PROJEKTDOKUMENTATION JETSTREAM SKISERVICE API - MONGODB

IBZ, ICT MODUL 165

Alexander Ernst

Inhaltsverzeichnis

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 Neu-eröffnungen 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.

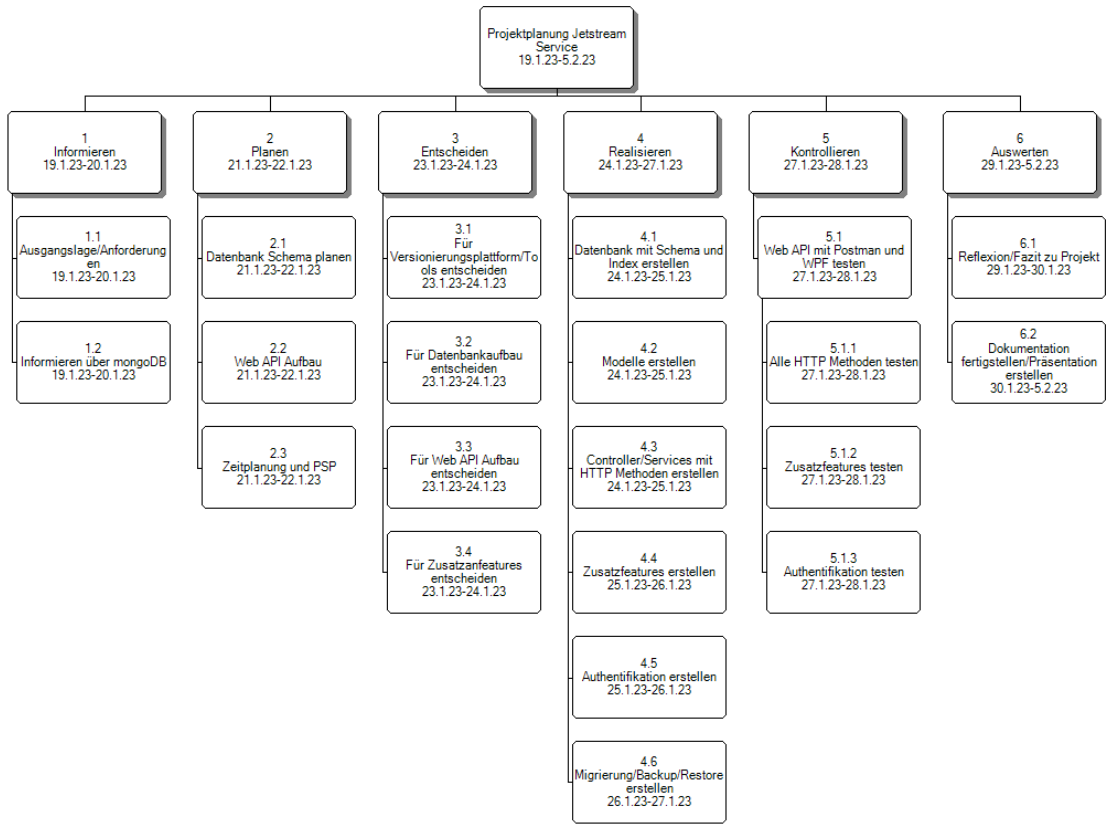
Anforderungen

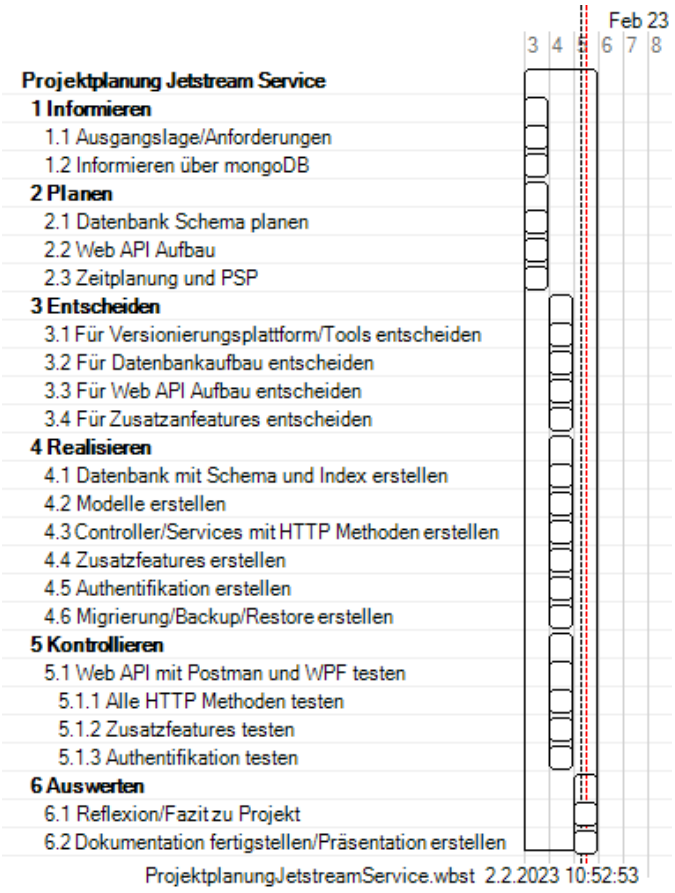
- Das Auftragsmanagement muss folgende Funktionen zur Verfügung stellen:
- Login mit Benutzername und Passwort
- Anstehende Serviceaufträge anzeigen (Liste)
- Bestehende Serviceaufträge mutieren. Dazu stehen folgende Stati zu Verfügung: Offen, InArbeit und abgeschlossen
- Aufträge löschen (ggf. bei Stornierung)

Nr.	Beschreibung
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

Zeitplanung/PSP

Mit WBSTool Version 2.4.0.0 erstellt





Vorgehensweise

1. Informieren

1.1. Ausgangslage/Anforderungen

Als ersten Arbeitsschritt habe ich mich über die Anforderungen und die Ausgangslage dieses Projektes informiert. Hierzu habe ich die zur Verfügung gestellten Unterlagen verwendet.
Durch diese Recherche konnte ich die Anforderungen und das Grundkonzept der Web-APIs verstehen, und wusste ungefähr, was zu tun war.

1.2. Informieren über Web API

Als Nächstes habe ich mich über Technologien, die ich für dieses Projekt brauchen werde, informiert. Zu diesen gehört einerseits die MongoDB Datenbank, aber auch Faktoren wie die C# MongoDB Driver.
Diese Recherche habe ich einerseits mit dem Unterrichtstoff durchgeführt. Andererseits habe ich aber auch mich im Internet weiter informiert.

2. Planen

2.1. Datenbank Aufbau

Hier habe ich den Aufbau/das Schema der Datenbank geplant. Ich habe mir einerseits überlegt ein Schema zu erstellen, welches Embedded JSON Dokumente enthält, andererseits aber auch ein simpleres Schema ohne Embedding oder Referenzen.
Zusätzlich habe ich hier auch das Berechtigungskonzept und Validierungsmöglichkeiten durch Pflichtfelder, Enums und Minium/Maximum Werte geplant.
In diesem Schritt musste ich auch Backup und Restore Möglichkeiten planen. Zusätzlich musste ich mich auch entscheiden, ob ich Neo4j oder MongoDB verwenden wollte.

2.2. Web API Aufbau

In diesem Schritt habe ich geplant, wie ich, dass Web API aufbauen will, hierzu gehören die DTO Klassen, welche Methoden ich implementiere, und welche Zusatzfeatures ich noch hinzufüge.
Vieles dieser Anforderungen waren mir auch schon vorgegeben, da wir schon eine API mit einer Relationalen Datenbank haben und ich dort schon die Entscheidungen zu Authentifikation/Aufbau getroffen habe.

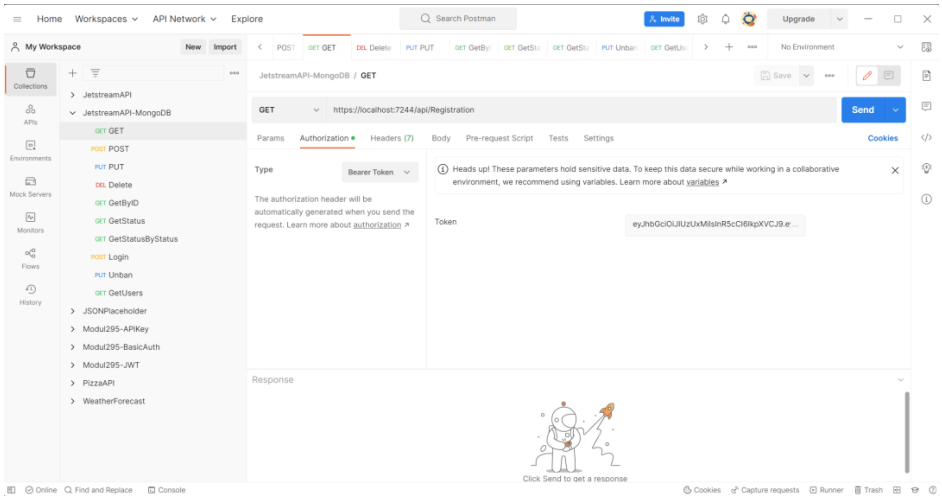
2.3. Zeitplanung und PSP

Als Letztes habe ich eine Zeitplanung/ein PSP erstellt, da ich jetzt eine grobe Planung meines Projektes habe.

3. Entscheiden

3.1. Für Versionierungsplattform/Tools entscheiden

Hier musste ich mich entscheiden, welche Versionierungsplattform/andere Tools ich verwende. Für das Versionieren des Codes habe ich GitHub verwendet, und für das Schreiben des Codes habe ich den IDE Visual Studio verwendet.
Als Testing Tool für die API habe ich Postman gebraucht, da dies eine grosse Funktionalität bietet und für einfaches exportieren von Tests erlaubt. Zusätzlich habe ich auch die WPF Applikation verwendet, da dies ist, was die Kunden der Applikation verwenden werden.
Ich habe ich mich auch noch entschieden einen Logger in das Projekt einzubauen, der Fehler in einer Logdatei loggt.
GitHub Repository: https://github.com/alexanderternst/JetstreamSkiserviceAPI-MongoDB



3.2. Für Datenbankaufbau entscheiden

Als Erstes habe ich mich entschieden MongoDB zu verwenden, weil Neo4j für dieses Projekt nicht viel Sinn ergibt, da wir in diesem Projekt fast keine Verbindungen/Vernetzungen haben. Weiterhin bin ich mit der Neo4j C# Verbindung nicht sehr familiärer.
Als Nächstes musste ich mich für den Aufbau/das Schema der Datenbank entscheiden.
Hier habe ich mich entschieden, zwei Collections zu erstellen, eine für Registrationen und eine für Benutzer. In diesen Collections habe ich keine Embedded Dokumente oder Referenzen auf andere Dokumente.
Für dies habe ich mich bewusst entschieden, da diese API keine Embedded/Referenzierte Dokumente braucht, und dies die ganze API viel komplexer machen würde. Dies würde zu einer unnötigen Komplexität führen, die die API gar nicht braucht.
Ich habe mich auch dafür entschieden eine Schemavalidierung, hier habe ich eine kleine Übersicht über die Validierungen, die eingesetzt wurden:
Registrationen Collection:

- Name, Priorität, Status, Dienstleistung 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"

User Collection:

- Username, Password, Counter, Rolle sind Pflichtfelder
- Username und Counter sind indexiert
- Eigenschaften müssen sinnvollen Datentyp haben
- Counter hat einen Minimumwert von 0 und einen maximalen Wert von 3
- Eigenschaft Rolle darf nur die Werte, «level1», «level2» oder «level 3 haben.

Dies Validierungen habe ich einerseits eingesetzt, damit Daten nicht im falschen Format eingegeben werde und damit Pflichtfelder nicht leer gelassen werden und nichtexistierende Status nicht eingetragen werden.

3.2.1. Skripte für das Migrieren von Daten

Zusätzlich habe ich mich auch entschieden Skripte für das Migrieren von Daten von SQL auf MongoDB in Form eines SQL und MongoDB Skriptes zu erstellen.

3.2.2. Skripte für das Backup/Restore

Um Backup und Restore Möglichkeiten durchzusetzen, habe ich zwei PowerShell Skripte erstellt, einen für das Backup und einen für die Restore Möglichkeit.
Damit diese Skripte funktionieren muss der «bin» Folder der MongoDB Tools den Environment variables hinzugefügt werden, da wir hier die Tools mongodump und mongorestore verwenden.
Diese beiden Skripte fragen den Benutzer nach Angaben wie Datenbank und Dateipfad und führen dann Backup/Restore durch.

3.2.3. Berechtigungskonzept

Bei dem Berechtigungskonzept habe ich mich entschieden zwei Datenbankbenutzer zu erstellen, einer dieser Nutzer hat lese und schreib rechte, einer der Nutzer hat nur Leserechte.
Dies habe ich so getan, da man vielleicht auch mal dem Kunden den Datenbankzugriff will geben, der Kunde soll aber die Datenbank nicht modifizieren können.
Um diese Benutzer zu erstellen habe ich auch einen MongoDB Skript, das Login durch den Nutzer kann man entweder nach der Verbindung oder während der Verbindung herstellen, mehr Informationen hierzu finden Sie auch in der Skriptdatei.
Zusätzlich habe ich auch eine Authentifikation durch ein JWT Token erstellt, welches man durch ein Login mit Benutzer und Passwort abrufen kann. Mit diesem JWT-Token hat man Zugang auf alle CRUD Befehle, ohne dieses kann man nur Registrationen erstellen und sich anmelden.
Auf wieso ich das JWT Token gewählt habe, werde ich hier nicht mehr eingehen, dies finden Sie in der Dokumentation zum anderen API.
Der Login durch das JWT-Token existiert, da es vielleicht auch Mitarbeiter bei Jetstream Service gibt, die nur Aufträge erstellen können, und diese nicht modifizieren/lesen können.

3.3. Für Web API Aufbau entscheiden

Wie schon erwähnt, hatte ich mich in dem letzten Projekt bereits für Projektaufbau und Authentifikation entschieden. Hier ist nochmals eine Übersicht der wichtigsten Punkte.

- Projekt mit Modellen, Controllern und Services umgesetzt für Wiederverwendbarkeit und Lesbarkeit
- Exception Handling mit Logger für Fehlerbehandlung/Diagnostikzwecke
- Dependency Injection bei Services verwendet für Wiederverwendbarkeit
- JWT-Token für Authentifikation, da dies das Sicherste und das geeignete für einen Benutzerlogin ist.
- 3 Controller (mit zugehörigen Services) für Status, Registrationen, Benutzer

Wichtig noch zu notieren hier ist das ich mich entschieden habe alle Variablen/Klassen/Methoden/Eigenschaften nach der camelCase Konvention mit Aussagekräftigen Namen zu benennen. Zusätzlich habe ich mich auch entschieden den ganzen Code mit XML

Kommentaren zu verlegen. Des habe ich getan, um die Lesbarkeit und Verständlichkeit des Codes zu verbessern.

3.4. Für Zusatzfeatures entscheiden

Hier habe ich mich entschieden, welche Zusätze ich durchsetzen wollte.
Ich habe mich hier, für das Bannen von Benutzern nach 3 falschen versuchen (auch entsprechende Entbannung Funktion), und den Kommentar bei Registrationen entschieden. Zusätzlich habe ich auch noch eingebaut, dass man über Status auf Registrationen zugreifen kann.

4. Realisieren

4.1. Datenbank mit Schema und Index erstellen

In diesem Schritt habe ich die MongoDB Datenbank und das Schema mit Validierungsmöglichkeiten mit einer Skriptdatei erstellt, die Beschreibung des Schemas finden Sie unter Punkt 3.2. Zusätzlich habe ich auch die Indexe erstellt.

4.2. Modelle erstellen

Als Nächstes habe ich die Modelle für den Datenbankzugang/Ausgabe der Daten erstellt. Hier habe ich ein Modell für die Registrationen, eines für die Benutzer und ein letztes mit Konfigurationsdaten für die Datenbankverbindung.
In diesen Modellen habe ich einerseits die Eigenschaften, aber auch JSONPropertyName für die Ausgabe und BsonElemente für die Eingabe. Dies habe ich so getan, damit ich in der MongoDB Datenbank, im C# Code und in der Ausgabe andere Eigenschaftsnamen verwenden kann. Da MongoDB per Namenskonvention Eigenschaften kleinschreibt und C# diese nach Konvention mit camelCase schreibt.

4.3. Controller/Services mit HTTP Methoden erstellen

Bei diesem Schritt habe ich die Controller und Services mit den dazugehörigen HTTP Methoden erstellt. Ich habe 3 Controller mit zugehörigen Services erstellt. Einen für Registrationen mit POST, UPDATE, DELETE, GET (all), GET (by ID) Methoden erstellt. Einen für Benutzer, mit einer Methode für das Entbannen, das Login und das Abrufen von Benutzern. Der letzte Controller/Service ist für das Aktualisieren von nur Status, die Abfrage von Registrationen nach Status, und um Sie nach Status zu sortieren.
Für das Erstellen der Services habe ich Dependency Injection verwendet, die Begründung für dies finden Sie unter Punkt 3.3. Zusätzlich habe ich auch Exception Handling eingebaut, welcher Fehler in einer Logdatei dokumentiert.

4.3.1. Versionen und Softwarevoraussetzungen

Dieses Projekt wurde auf *.NET 6.0* aufgesetzt/erstellt und folgende NuGet Pakete (mit Versionen) wurden/kamen installiert:

- Microsoft.AspNetCore.Authentication.JwtBearer, *Version=6.0.13*
- MongoDB.Driver, *Version=2.18.0*
- Serilog.AspNetCore, *Version=6.1.0*
- Swashbuckle.AspNetCore, *Version=6.5.0*

Visual Studio 2022 ist bei Erstellung dieses Projektes auf *Version 17.4.4*
Auf dem Rechner, mit dem das Projekt erstellt wurde, läuft eine Version von *Microsoft Windows 11.*

4.4. Zusatzfeatures erstellen

In diesem Schritt habe ich die Zusatzfeatures erstellt. Hier habe ich die Bannung und die entsprechende Entmannung Funktion erstellt. Diese funktionieren mit einem Counter in der Datenbank, der einen minimalen Wert von 0 und einen maximalen Wert von 3 hat. Wenn man

Benutzer richtig aber Passwort falsch eingibt, wird der Counter + 1 gesetzt und wenn der Counter 3 erreicht hat, ist der Benutzer gebannt und muss entbannt werden.

4.5. Authentifikation erstellen

Hier habe ich die JWT-Authentifikation erstellt, der Benutzer befindet sich in der User Collection in der MongoDB Datenbank. Mit der Authentifikation kann man auf die volle Selektion der CRUD Befehle zugreifen. Ohne die Authentifikation hat man nur Zugang auf die Methode, um sich anzumelden und Registrationen zu erstellen.
Zusätzlich habe ich in der Benutzer Collection einen Counter für das Bannen/Entbannen (Mehr Informationen, Kapitel 4.4.). In dieser Collection habe ich auch eine Level-Eigenschaft, in der das Level des Benutzers gespeichert wird.

4.6. Migrierung/Backup/Restore erstellen

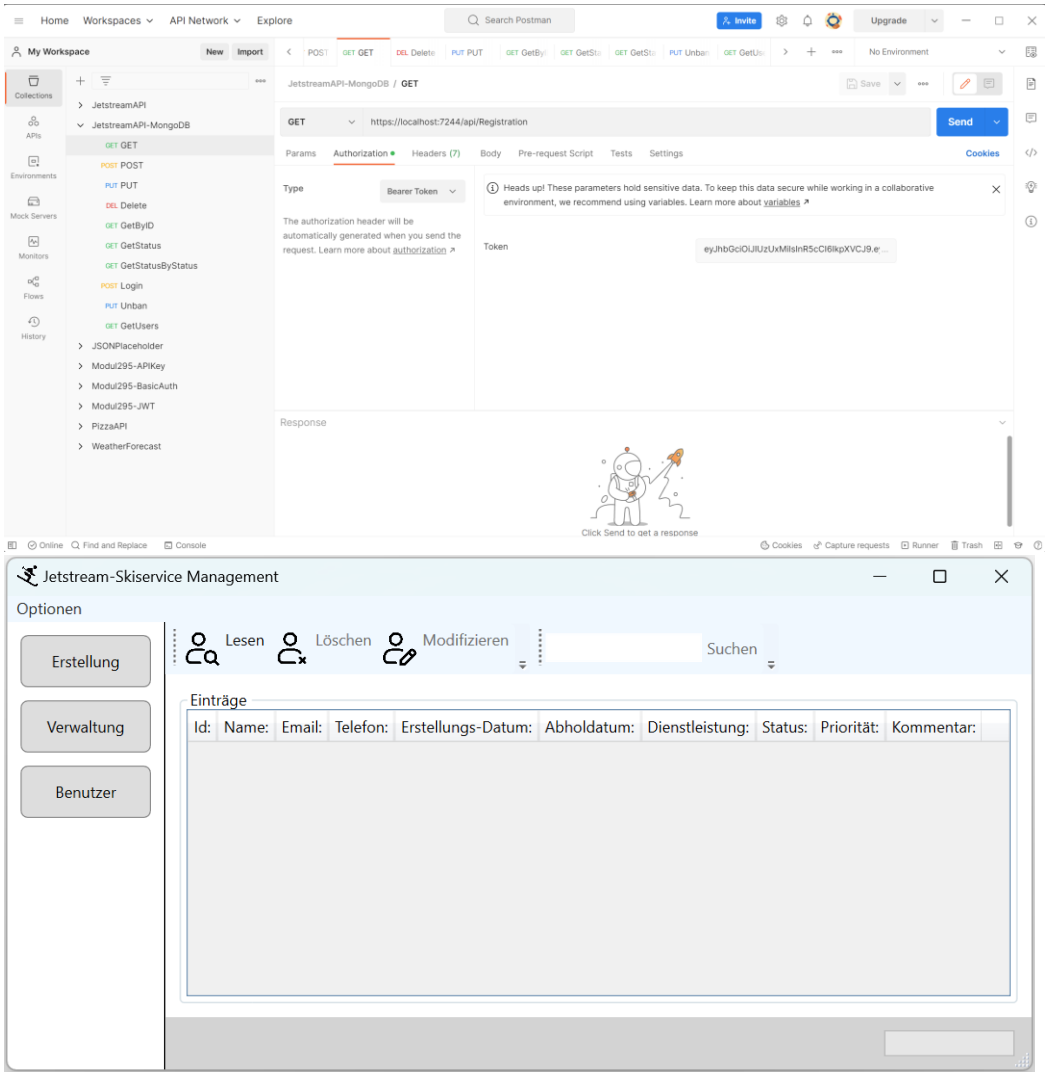
Hier habe ich Skripte für die Migrierung von Daten durch MongoDB und SQL erstellt. Danach habe ich auch noch PowerShell Skripte für das Backup und Restore der Datenbank erstellt, welche nach Benutzereingaben fragen und dann die Tools mongodump und mongorestore verwenden. Damit diese Skripte funktionieren muss der «bin» Folder der MongoDB Tools den Environment variables hinzugefügt werden.

Um mein geplantes Berechtigungskonzept durchzusetzen habe ich eine MongoDB Skriptdatei erstellt, welche zwei Benutzer erstellt, einen mit Lese- und Schreibrechten und einen mit nur Leserechten.
Das Login durch den Nutzer kann man entweder nach der Verbindung oder während der Verbindung der Datenbank herstellen, mehr Informationen hierzu finden Sie auch in der Skriptdatei.
In dem Web API melden wir uns mit dem Benutzer an, der Lese- und Schreibrechte hat.
Mehr Informationen zur Erstellung der JWT-Authentifikation finden Sie unter dem Punkt 4.5.

5. Kontrollieren

5.1. Web API mit Postman und WPF Applikation testen

In diesem Schritt habe ich mein Web API mit Postman getestet und mit meiner WPF Applikation getestet. Für Postman habe ich eine neue Collection erstellt, welche sich ihm GitHub Repository befindet. Um die API mit der WPF Applikation zu testen, musste ich nur die URL ändern und den Datentyp der ID auf einen String wechseln. Die Änderungen zur WPF Applikation befinden sich auch auf meinem GitHub Repository.



6. Auswerten

6.1. Reflexion/Fazit zu Projekt

Ich denke, ich habe die Hauptanforderungen dieses Projektes geschafft, da das Web API mit Authentifikation und sogar mit manchen Zusatzfeatures komplett lauffähig ist.
Ich habe durch dieses Projekt auch sehr viel über MongoDB und manches Neues über C# gelernt, und konnte mein Wissen um einiges erweitern.
Was meiner Meinung in diesem Projekt nicht so gut lief, war das ich nicht so viele der angegebenen Zusatzanforderungen durchgeführt habe, da ich sonst schon unter einem grossen Zeitstress stand.
Im Allgemeinen lief dieses Projekt aber schon sehr gut und ich habe viel Neues vor allem über MongoDB gelernt.

6.1.1. Was habe ich nicht geschafft/Verbesserungen

Was ich gerne noch geschafft hätte, wäre die Zusatzanforderung mit einer statistischen Auswertung, wie schon erwähnt Stande ich unter Zeitstress und hatte keine Zeit für diese Anforderungen.

6.2. Dokumentation fertigstellen/Präsentation erstellen

Als aller letzten Arbeitsschritt habe ich die Dokumentation nochmals überarbeitet und fertiggestellt, und eine Präsentation mit Markdown (Marp Framework/https://marp.app/) erstellt.