQL-Datenbanken zu verstehen, da es ein wesentlicher Bestandteil dieses Projekts ist.

### Fragen aufschreiben

Ich habe bei meiner Recherche immer wieder Fragen notiert, wenn ich nicht weitergekommen bin. Diese Fragen habe ich entweder aufbewahrt und weiter erforscht (einige haben sich von selbst geklärt), oder ich musste sie meinem Lehrer oder einer Klassenkamerad stellen.

Die Unklarheiten waren unter anderem:

- Wie kann man von eine SQL DB zu MongoDB migrieren?
- Was sind die Notwendige Tools die wir für den Export und Import zur Verfügung haben?
- Welche Nugets sind für dieses Projekt notwendig?
- Muss ein Dokument embedded sein ?
- Wenn macht es sin ein Dokument zu embedden?
- Wie kann ich entscheiden welche Attribut ich Indexieren soll?

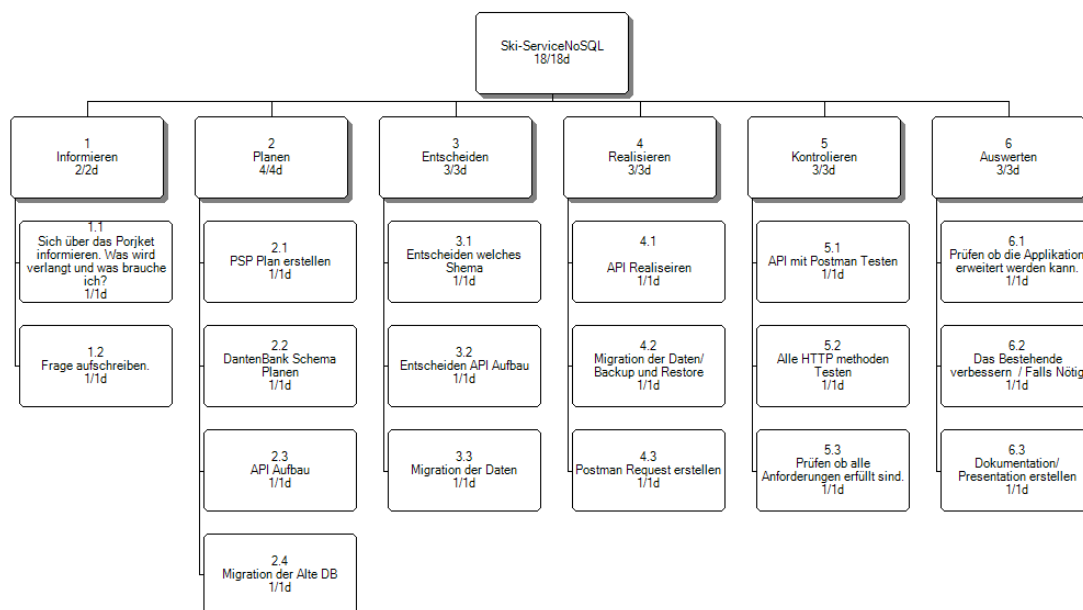
Nachdem die Unklarheiten ausgeräumt waren, habe ich mich weiter im Internet informiert, um die im Arbeitsauftrag beschriebenen Schritte ausführen zu können. In diesem Modul haben wir auch verschiedene Tools und Methoden kennengelernt, um Daten von einer Datenbank zu einer anderen zu übertragen oder ein Backup durchzuführen.

## Planen

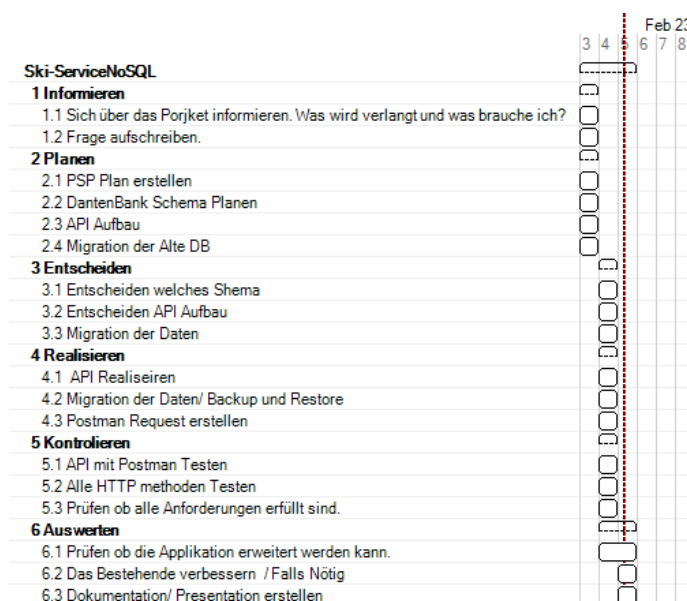
Als Erstes in der Planungsphase habe ich zeitliche Schätzungen für jeden Teil des Projekts vorgenommen und einen Zeitplan erstellt. Dadurch war es möglich, einen konkreten Plan für die API aufzustellen, indem ich das passende Schema ausgewählt habe.

### PSP und GANT Plan erstellen

Um das Projekt erfolgreich innerhalb des gegebenen Zeitrahmens abzuschliessen, habe ich mithilfe des WBStools einen Ablaufplan erstellt. Dieser GANT- und PSP-Plan sollte mir helfen, meine Arbeitspakete im Blick zu behalten und stets zu wissen, welche Aufgaben als Nächstes anstehen. Obwohl es mir nicht immer gelungen ist, exakt nach Plan zu arbeiten, habe ich den Zeitrahmen im Grossen und Ganzen eingehalten.



Projekt1.wbst 1.2.2023 13:29:12



Projekt1.wbst 1.2.2023 13:29:12

Abbildung 1 GANT und PSP



## Databank Aufbau

Im Rahmen meiner Planung habe ich die Struktur und das Schema meiner Datenbank festgelegt. Hierbei habe ich zwei Optionen in Betracht gezogen: Entweder ein Schema mit eingebetteten JSON-Dokumenten oder ein einfacheres Schema ohne Embedding oder Referenzen.

Ausserdem habe ich das Zugriffs- und Validierungskonzept bestimmt, darunter die Definition von Pflichtfeldern, Enums und Minimal-/Maximalwerten.

Des Weiteren war es notwendig, Möglichkeiten für Backup und Restore zu berücksichtigen.

## API Aufbau

In diesem Schritt habe ich die Konstruktion des Web-APIs geplant. Dies beinhaltet die Definition der Klassen, die Auswahl der Methoden, die implementiert werden sollen, sowie die Überlegungen zu zusätzlichen Anforderungen. Der Vorteil war das einige dieser Anforderungen waren bereits vorgegeben, da es bereits eine API mit relationaler Datenbank gibt und entsprechende Entscheidungen in Bezug auf Authentifizierung und Aufbau getroffen wurden.

## Migration der DB

Ich war lange unsicher, wie ich die Migration der Datenbank vornehmen sollte. Ein manuelles Einfügen der Daten im VisualCode erschien mir nicht wie eine wirkliche Migration. Also begann ich im Internet zu recherchieren und fand auf einer Webseite heraus, dass es einen Befehl in SQL gibt, um die Datenbank in eine .JSON-Datei zu konvertieren. Diese Möglichkeit schien mir für die zusätzlichen Anforderungen die beste Lösung zu sein. Es muss Jedoch noch ein zusätzlichen schritt gemacht werden. Und zwar, nach der Ausführung des Befehl in SQL, bekommt man einen Link der die Daten in einem Json Format formatiert. Diese Json file kann dann anschliessend mit mongoimport in der Datenbank hinzugefügt werden.

## Entscheiden

Am letzten Tag des Moduls traf ich eine endgültige Entscheidung bezüglich des Aufbaus meiner Datenbank. Vor allem welche Art von Datenbank (MongoDB oder Node4J). Zusätzlich überlegte mir, ob ich meine Dokumente einbetten und wie viele ich davon haben möchte, sowie wie mein Schema aussehen sollte. Darüber hinaus entschied ich, wie ich die API aufbauen und die Daten migrieren werde.

### Entscheid der Datenbank

Ich habe mich für MongoDB als meine primäre Datenbankentscheidung entschieden, weil es besser zu meinen Anforderungen passt als Node4j.

Zunächst war es die Flexibilität von MongoDB, die mich überzeugt hat. Es ermöglicht das Speichern unterschiedlicher Datenstrukturen in einer einzigen Sammlung, was für mich von großer Bedeutung ist, da meine Daten in der Regel sehr heterogen sind.

Ein weiterer wichtiger Faktor war die Skalierbarkeit von MongoDB. Es kann einfach horizontale Skalierbarkeit erreichen, indem man weitere Knoten hinzufügt, um die Last aufzuteilen. Dies gibt mir die Gewissheit, dass meine Anwendung auch bei wachsendem Datenvolumen stabil bleiben wird.

Insgesamt hat MongoDB in Bezug auf Flexibilität und Skalierbarkeit die besseren Funktionen für mich, und deshalb habe ich mich für MongoDB als meine primäre Datenbankentscheidung entschieden.

### Entscheid der Datenbankschema

Ich musste mich für den Aufbau und das Schema meiner Datenbank entscheiden. Ich entschied mich für die Erstellung von zwei Collections, einer für Client und einer für Mitarbeiter, ohne Embedded Dokumente oder Referenzen auf andere Dokumente. Diese Entscheidung traf ich bewusst, da die API keine Embedded- oder Referenzdokumente benötigt und somit unnötige Komplexität hinzufügt. Zusätzlich setzte ich auf eine Schemavalidierung die wie folgt aussieht:

#### Client Collection:

- Name, Priorität, Status, Dienstleistung(«Fasility») sind Pflichtfelder und sind Indexiert
- Eigenschaften müssen sinnvollen Datentyp haben
- Status, Priorität, Dienstleistung dürfen nur bestimmte Werte haben:
  - Status: "Offen", "InArbeit", "Abgeschlossen"
- Priorität
  - "Express", "Standard", "Tief"
- Dienstleistung:
  - "Kleiner-Service", "Grosser-Service", "Rennski-Service", "Bindung-montieren-und-einstellen", "Fell-zuschneiden", "Heisswachsen"

#### Mitarbeiter Collection:

- Name, Api-Key:
- Eigenschaften müssen sinnvollen Datentyp haben
- Api-Key darf nur ein bestimmte wert haben. Da es der Schlüssen ist um API Request durchzuführen

Ich habe Validierungen implementiert, um sicherzustellen, dass Daten im richtigen Format eingegeben und Pflichtfelder ausgefüllt werden. Ausserdem verhindern sie das Eintragen nicht existierender Status.

### API Aufbau

Dies ist eine Zusammenfassung der wichtigsten Entscheidungen und Überlegungen bei der Umsetzung des Projekts:

- Strukturierung des Projekts mit Modellen, Controllern und Services für bessere Wiederverwendbarkeit und Lesbarkeit
- Integrierung von Exception Handling und Logger zur effektiven Fehlerbehandlung und Diagnostik
- Verwendung von Dependency Injection bei Services für erhöhte Wiederverwendbarkeit
- Einsatz von ApiKey für Authentifikation als sicherste Option für Benutzeranmeldung
- Definition von 3 Controllern (inklusive zugehöriger Services) für Status, Registrationen und Benutzer
- Verwendung der camelCase Konvention für Benennung von Variablen, Klassen, Methoden und Eigenschaften sowie hinzufügen von XML Kommentaren zur Verbesserung der Lesbarkeit und Verständlichkeit des Codes.

### Migration der Daten

Für die Migration der Daten musste ich mir Gedanken darüber machen wie ich am sinnvollsten die Daten mit einem Skript rüber bringen kann. Diese Skripte sind für mich von grosser Bedeutung, da sie den Übergang von einer Relationalen Datenbank zu einer NoSQL Datenbank vereinfachen. Sie ermöglichen es mir, die Daten automatisch zu übertragen und sicherzustellen, dass keine Daten verloren gehen oder beschädigt werden. Ausserdem bieten sie eine einfache Möglichkeit, die Daten in Zukunft zu aktualisieren und zu pflegen, ohne dass ich den manuellen Übertragungsprozess wiederholen muss. Darüber hinaus kann ich die Skripte auch an andere Projekte und Teams weitergeben, was die Übertragung ihrer Datenbanken erleichtern kann. Kurz gesagt, die Skripte für die Datenbankmigration sind ein wertvolles Werkzeug, das ich in meiner Arbeit nutzen würde. Ich musste aber die Daten auch Manuell implementieren da es eine Anforderung von diesem Projekt ist.

### Realisieren

Nach Abschluss der Planung und Beschaffung aller notwendigen Ressourcen für mein Projekt, habe ich die Realisierungsphase begonnen und mit dem Programmieren gestartet. Hierbei habe ich zuerst eine grundlegende Struktur erstellt und anschliessend die Feinheiten programmiert (einschliesslich der Integration von Loic) sowie die Verbindung zur API hergestellt.

Dieses Projekt wurde mit .NET 6.0 entwickelt und benötigte folgende NuGet-Pakete (inklusive Versionen):

- MongoDB.Driver (Version 2.18.0)
- Serilog.AspNetCore (Version 6.1.0)
- Swashbuckle.AspNetCore (Version 6.5.0)

Die Entwicklung erfolgte mit Visual Studio 17.4.4 auf einem Rechner mit Microsoft Windows 11.

## API Realisieren

Für dieses Projekt bin ich wie folgt vorgegangen:

**Projekterstellung:** Ich habe ein neues ASP.NET Web API Projekt in Visual Studio erstellt.

**Verbindung zur MongoDB herstellen:** Um eine Verbindung zu deiner MongoDB herzustellen habe ich das MongoDB Driver NuGet Package installiert.

Um eine neue Verbindung herzustellen wird die Klasse MongoClient aufgerufen und sieht wie folgt aus :

```
public ClientService(IOptions<SkiServiceDatabaseSettings> SkiServiceDatabaseSettings)
{
    var mongoClient = new MongoClient(SkiServiceDatabaseSettings.Value.ConnectionString);
    var mongoDatabase = mongoClient.GetDatabase(SkiServiceDatabaseSettings.Value.DatabaseName);
    _skiServiceCollection = mongoDatabase.GetCollection<Client>(SkiServiceDatabaseSettings.Value.SkiServiceCollectionNameClient);
}
```

Abbildung 2: Neue Datenbank Verbindung

**Modelldefinition:** Datenmodelle als Klassen definiert mit den benötigten Eigenschaften und Attributen.

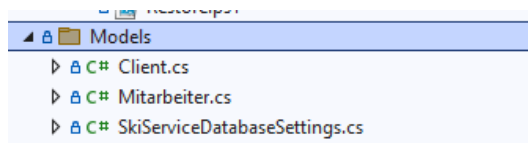


Abbildung 3: Models Ordner

**Service-Schicht:** Services erstellt, die als Vermittler zwischen dem Controller und der Datenbank fungiert. Ich wollte hier Dependency Injection verwenden, um Abhängigkeiten zu verwalten aber die Zeit hat nicht gereicht.

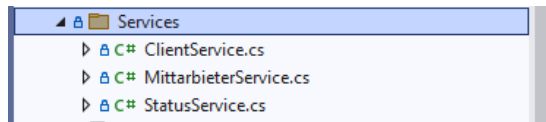


Abbildung 4: Services Ordner

**Controller-Schicht:** Controller Erstellt für jede Funktionalität, die ich implementieren möchte. Ich habe hier Methoden wie Get, Post, Put und Delete für die CRUD-Operationen verwendet.

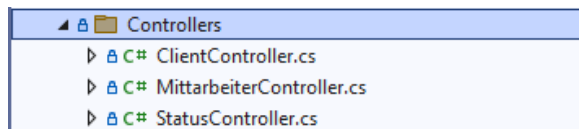


Abbildung 5: Controllers Ordner

**Authentifikation und Autorisierung:** Ich habe die Implementierung einer API-KEY-Authentifikation vorgenommen. Der Benutzer wird in der User Collection in der MongoDB Datenbank verwaltet. Durch die Authentifikation kann auf alle CRUD-Operationen zugegriffen werden, ohne Authentifikation ist nur der Zugang zur Anmeldung und zur Erstellung von Registrierungen verfügbar.

**Testen und Debuggen:** Als letztes werde ich die API Testen, indem ich sie aufrufe und die Ergebnisse prüfe. Dafür verwende ich Debugging-Tools wie den Integrierten Entwickler-Server und den Debugger, um Fehler zu finden und zu beheben.

## Migration der Daten/ Backup und Restore

Im Einzelnen sieht das Vorgehen folgendermassen aus:

**Migration:** Für die Migration habe ich Skripte erstellt, um Daten durch MongoDB und SQL zu migrieren, sowie PowerShell Skripte für Backup und Restore der Datenbank. Die Skripte fragen nach Benutzereingaben und verwenden die Tools mongodump und mongorestore. Der "bin" Folder von MongoDB Tools muss zu den Environment Variables hinzugefügt werden, damit die Skripte funktionieren. Um das geplante Berechtigungskonzept umzusetzen, habe ich eine MongoDB Skriptdatei erstellt, die zwei Benutzer erstellt - einen mit Lese- und Schreibrechten und einen mit nur Leserechten. Die Anmeldung des Benutzers kann entweder nach oder während der Verbindung mit der Datenbank erfolgen. In unserer Web API melden wir uns mit dem Benutzer an, der Lese- und Schreibrechte hat.

**Backup:** Hier habe ich den Befehl "mongodump" auf der Kommandozeile eingegeben, um ein Backup der Datenbank zu erstellen. Im Befehl ist das Backup-Verzeichnis in einem Neuen Ordner, auf dem die Daten wiederhergestellt werden sollen.

**Backup wiederherstellen:** Dafür habe ich den Befehl "mongorestore" auf der Kommandozeile, um das Backup auf den neuen Server wiederherzustellen. Und anschliessend habe ich die Wiederherstellung überprüft, ob alle Daten wie erwartet wiederhergestellt wurden.

## Postman Request erstellen

Hier beschreibe ich wie ich mit Postman vorgegangen bin.

**Neue Collection erstellen:** In Postmann habe ich zuerst eine neue Collection erstellt, in der ich alle meine Requests speichern können.

**Neuen Request erstellen:** Danach habe ich auf die neue Collection ein Neues Request auf den Button "New" um einen neuen Request zu erstellen.

**Request-Methoden auswählen:** Anschliessend habe ich die HTTP-Methode ausgewählt, welche ich für meine Request verwenden möchten (GET, POST, PUT, DELETE, etc.).

**URL eingeben: Geben:** Bei jedem Request musste ich die URL meine API eingeben, die ich testen möchte.

**Request-Header einstellen (optional):** Request-Header einstellen, beispielsweise einen Autorisierungs-Token oder Inhalts-Typ im Request mitzugeben.

**Request-Body (optional):** Wenn es sich um einen POST- oder PUT-Request handelt, können kann ich auch einen Request-Body hinzufügen, um Daten an die API zu senden.

**Ergebnis prüfen:** Und zum Schluss habe ich überprüft ob das Ergebnis, um sicherzustellen, dass es mit meine Erwartungen übereinstimmt.

## Kontrollieren

Dieser Prozess beanspruchte die meiste Zeit, da es das Testen der Anwendung beinhaltete. Es gab ständig Fehler, die ich beheben musste. Ich verwendete den Debugger, um die Probleme zu lokalisieren. Ich überprüfte, ob alle Projektanforderungen erfüllt waren, und ob die gesamte Logik funktionierte. Schliesslich habe ich die Try-Catch-Methode implementiert, um eventuelle Fehlermeldungen abzurufen.

## API mit Postman Testen

Ich habe mein Web API mithilfe von Postman und meiner WPF-Anwendung getestet. Für Postman habe ich eine neue Collection im GitHub-Repository erstellt. Um die API mit meiner WPF-Anwendung zu testen, musste ich nur die URL ändern und den Datentyp der ID auf String ändern. Diese Änderungen sind auch im GitHub-Repository zu finden.

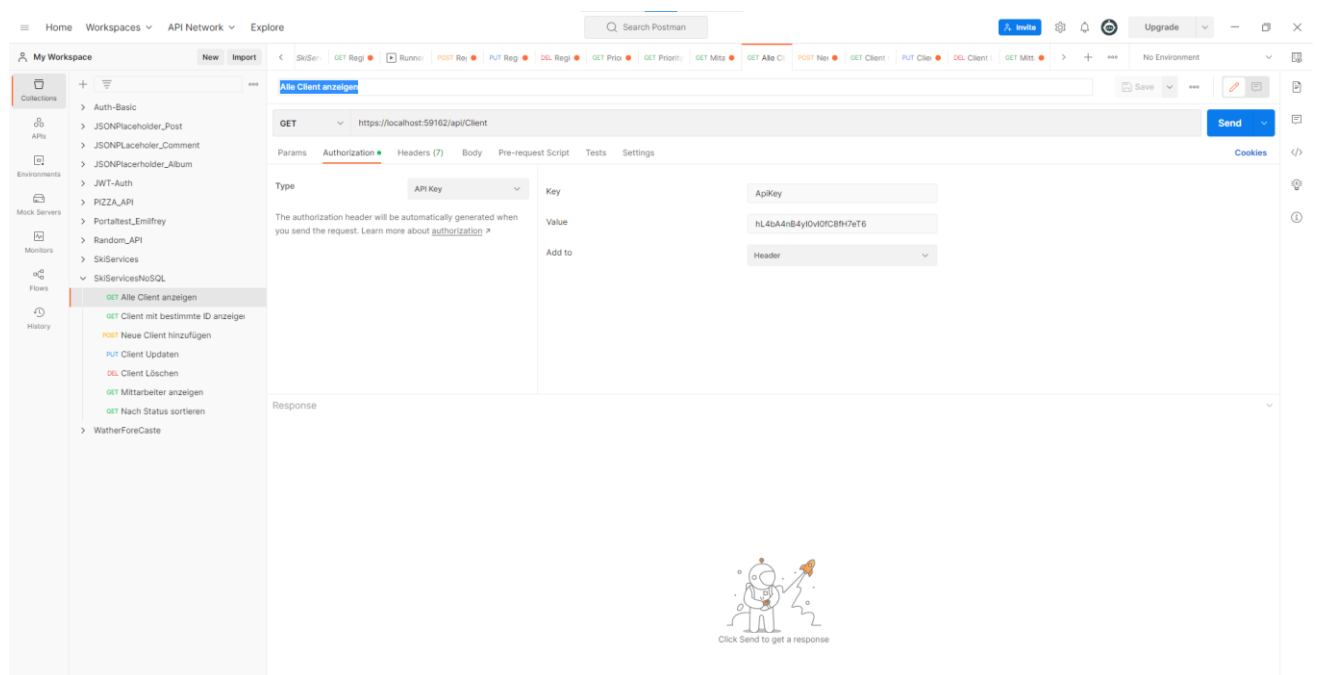


Abbildung 6: Postman Fenster

## Alle HTTP Methode Testen

Um die Integrität der HTTP-Anfragen zu gewährleisten, verwendete ich Postman, um jede einzelne Anfrage zu testen. Obwohl es anfänglich nicht perfekt funktionierte, musste ich meinen Code anpassen, insbesondere die Verbindung mit der MongoDB-Datenbank, um die gewünschten Ergebnisse zu erzielen. Ich stellte sicher, dass alle Anfragen korrekt ausgeführt werden und dass das Ergebnis den Erwartungen entspricht, bevor ich mit dem nächsten Schritt fortfuhr. Durch kontinuierliche Anpassungen und Tests war ich in der Lage, eine reibungslose Funktionalität sicherzustellen.

## Anforderungen erfüllt

Ich habe den Fortschritt des Projekts regelmässig überprüft, um sicherzustellen, dass alle Anforderungen erfüllt werden. Trotzdem habe ich manchmal den Fokus verloren und Zeit für unerforderliche Aufgaben verschwendet. Am Ende konnte ich jedoch alle Anforderungen erfüllen und stellte sogar fest, dass ich zwei zusätzliche Anforderungen erfüllt hatte.

## Auswerten

Eines der Dinge, die ich verbessern könnte, ist die Zeitmanagement, um eine bessere Projektplanung zu erreichen und meine Fähigkeiten zu verbessern, um Herausforderungen zu bewältigen. Ich denke auch daran, wie ich mein Code modular gestalten und einfacher lesbar machen kann, um in Zukunft schnellere Fehlerbehebungen und Wartung zu ermöglichen. Eine weitere Möglichkeit wäre, die Benutzerfreundlichkeit der API zu verbessern, indem ich eine intuitive und einfach zu verwendende Schnittstelle bereitstelle, die Benutzer ermöglicht, mit der API effektiver zu arbeiten.

## Schlussfolgerung

Ich bin zu dem Schluss gekommen, dass diese Arbeit sehr fordernd war und erfordert viel Geduld und Fachwissen. In Zukunft werde ich früher mit dem Projekten beginnen und eine gezielte Arbeitsweise anstreben. Ein Aspekt, den ich leider nicht wie geplant umsetzen konnte, ist das effektive Dokumentieren meiner Arbeit. Eine Vorlage zu erstellen, um das Dokumentieren von Anfang an in Angriff zu nehmen, wäre ein guter Ansatz gewesen, um meine Arbeit besser zu organisieren.

## Zukunft dieser API

Eine weitere Möglichkeit wäre auch, ein ausführliches Sicherheitskonzept für die API zu erstellen, um sicherzustellen, dass sensitive Daten geschützt bleiben. Auch die Möglichkeit für Automatisierungstests könnte hinzugefügt werden, um die Integrität der Daten zu überprüfen und sicherzustellen, dass die API stabil und zuverlässig funktioniert. Ein effizientes Monitoring-System könnte auch hinzugefügt werden, um die API-Performance im Auge zu behalten und sicherzustellen, dass sie immer verfügbar und leistungsstark ist.

## Fazit

Ich habe bei diesem Projekt viel Freude empfunden und unglaublich viel gelernt. Das Schreiben von C#-Programmen war für mich begeisternd und die erfolgreiche Umsetzung meines Codes hat mich immer wieder motiviert.

Leider hatte ich nicht genug Zeit, um das Projekt noch weiter zu verbessern und alle zusätzlichen Anforderungen zu erfüllen. Dennoch war es für mich eine grosse Herausforderung, da wir in kurzer Zeit viel lernen und umsetzen mussten. Der Umfang des Projekts war die grösste Herausforderung für mich. Trotz allem liebe ich C# und werde auch weiterhin an diesem Projekt arbeiten.

C# Macht Spass!

## Quellenverwies

- <https://hevodata.com/learn/mongodb-export-to-json/>
- <https://www.mongodb.com/docs/>
- <https://stackoverflow.com/>
- <https://docs.debart.com/data-pump/exporting-data/export-to-json.html>
- <https://studio3t.com/knowledge-base/articles/sql-to-mongodb-migration/>
- <https://www.mongodb.com/blog/post/simplifying-data-migrations-legacy-mongodb-atlas-studio-3t-hackolade>
- <https://stackoverflow.com/questions/32699431/migrate-from-sql-to-mongodb>