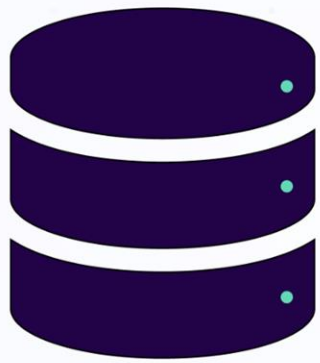


# Projektdokumentation Modul 165

## No-SQL Datenbanken einsetzen



# NoSQL

No-SQL

## Inhaltsverzeichnis

<b>Inhaltsverzeichnis</b>	<b>2</b>
<b>Informieren:</b>	<b>3</b>
<b>Planen:</b>	<b>4</b>
Anforderungen an das Projekt	4
<b>Entscheiden:</b>	<b>5</b>
Dokumentenstruktur:	5
Logik in Service-Klassen:	5
Sicherheit mit JSON Web Token (JWT):	5
<b>Realisieren:</b>	<b>6</b>
Testing:	8
<b>Kontrollieren:</b>	<b>9</b>
Klassendiagramm	9
<b>Auswerten:</b>	<b>10</b>
<b>Abbildungsverzeichnis</b>	<b>10</b>

## Informieren:

### 1 Ausgangssituation

Die Firma Jetstream-Service führt als KMU in der Wintersaison Skiservicearbeiten durch und hat in den letzten Jahren grosse Investitionen in eine durchgängige digitale Auftragsanmeldung und Verwaltung, bestehend aus einer datenbankbasierender Web-Anmeldung und Auftragsverwaltung getätigt.

Aufgrund guter Auftragslage hat sich die Geschäftsführung für eine Diversifizierung mit Neueröffnungen an verschiedenen Standorten entschieden.

Die bis anhin eingesetzte relationale Datenbank genügt den damit verbundenen Ansprüchen an Datenverteilung und Skalierung nicht mehr. Um einerseits den neuen Anforderungen gerecht zu werden sowie andererseits Lizenzkosten einzusparen, soll im Backend der Anwendung die Datenbank auf ein NoSQL Datenbanksystem migriert werden.

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.

Abbildung 2 Ausgangslage

In dieser Informationsphase wurden auch die Vor- und Nachteile verschiedener Datenbankoptionen untersucht. Aufgrund der Anforderungen an die Datenverwaltung und Flexibilität entschied ich mich für MongoDB als Datenbanklösung.

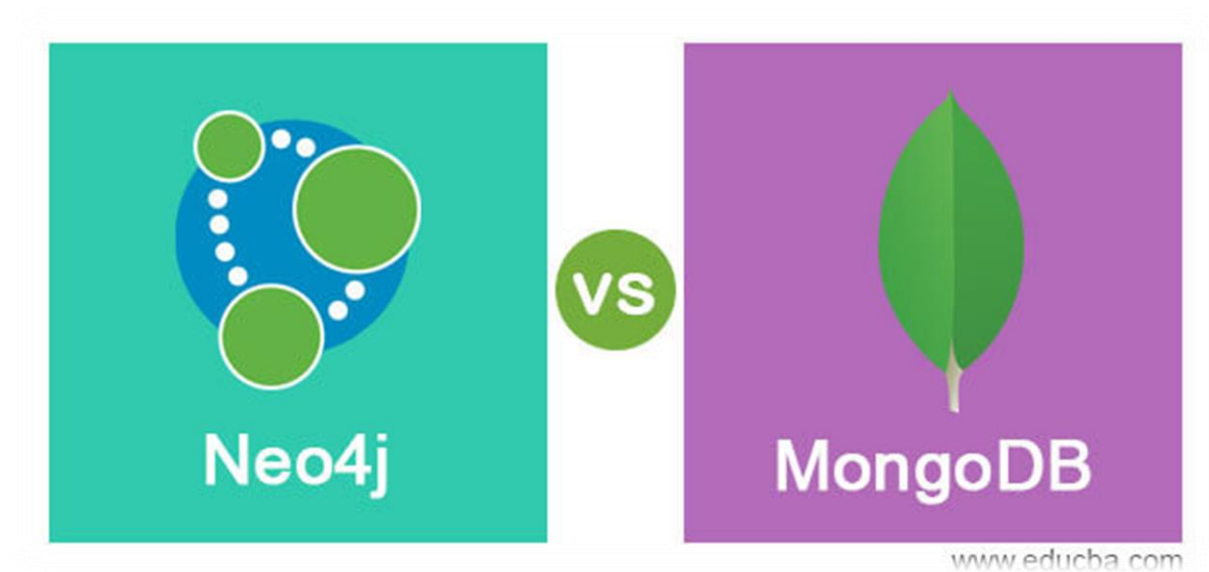


Abbildung 3 Neo4j vs MongoDB

## Planen:

### Anforderungen an das Projekt

#### 2.1 Zusammenfassung der Anforderungen

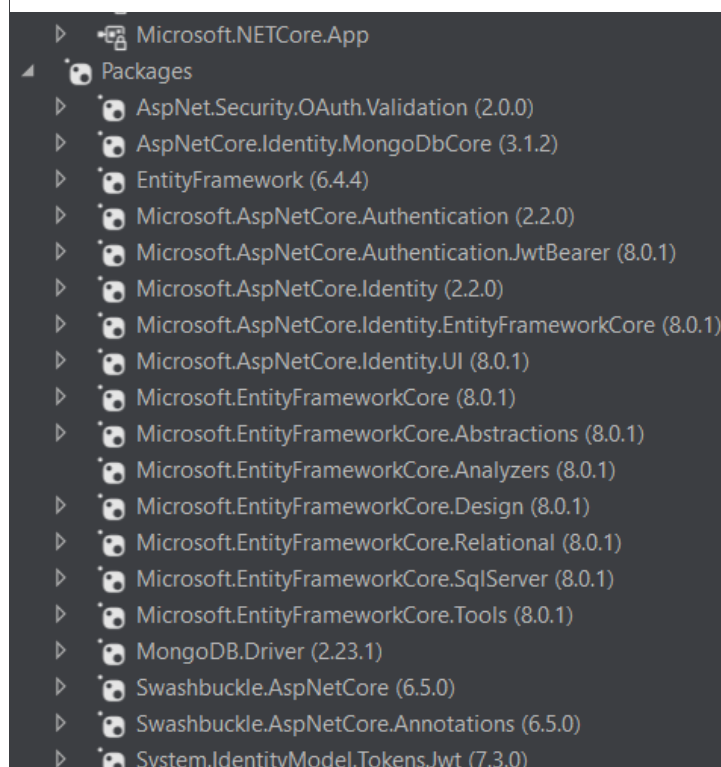
Nr.	Beschreibung
A1	Datenbasis aus relationaler Datenbank vollständig nach NoSQL migriert
A2	Benutzerkonzept mit min. 2 Benutzeranmeldungen mit verschiedenen Berechtigungsstufen implementiert.
A3	Für die Web-API Applikation muss ein eigener Datenbankbenutzerzugang mit eingeschränkter Berechtigung (DML) 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

#### 3 Zusätzliche Anforderungen

Zusatzpunkte für optionale Erweiterungen. Zur Erreichung der max. Punktzahl müssen zwei optionale Anforderungen umgesetzt werden. Es werden nur zwei zusätzliche Anforderungen bewertet.

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

Abbildung 4 Anforderungen



Verwendete NuGet Pakete, von denen aber nicht alle relevant sind

Abbildung 5 NuGet Pakete

## Entscheiden:

Dokumentenstruktur:

Ich habe mich für die einfachste Dokumentenstruktur entschieden, bei der jedes Dokument eine Bestellung repräsentiert. Dies erleichtert die Handhabung von Daten und bietet Flexibilität, da jeder Datensatz unabhängig voneinander ist.

Logik in Service-Klassen:

Um die Übersichtlichkeit und Einheitlichkeit des Codes sicherzustellen, habe ich den Großteil der Anwendungslogik in separaten Service-Klassen platziert. Dies fördert eine klare Trennung der Verantwortlichkeiten und erleichtert die Wartbarkeit der Anwendung.

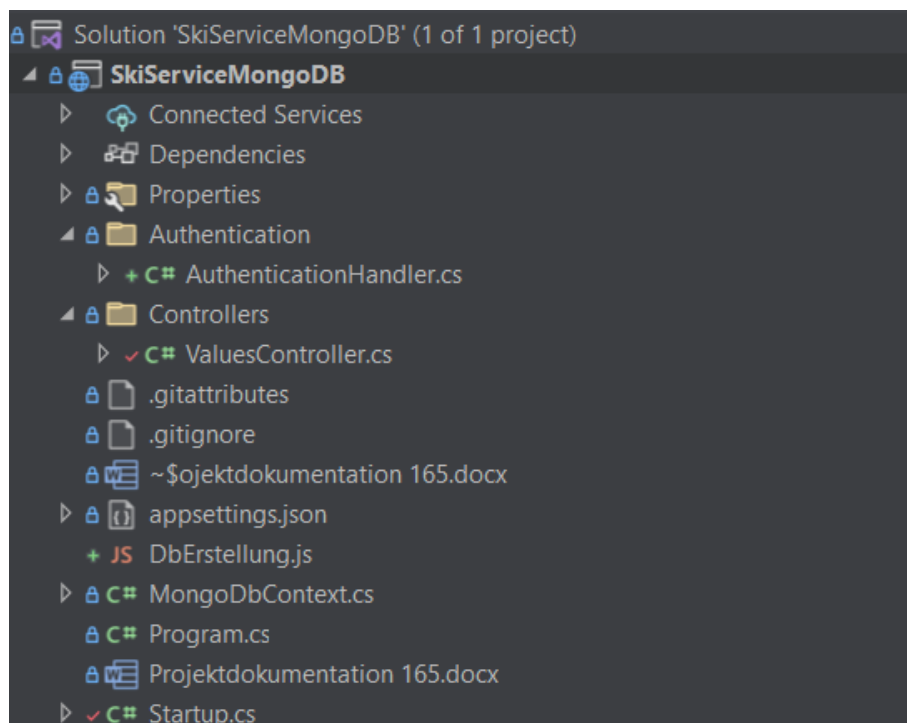


Abbildung 6 Klassenübersicht

Sicherheit mit JSON Web Token (JWT):

Für die Sicherheit habe ich auf JSON Web Token (JWT) gesetzt. JWT bietet eine effektive Möglichkeit zur Authentifizierung und Autorisierung von Benutzern. Es ermöglicht die sichere Übertragung von Informationen zwischen Parteien und wird in Verbindung mit Authentication-Middleware in meiner Anwendung verwendet.

## Realisieren:

Als erstes habe ich mir Gedanken über das Datenbankschema gemacht:

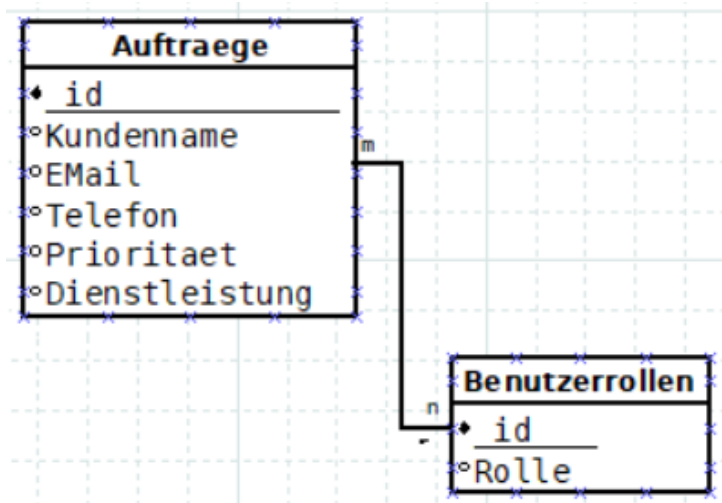


Abbildung 7 DB Schema

Als nächsten Schritt habe ich die MongoDB erstellt namens "SkiServiceDB", mit den Collections "Auftraege" und "Benutzerrollen".

Diese habe ich erstmal mit zwei vordefinierten Aufträgen befüllt.

Das auszuführende Skript findet sich unter dem Namen: "DbErstellung.js"

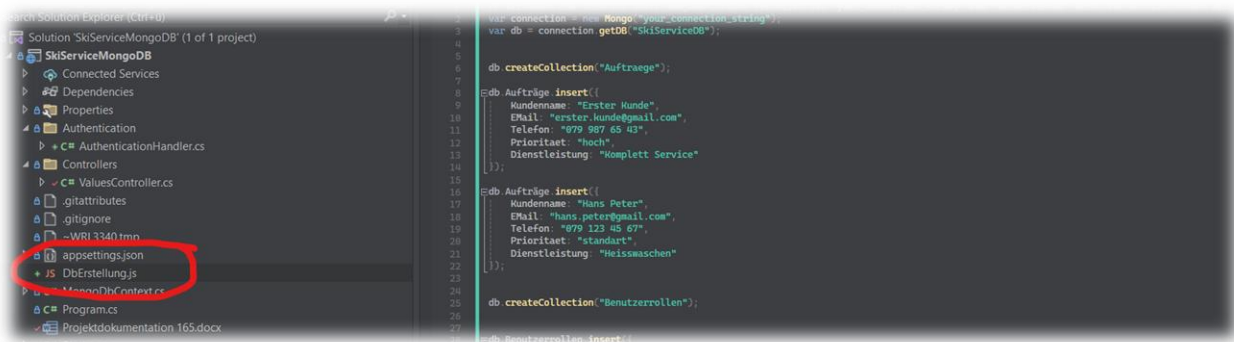


Abbildung 8 DbErstellung.js

Danach habe ich die Klasse "MongoDbContext" erstellt und in den appsettings.json dafür gesorgt dass die Datenbank mit meinem Programm verbunden werden kann.

```

1 using Microsoft.EntityFrameworkCore;
2 using MongoDB.Bson;
3 using MongoDB.Driver;
4
5 namespace SkiServiceMongoDB.Data
6 {
7     3 references
8     public class MongoDBContext
9     {
10         private readonly IMongoDatabase _database;
11
12         1 reference
13         public MongoDBContext(string connectionString, string databaseName)
14         {
15             var client = new MongoClient(connectionString);
16             _database = client.GetDatabase(databaseName);
17
18         0 references
19         public IMongoCollection<BsonDocument> Auftraege => _database.GetCollection<BsonDocument>("Auftraege");
20     }
21 }
    
```

Abbildung 9 MongoDBContext.cs

Nun habe ich mich der Erstellung der verschiedenen CRUD-Controllern gewidmet:

Get ist ohne Anmeldung möglich während die anderen Post, Put, Delete alle einen Username und ein Passwort erwarten um ausgeführt zu werden.

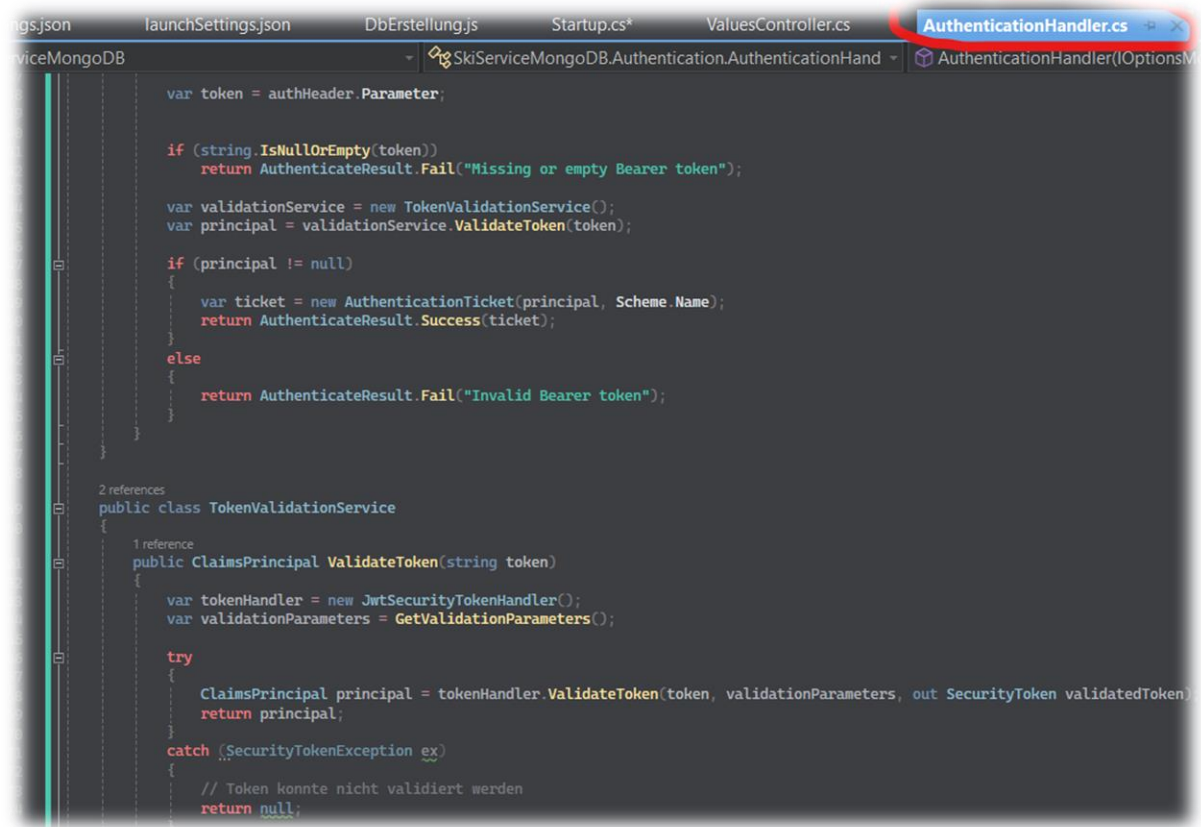
```

21
22 0 references
23 public ValuesController(IMongoClient mongoClient)
24 {
25     _mongoClient = mongoClient;
26     var database = _mongoClient.GetDatabase("SkiServiceDB");
27     _auftraegeCollection = database.GetCollection<Auftrag>("Auftraege");
28 }
29
30 // Aufträge auslesen
31 [HttpGet]
32 [AllowAnonymous]
33 public ActionResult<IEnumerable<Auftrag>> Get()
34 {
35     var auftraege = _auftraegeCollection.Find(new BsonDocument()).ToList();
36     return Ok(auftraege);
37 }
38
39 [HttpPost]
40 [SwaggerOperation(Summary = "Add a new Auftrag", Description = "Requires username and password. Returns JWT Bearer token.")]
41 public ActionResult<string> Post([FromBody] UserCredentials userCredentials)
42 {
43     // Überprüfen Sie die Benutzeranmeldedaten und generieren Sie einen JWT-Token
44     var isValid = ValidateUserCredentials(userCredentials.Username, userCredentials.Password);
45     if (isValid)
46     {
47         var token = GenerateJwtToken(userCredentials.Username);
48         return Ok(new { Token = token });
49     }
50     else
51     {
52     }
53 }
    
```

Abbildung 10 ValuesController.cs



Nachdem die CRUD-Operationen ohne Berechtigungen erstmal funktionierten via Swagger, beschäftigte ich mich mit der Authentication und dem einrichten der JWT Funktionalität.



```

ngs.json launchSettings.json DbErstellung.js Startup.cs* ValuesController.cs AuthenticationHandler.cs
ServiceMongoDB SkiServiceMongoDB.Authentication.AuthenticationHand AuthenticationHandler(IOptionsM

var token = authHeader.Parameter;

if (string.IsNullOrEmpty(token))
    return AuthenticateResult.Fail("Missing or empty Bearer token");

var validationService = new TokenValidationService();
var principal = validationService.ValidateToken(token);

if (principal != null)
{
    var ticket = new AuthenticationTicket(principal, Scheme.Name);
    return AuthenticateResult.Success(ticket);
}
else
{
    return AuthenticateResult.Fail("Invalid Bearer token");
}

2 references
public class TokenValidationService
{
    1 reference
    public ClaimsPrincipal ValidateToken(string token)
    {
        var tokenHandler = new JwtSecurityTokenHandler();
        var validationParameters = GetValidationParameters();

        try
        {
            ClaimsPrincipal principal = tokenHandler.ValidateToken(token, validationParameters, out SecurityToken validatedToken);
            return principal;
        }
        catch (SecurityTokenException ex)
        {
            // Token konnte nicht validiert werden
            return null;
        }
    }
}
    
```

Abbildung 11 Authenticationhandler.cs

### Testing:

Die Konfiguration der Controller und die Implementierung der JWT-Token-Authentifizierung stellten sich als herausfordernd, aber entscheidend für die Entwicklung meines Projekts heraus. In zahlreichen Iterationen und Tests mit Postman habe ich die Swagger-Dokumentation präzise eingestellt, um eine klare API-Interaktion zu gewährleisten.

Die JWT-Token-Authentifizierung erforderte genaue Einstellungen, um einen reibungslosen Ablauf zu gewährleisten. Durch intensive Zusammenarbeit mit meinem OpenAI-Assistenten und kontinuierliches Testen konnte ich sicherstellen, dass die Authentifizierung wie erwartet funktionierte.



## Kontrollieren:

Die Kontrolle meines Projekts war entscheidend für einen reibungslosen Entwicklungsprozess. Während der Integration von Swagger und der Anpassung der Controller haben mein Assistent und ich zusammen Herausforderungen gemeistert. Durch schnelles Testen und iterative Anpassungen konnten wir potenzielle Schwachstellen effektiv beheben.

Die Implementierung der JWT-Token-Authentifizierung erforderte genaue Einstellungen und sorgfältiges Testen. Durch den Dialog mit dem Assistenten wurde sichergestellt, dass die Authentifizierung den Sicherheitsstandards entspricht.

Regelmäßige Überprüfungen und Updates der Dokumentation ermöglichten es, den Fortschritt des Projekts im Blick zu behalten. Die enge Zusammenarbeit mit dem Assistenten trug dazu bei, Verbesserungsmöglichkeiten zu identifizieren und die Qualität des Projekts kontinuierlich zu steigern.

## Klassendiagramm

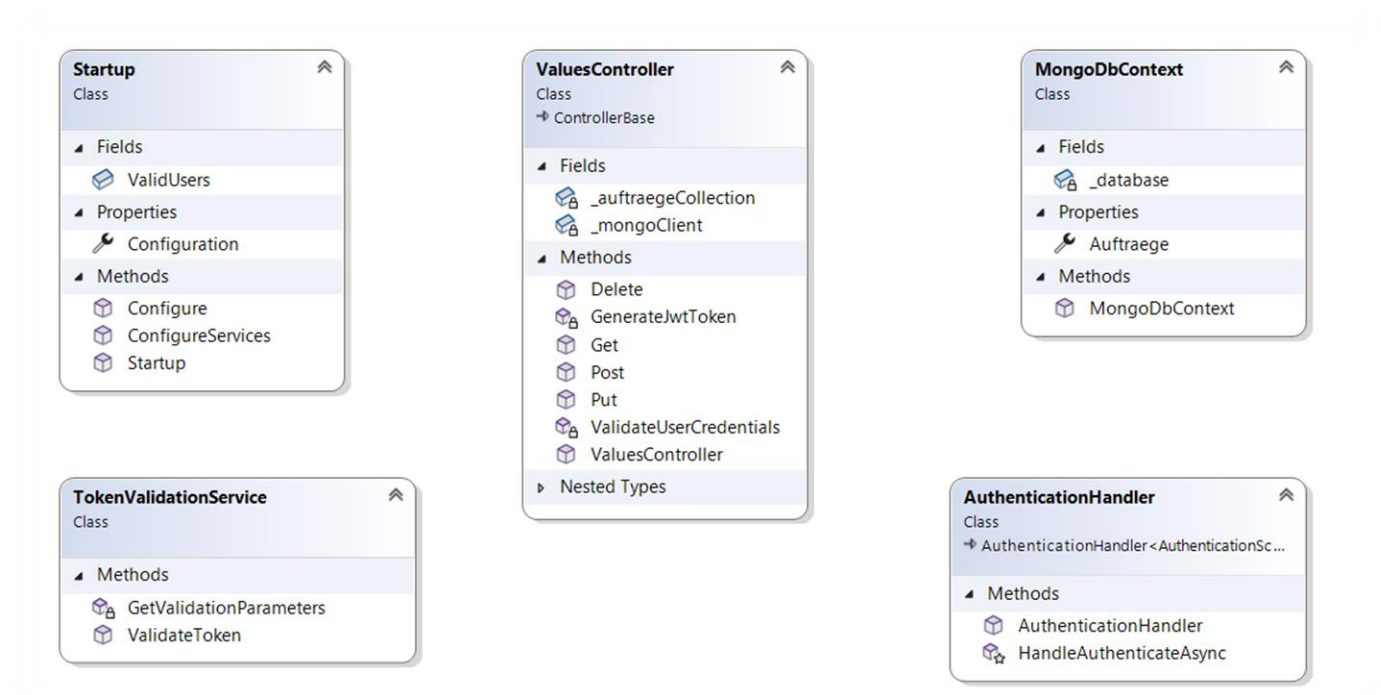


Abbildung 12 Klassendiagramm

Auswerten:

Nach Abschluss der Implementierung erfolgte eine umfassende Auswertung des gesamten Projekts. Dies beinhaltete eine Überprüfung der erreichten Meilensteine, eine Analyse der Codequalität, Leistungsbewertung sowie eine Evaluierung der Gesamtfunktionalität der entwickelten API.

Das Projekt verlief relativ gut im vergleich zu der Partnerarbeit, jedoch hatte ich durchaus Schwierigkeiten den gelernten Unterrichtsstoff in der Praxis anzuwenden.

Ich bin jedoch mit dem Abschluss des Projekts zufrieden da die Grundbasis erfüllt ist und werde mich auf jeden fall intensiver mit Datenbanken beschäftigen.

Abbildungsverzeichnis

Abbildung 1 No-SQL .....	1
Abbildung 2 Ausgangslage.....	3
Abbildung 3 Neo4j vs MongoDB.....	3
Abbildung 4 Anforderungen .....	4
Abbildung 5 NuGet Pakete .....	4
Abbildung 6 Klassenübersicht .....	5
Abbildung 7 DB Schema .....	6
Abbildung 8 DbErstellung.js.....	6
Abbildung 9 MongoDBContext.cs.....	7
Abbildung 10 ValuesController.cs.....	7
Abbildung 11 Authenticationhandler.cs .....	8
Abbildung 12 Klassendiagramm .....	9