Alexander Ernst

Projektdokumentation Jetstream Skiservice API

Inhaltsverzeichnis

[Ausgangslage 2](#_Toc119096885)

[Anforderungen 2](#_Toc119096886)

[Zeitplanung/PSP 3](#_Toc119096887)

[Vorgehensweise 4](#_Toc119096888)

[1. Informieren 4](#_Toc119096889)

[1.1. Ausgangslage/Anforderungen 4](#_Toc119096890)

[1.2. Informieren über Web API 4](#_Toc119096891)

[2. Planen 4](#_Toc119096892)

[2.1. Datenbank Aufbau 4](#_Toc119096893)

[2.2. Web API Aufbau 4](#_Toc119096894)

[2.3. Zeitplanung und PSP 4](#_Toc119096895)

[3. Entscheiden 5](#_Toc119096896)

[3.1. Für Versionierungsplattform/Tools entscheiden 5](#_Toc119096897)

[3.2. Für Web API Aufbau entscheiden 5](#_Toc119096898)

[3.3. Für Datenbankaufbau entscheiden 6](#_Toc119096899)

[3.4. Für Zusatzanforderungen entscheiden 7](#_Toc119096900)

[3.5. Für Web API Authentifikation entscheiden 7](#_Toc119096901)

[4. Realisieren 8](#_Toc119096902)

[4.1. Datenbank erstellen 8](#_Toc119096903)

[4.2. DTOs für Zugriff erstellen 9](#_Toc119096904)

[4.3. HTTP Methoden/Services erstellen 10](#_Toc119096905)

[4.4. Zusatzfeatures erstellen 11](#_Toc119096906)

[4.5. Authentifikation erstellen 12](#_Toc119096907)

[5. Kontrollieren 12](#_Toc119096908)

[5.1. Web API mit Postman testen 12](#_Toc119096909)

[6. Auswerten 12](#_Toc119096910)

[6.1. Reflexion/Fazit zu Projekt 12](#_Toc119096911)

[6.2. Dokumentation fertigstellen/Präsentation erstellen 12](#_Toc119096912)

# Ausgangslage

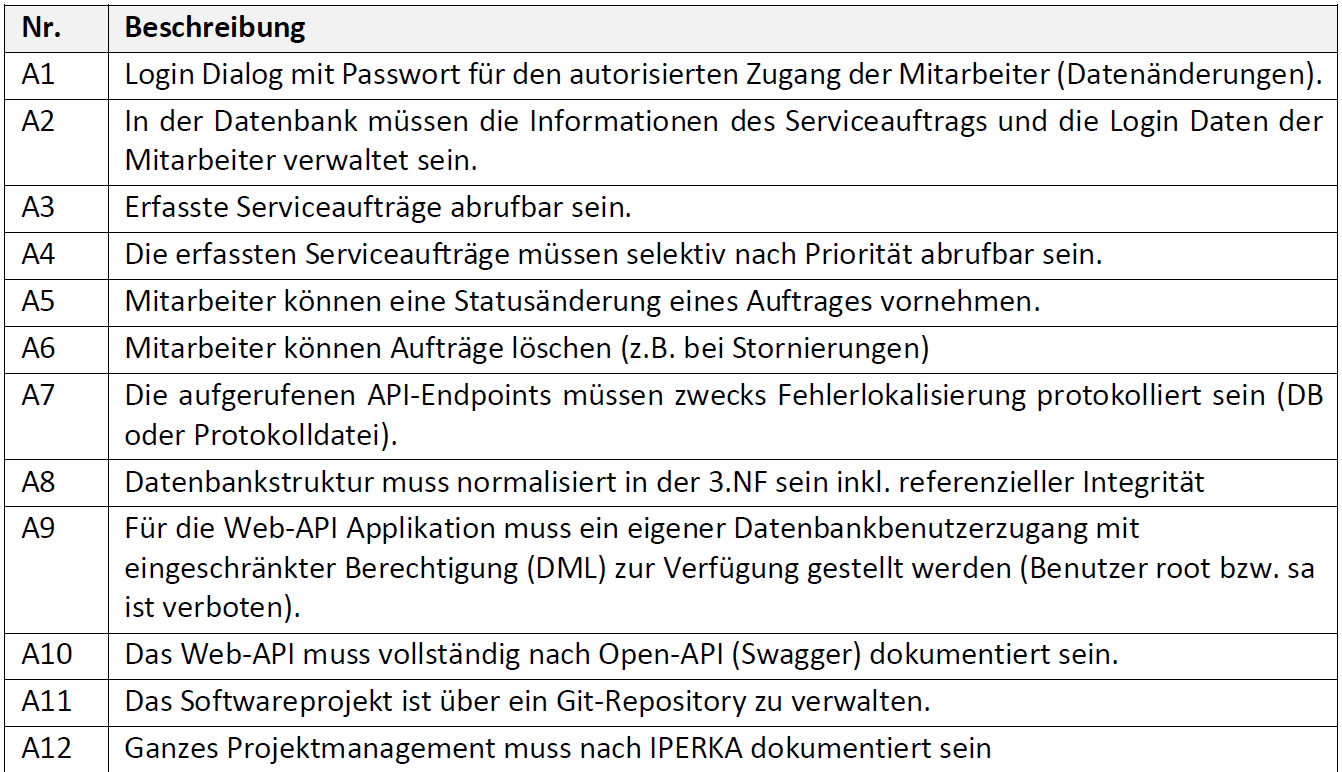
Die Firma Jetstream-Service führt als KMU in der Wintersaison Skiservice Arbeiten durch, will im Zuge der Digitalisierung die interne Verwaltung der Skiservice Aufträge komplett über ein Web und Datenbank basierten Anwendung abwickeln. Die bereits existierende Onlineanmeldung soll bestehen bleiben und mit den erforderlichen Funktionen für das Auftragsmanagement erweitert werden.

In der Hauptsaison sind bis zu 10 Mitarbeiter mit der Durchführung der Serverarbeiten beschäftigt. Diese sollen einen autorisierten passwortgeschützten Zugang zu den anstehenden Aufträgen erhalten und diese zur Abarbeitung übernehmen und ändern können.

# Anforderungen

Das Auftragsmanagement muss folgende Funktionen zur Verfügung stellen:

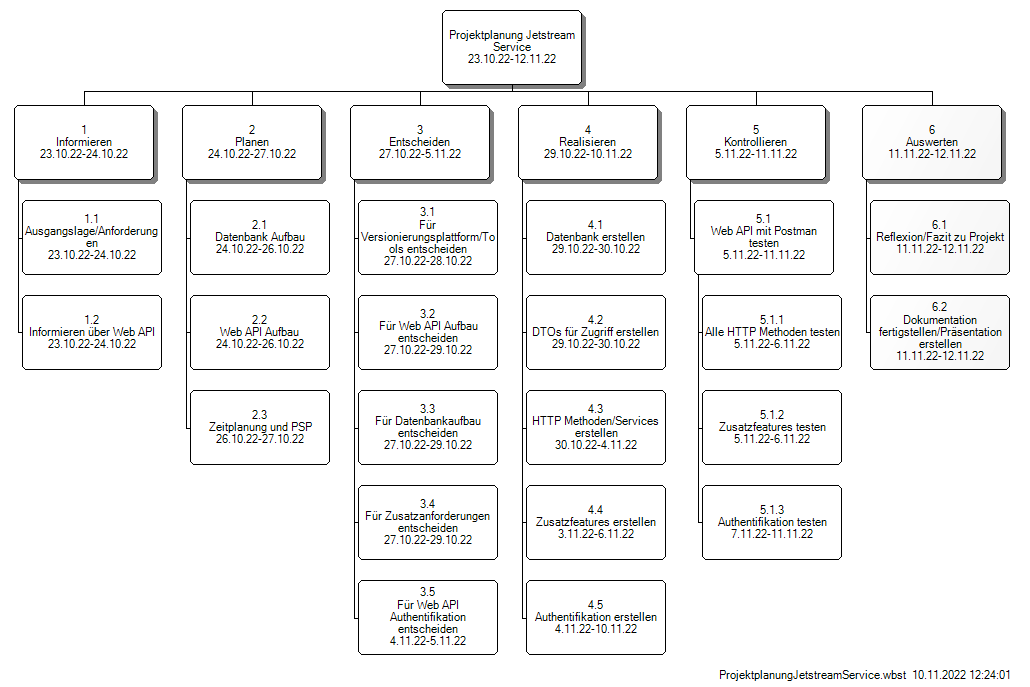
* Login mit Benutzername und Passwort
* Anstehende Serviceaufträge anzeigen (Liste)
* Bestehende Serviceaufträge mutieren. Dazu stehen folgende Stati zur Verfügung: Offen, InArbeit und abgeschlossen.
* Aufträge löschen (ggf. bei Stornierung)



A picture containing text

Description automatically generated

# Zeitplanung/PSP



A picture containing table

Description automatically generated

# Vorgehensweise

## Informieren

### 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.

### Informieren über Web API

Als Nächstes habe ich mich über Technologien, die ich für dieses Projekt brauchen werde, informiert. Dies war aber auch ein laufender Prozess, dies heisst, dass ich mich in dem ganzen Laufe des Projektes weiter informieren/in das Themengebiet vertiefen musste.

Diese Recherche habe ich einerseits mit dem Unterrichtstoff durchgeführt. Andererseits habe ich aber auch mich im Internet weiter informiert.

## Planen

### Datenbank Aufbau

Hier musste ich den Datenbankaufbau planen, dies heisst, wie die Verbindungen zwischen den Tabellen aussehen soll, und ob ich einen Code First oder Database First Ansatz verwende.

### Web API Aufbau

In diesem Schritt musste ich planen, wie ich, dass Web API aufbauen wollte, hierzu gehören die DTO Klassen, welche Methoden ich implementiere, und welche Zusatzfeatures ich noch hinzufüge.

Hier musste ich auch meine Authentifikationsart planen, hier gab es eigentlich 3 Optionen, JWT Token, API Key oder Basic Authentifikation.

Hier habe ich vor und Nachteile jeder Option aufgelistet:

|  |  |  |
| --- | --- | --- |
| **Art** | **Vorteil** | **Nachteil** |
| Basic Authentifikation | Einfach zu implementieren | Unsicher |
| API Authentifikation | Sicher | Ungeeignet für Anmeldung |
| JWT-Token | Sicher, geeignet für Anmeldung | Schwerer zu implementieren |

### Zeitplanung und PSP

Als Letztes konnte ich eine Zeitplanung/ein PSP erstellen, da ich jetzt eine grobe Planung meines Projektes hatte.

## Entscheiden

### 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 mich auch noch entschieden einen Logger in das Projekt einzubauen, der Fehler in einer Logdatei loggt.

GitHub: <https://github.com/alexanderternst/JetstreamSkiserviceAPI>

Graphical user interface, text, application

Description automatically generated

### Für Web API Aufbau entscheiden

Hier habe ich mich entschieden, dass Projekt mit DTO (Data Transfer Object) Klassen umzusetzen, dies heisst, dass ich DTO Klassen benutze, um Daten zu lesen und zu schreiben.

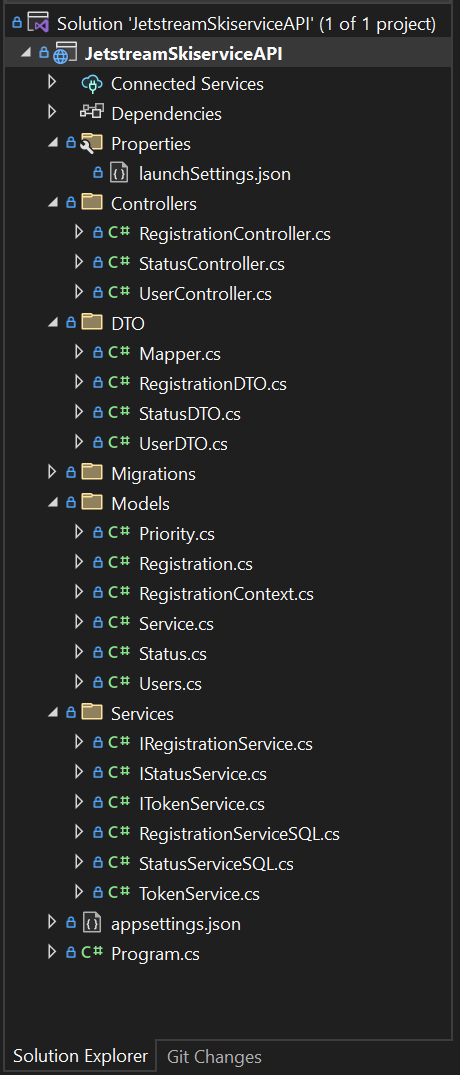
Diese funktionieren so, dass man das, was man schreiben, lesen will, erst in ein DTO Objekt tut, und dieses dann ausgibt/schreibt.

Dies hat den Vorteil, dass, wenn sich die Namen von einer Property ändern, man diese nicht in der Datenbankklasse ändern muss.

Zusätzlich habe ich mich entschieden, die Logik des Programmes in Service Klassen zu verbauen und das ganze Projekt mit Dependency Injection umzusetzen. Dies hat den Vorteil der höheren Wiederverwendbarkeit und weniger Abhängigkeit.

In diesem Projekt habe ich drei Controller, einen für die Verwaltung von Registrationen, einen für das Auslesen von Registrationen nach Status, und einen für die Authentifikation.

Dementsprechend habe ich auch drei Service Klassen mit jeweils einem Interface.



### Für Datenbankaufbau entscheiden

Als Erstes musste ich mir überlegen, ob ich die Datenbank mit einem Code- oder Database-First Prinzip erstelle. In diesem Projekt habe ich mich für die Codefirst Variante entschieden, da dies ein neues Themengebiet war, indem ich noch nicht so viel Erfahrung hatte und ich meine Erfahrung in diesem Bereich erhöhen wollte.

Beim Datenbankaufbau habe ich mich für eine Datenbank mit 4 Tabellen entschieden.

Eine Tabelle mit allen Registrationen, und 3 andere Tabellen (Status, Priority, Service) welche via. Foreign Key, mitden Registrationen verbunden sind. In diesen 3 Tabellen speichern wir Status, Priorität und Dienstleistung mit jeweils einer ID, auf die wir dann von den Registrationen zugreifen können.

Zusätzlich gibt es noch eine Tabelle mit Benutzern, welche sich durch die Authentifikation anmelden können (Mitarbeiter).

****

### Für Zusatzanforderungen entscheiden

Hier habe ich mich entschieden, welche Zusatzanforderungen 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.

### Für Web API Authentifikation entscheiden

Hier habe ich mich für das JWT Token entschieden, da dies meiner Meinung nach für dieses Projekt am meisten Sinn ergibt.

Die Basic Authentifikation wäre zu unsicher und die API Key Authentifikation ist nicht für ein Benutzer-Login geeignet.

Ich habe mich hier entschieden die Benutzer in einer SQL Tabelle zu speichern, da dies ein guter Weg ist mehrere Benutzer zu speichern.

Jetzt musste ich mich noch entscheiden, auf welche Methoden Personen ohne Anmeldung Zugriff haben.

Ich habe es jetzt so durchgesetzt, dass man ohne JWT Token nur auf die POST Klasse zugreifen können, dies heisst normale Benutzer können Aufträge erstellen/ergeben, aber können keine Aufträge löschen/modifizieren/lesen.

Dies macht in diesem Projekt meiner Meinung nach Sinn, da dieses API für eine Registration gedacht ist, und ein Benutzer natürlich nicht andere Registrationen lesen sollte.

## Realisieren

### Datenbank erstellen

Hier habe ich angefangen, die Datenbank mit dem Code First Prinzip zu erstellen.

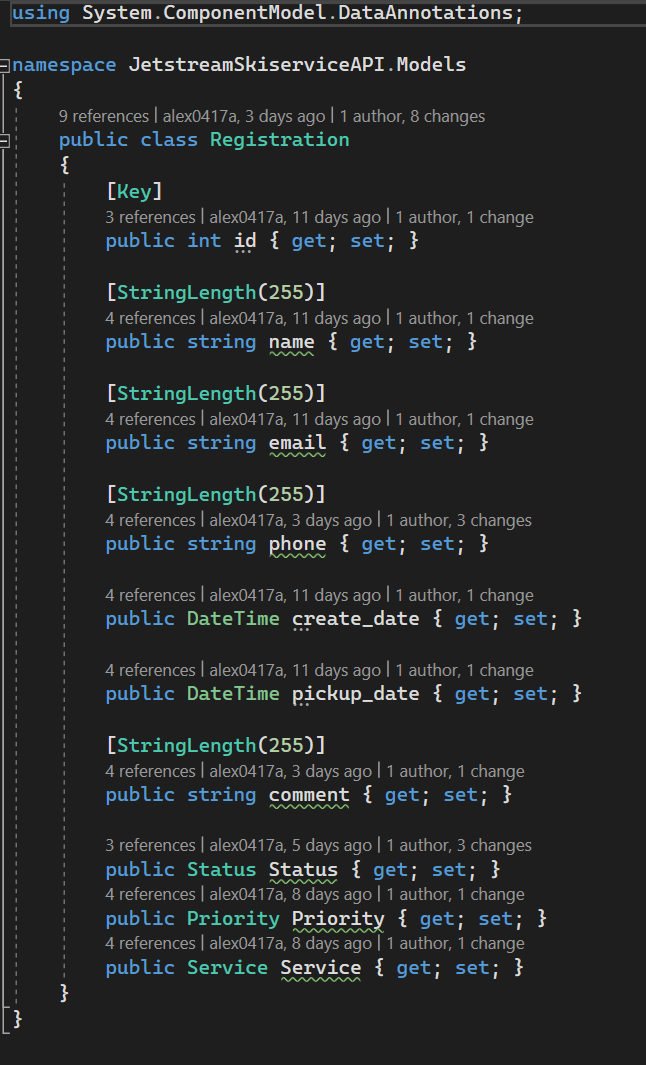
Als Erstes habe ich alle Klassen für die Datenbank erstellt (Registration, Status, Priority, Service, User) und eine Klasse namens RegistrationContext, welche die eigentliche Datenbank erstellt kreiert, der Connection String, der diese Klasse braucht, ist in ein JSON File ausgelagert.

Zusätzlich wird die Datenbankverbindung mit einem SQL User erstellt.

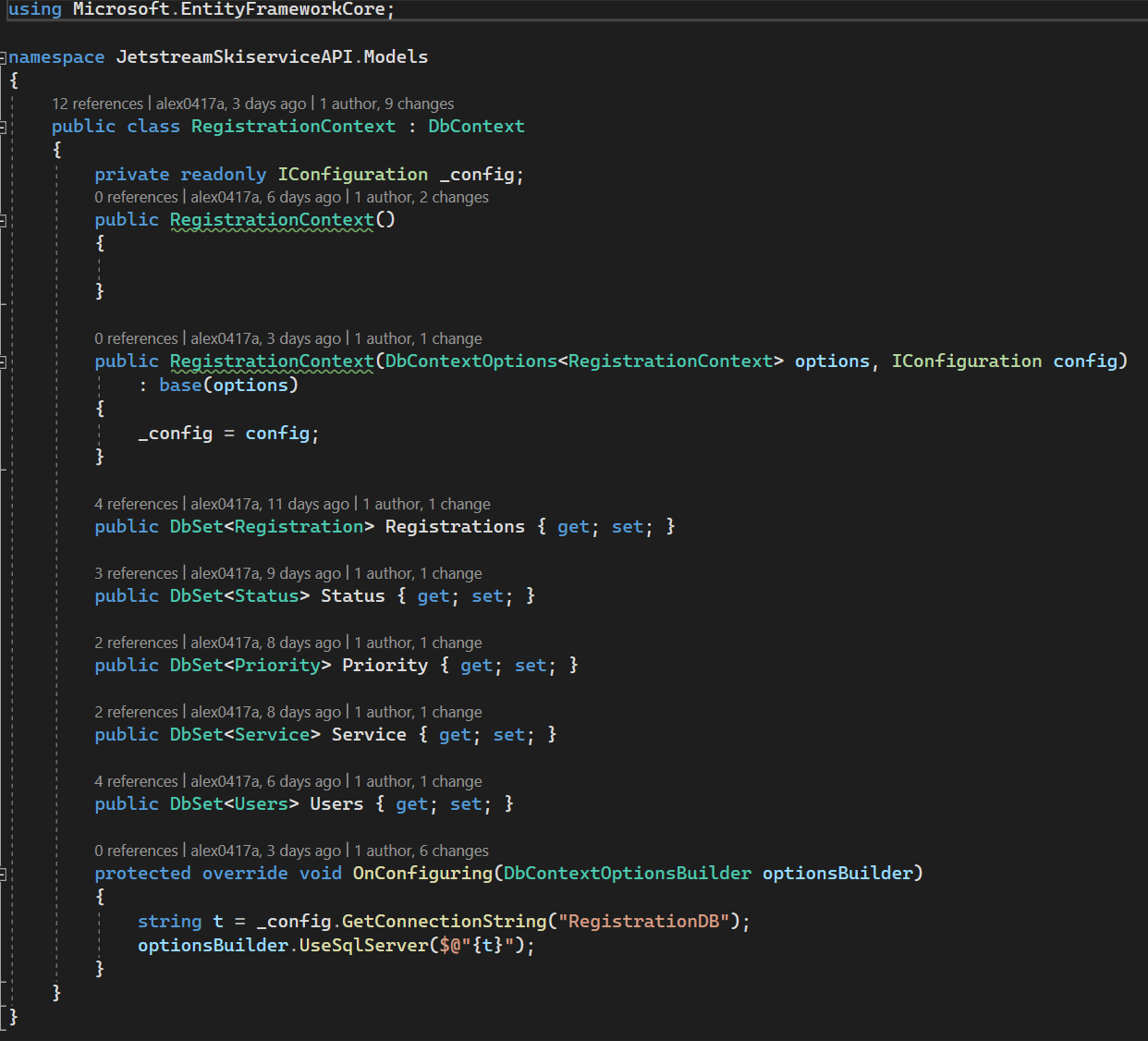
In den untigen Bildern sehen Sie den Aufbau dieser Klassen, einmal habe ich durch eine Eigenschaft (des Datentyp Status, Registration, Service) in der Klasse Registration eine Verbindung von Registrationen auf Stati, Prioritäten und Dienstleistungen.

Andererseits habe ich auch eine Eigenschaft in Form einer Liste (des Datentyp Registration) in den Klassen Status, Service und Priority.

Dies erlaubt uns eine Verbindung von Status auf Registrationen herzustellen, damit wir z.B. alle Registrationen mit Status offen sehen können.

Text

Description automatically generated



Nachdem ich diese Klassen erstellt habe, habe ich mit diesen Befehlen in der Package Manger Console die Datenbank erstellt:

Add-Migration IntialCreate

Update-Database

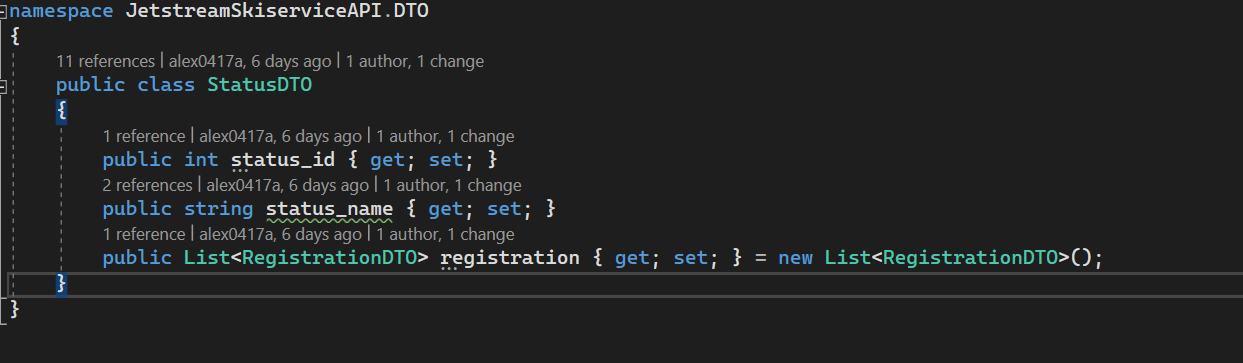
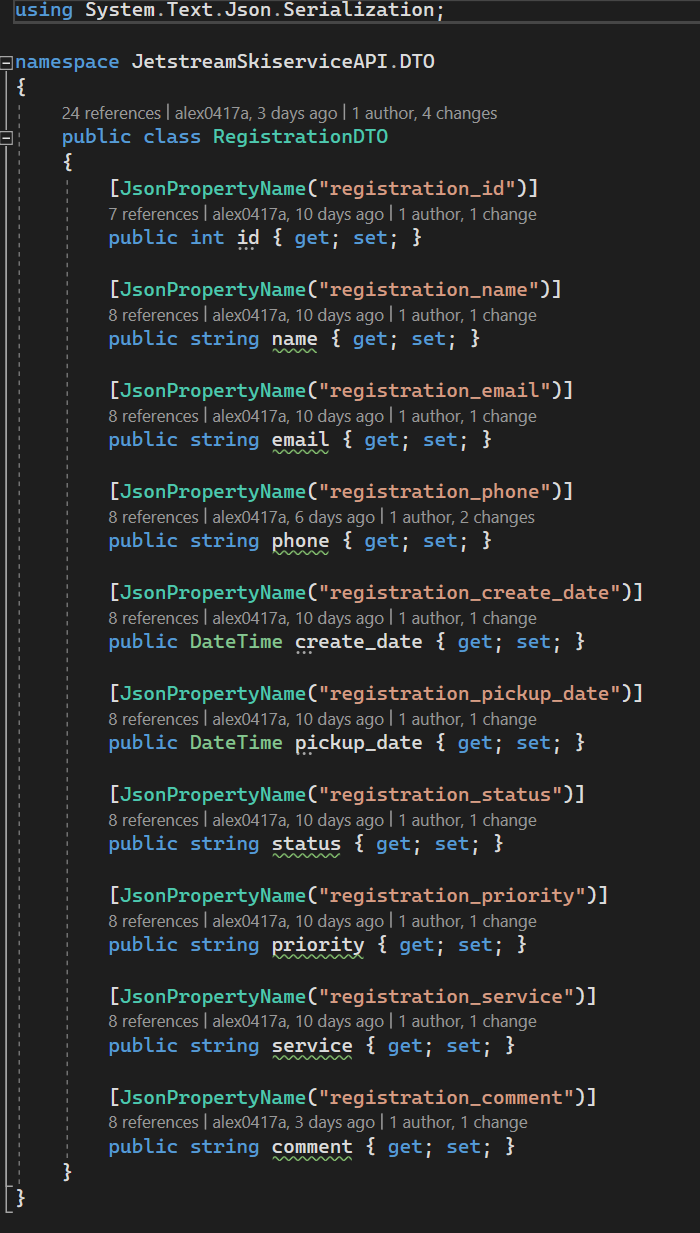
Für das Einfügen von Status, Service, Priority und User habe ich einen SQL Skript, den Sie auf GitHub finden, verwendet.

### DTOs für Zugriff erstellen

Als Nächstes habe ich DTO (Data Transfer Object) Klassen erstellt, über welche wir auf Daten zugreifen/und Daten schreiben.

Diese haben jetzt auch den grossen Vorteil, dass wen ich Registrationen mit ihrem entsprechenden Status ausgeben will, muss ich nicht alles ausgeben, sondern ich kann nur den Inhalt der DTO Klasse ausgeben, was die Ausgabe/Eingabe viel sauberer macht.

Wir haben drei DTO Klassen, eine für das Anmelden (UserDTO), eine für das Auslesen von Stati mit ihren Angehörigen Registrationen (StatusDTO) und eine für unsere Registrationen (RegistrationDTO).



### HTTP Methoden/Services erstellen

Als Nächstes habe ich die Controller und die Serviceklassen erstellt, in den Controllern passiert der Aufruf der HTTP Methoden und in dem Service befindet sich die ganze Logik, die Daten aus der SQL Datenbank liest/in die Datenbank schreibt.

Die Services sind zusätzlich noch mit DI (Dependency Injection) gebaut, dies heisst jede Serviceklasse hat ein Interface (erbt von einem Interface), über welches wir auf den Service zugreifen.

So schaffen wir eine immense Wiederverwendbarkeit und Abhängigkeit, dies heisst die Controller Klasse muss den Service gar nicht kennen, da diese durch das Interface komplett abgekoppelt sind.

Damit Dependency Injection funktioniert müssen wir im Programm.cs das Interface und die Klasse initialisieren und in der Controller Klasse das Interface im Konstruktor instanziieren (Wichtig ist hier das nur das Interface instanziiert wird, und nicht der konkrete Service).

Zusätzlich damit das JWT Token und die SQL Server Verbindung funktioniert müssen wir im Programm.cs auch Services hinzufügen, welche dann in entsprechenden Controllern instanziiert werden.

In diesem Programm haben wir drei Controller, einen für Registrationen, einen für Zugriff auf Registrationen über Status und einer für die Authentifikation.

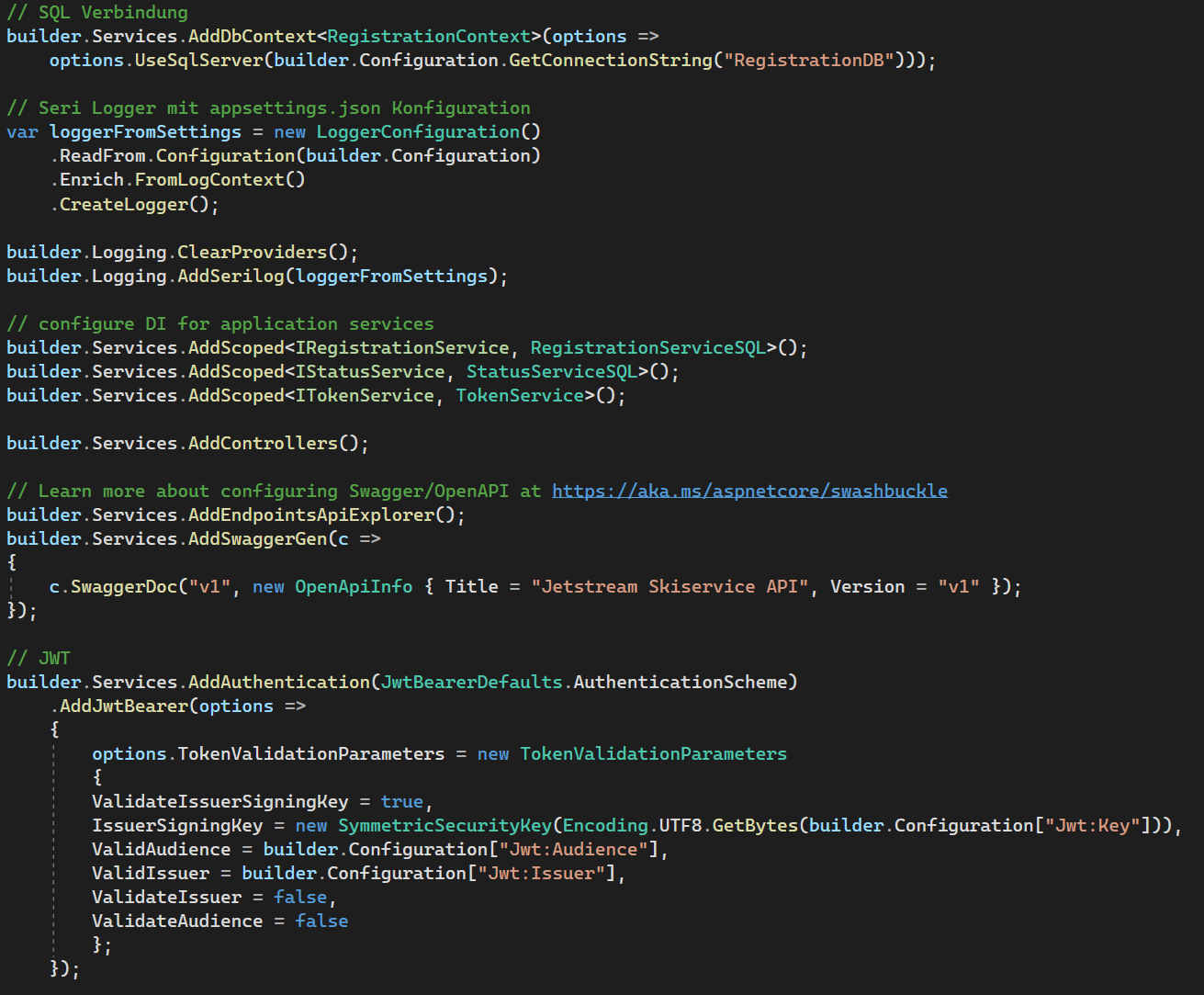
Dementsprechend haben wir auch 3 Services mit entsprechenden Interfaces, auf die wir von den Controllern zugreifen.

Alle Services und Controller haben auch eine try-catch Fehlerbehandlung, sodass das Programm nicht einfach abstürzt und der Controller loggt Fehler mit einem Logger in einer log Datei.

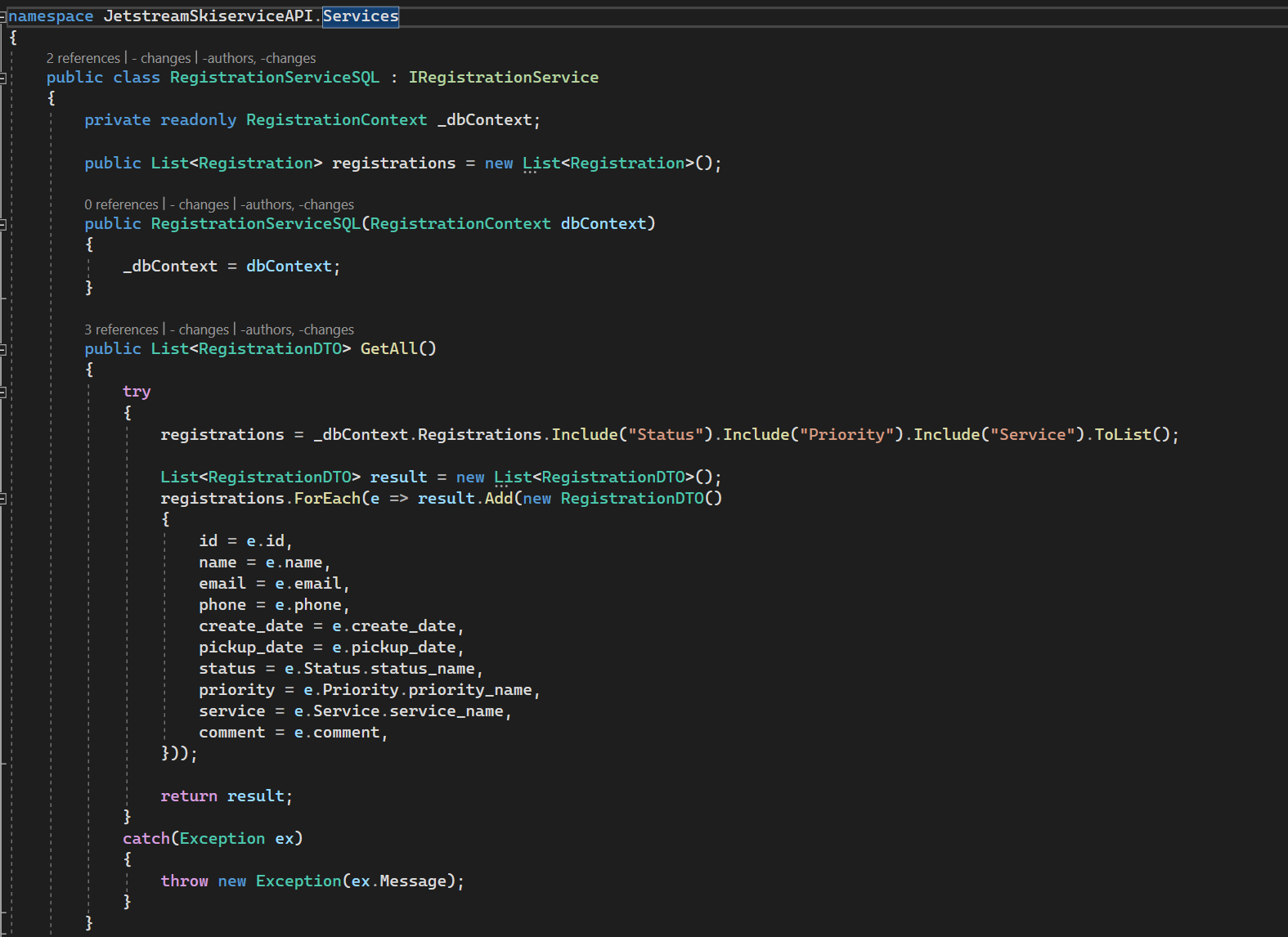
Wenn man mit dem API Daten einfügen will, muss man bei Status, Priority und Service einfach den Namen des gewünschten Status, Priority oder Service eingeben.

Wichtig ist natürlich, dass dieser Name auch mit dem Namen in der Datenbank übereinstimmt.

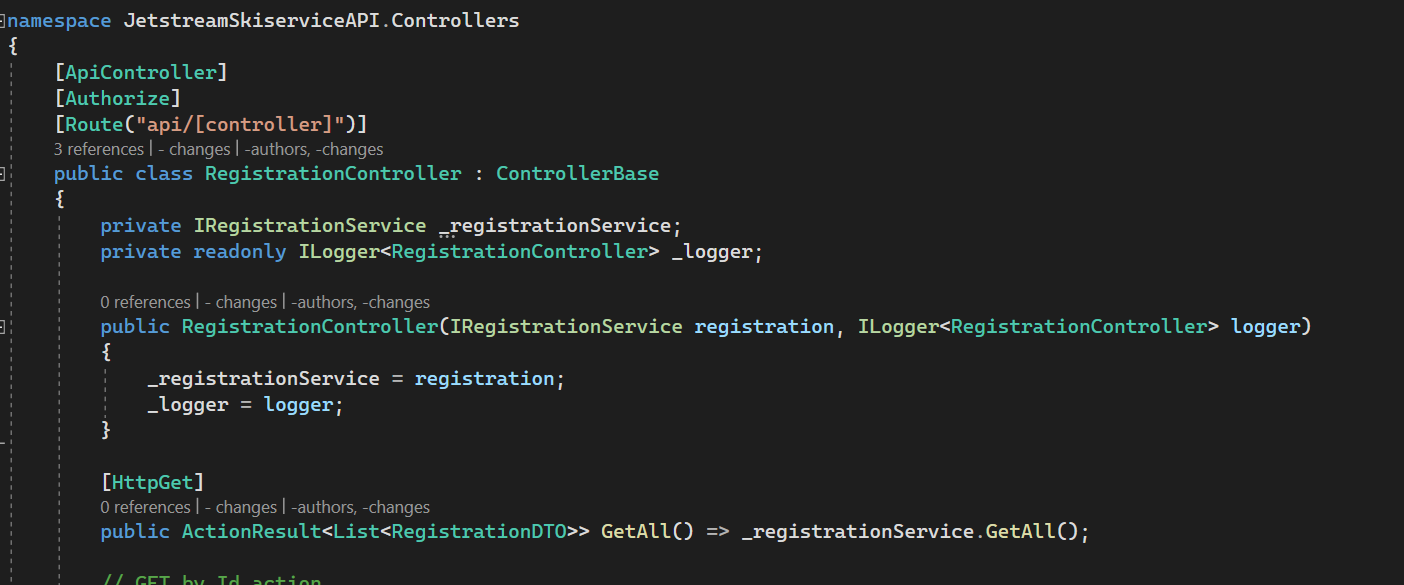
**Programm.cs** (wo das Programm startet)

****

**Service Registration** (erbt von Interface/behält Logik des Programmes)

****

**Controller Registration** (instanziiert Interface/ruft Service durch Interface auf)



### Zusatzfeatures erstellen

Um das Zugreifen von Status auf Registrationen durchzusetzen, musste ich erstmals eine neue DTO Klasse (StatusDTO) erstellen, welche status\_id, status\_name und eine Liste des Typen Registrationen enthält erstellen. Dann konnte ich durch mehrere Schleifen die Stati mit den zugehörigen Registrationen auslesen.

Zusätzlich habe ich noch eingebaut, dass man Registrationen nach Status auslesen kann, dies heisst ich kann z.B. alle Registrationen mit Status offen auslesen.

Als Nächstes habe ich auch noch ein Feature umgesetzt, welches einen User nach drei falschen Passwortangaben bannt, umgesetzt. Natürlich gibt es hier aber auch eine entsprechende Methode, um den User zu entbannen.

### Authentifikation erstellen

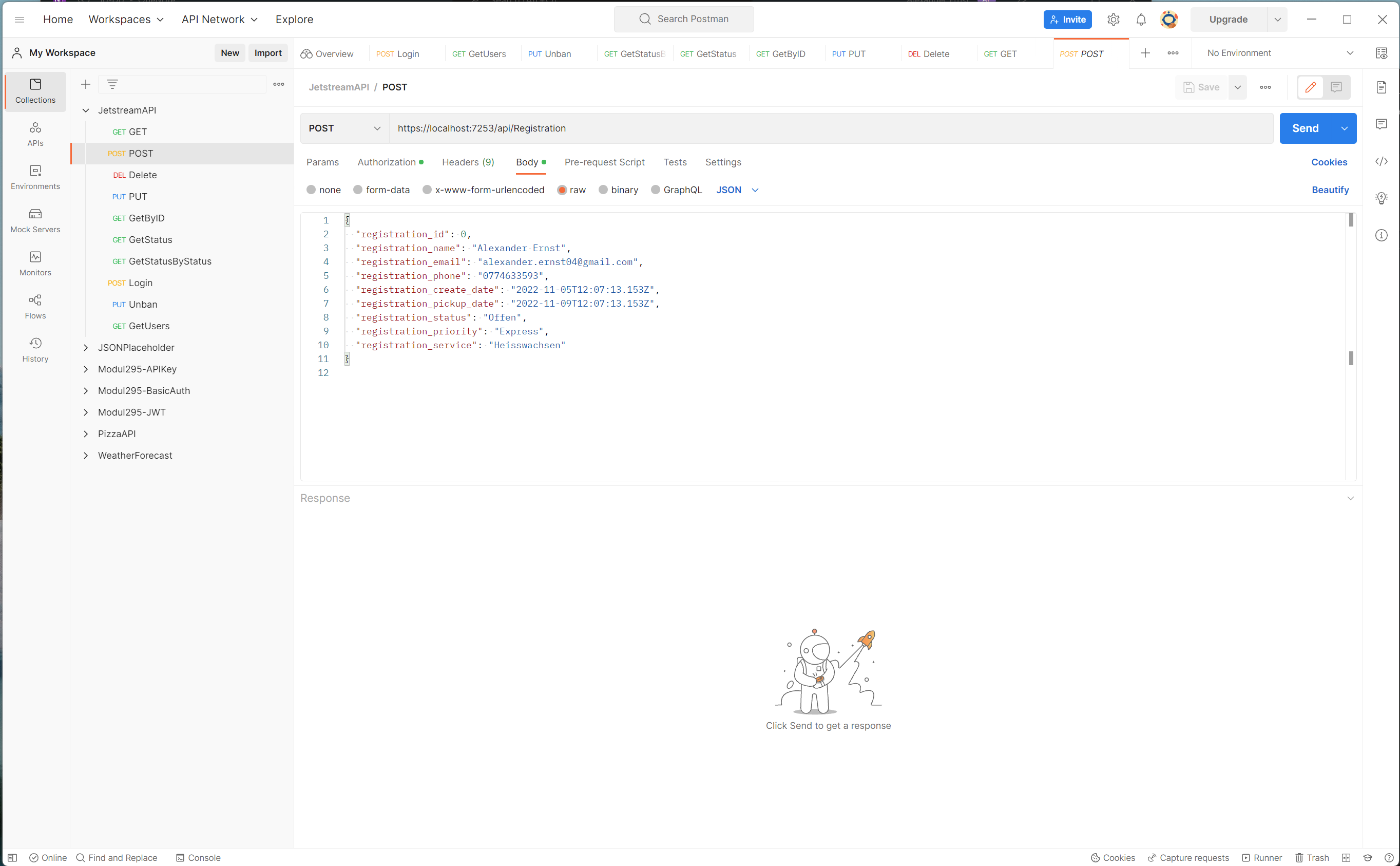
Als letzten Schritt habe ich die Authentifikation mit einem JWT Token durchgeführt, Benutzer, die ein Token kriegen, sind in einer SQL Tabelle gespeichert, die ausgelesen wird.

Für das authentifizieren und kreieren des JWT Tokens habe ich einen eigenen User Controller und einen eigenen Service mit entsprechendem Interface.

## Kontrollieren

### Web API mit Postman testen

In dieser Aktivität habe ich alle HTTP Methoden inklusive Authentifikation mit Postman getestet, ich habe auch Szenarien getestet, in denen ich eine nichtexistierende ID angeben, usw.



## Auswerten

### Reflexion/Fazit zu Projekt

Ich denke, ich habe die Hauptanforderungen dieses Projektes geschafft, da das Web API mit Authentifikation und sogar Zusatzfeatures komplett lauffähig ist.

Ich habe durch dieses Projekt auch sehr viel über APIs und C# gelernt, und konnte mein Wissen um einiges erweitern.

Was meiner Meinung in diesem Projekt nicht so gut lief, war die Zeitplanung, ich habe in meinem Zeitplan meinen Aufwand um einiges unterschätzt.

Dies liegt daran, dass man oft mehrere Stunden an einem kleinen Problem arbeitet, was dazu führt, dass man nicht mit anderem weiterkommt.

Im Allgemeinen lief dieses Projekt aber schon sehr gut und ich habe viel Neues gelernt.

#### Was habe ich nicht geschafft/Verbesserungen

Dieses Projekt lief aber auch nicht perfekt, zwei Features, die ich gerne noch geschafft hätte, wäre die Verschlüsselung von dem Connection String und einen Mapper welches den Code etwas verschönern würde.

Da diese Features, aber nicht Teil der Anforderungen waren, habe ich mich auf anderes/wichtigeres konzentriert.

### Dokumentation fertigstellen/Präsentation erstellen

Als aller letzten Arbeitsschritt habe ich die Dokumentation nochmals überarbeitet und die Präsentation der Projektarbeit erstellt.