

Projektarbeit M295

Jetstream-Service

JETSTREAM-SERVICE
RAPHAEL HUG

Inhalt

1. Informieren.....	2
1.1. Ausgangslage	2
1.2. Was ist verlangt?	2
1.3. Was muss ich mich Informieren	3
2. Planen	3
2.1. MSSQL oder MySQL.....	3
2.2. Datenbanken wie aufbauen	4
2.3. Web-API Authentication.....	5
2.4. VS-Klassendiagramm	6
2.5. Zeitplanung und PSP.....	6
3. Entscheiden	7
3.1. Welcher SQL-Server.....	7
3.2. Welcher Web-API Authentication	8
3.3. Datenbank Diagramm.....	8
4. Realisieren	9
4.1. Datenbank erstellen	9
4.2. DTO erstellt.....	10
4.3. Interface	11
4.4. Registration Controller und Service erstellt	11
4.5. Status Controller und Service erstellt.....	11
4.6. User und JWT-Token Controller und Service erstellt	11
5. Kontrollieren.....	12
5.1. Registration mit Postmann getestet	12
5.2. Status mit Postmann getestet	12
5.3. User und JWT-Token mit Postmann getestet.....	12
5.4. Checkliste.....	12
6. Auswertung	13
6.1. Was war schwierig.....	13
6.2. Was habe ich gelernt.....	13
6.3. Habe ich alle Anforderungen erfüllt?	13

1. Informieren

1.1. Ausgangslage

Wir haben ein Projekt im Modul -295 Backend-Server zu erstellen.
Diese Angaben hatten wir:

«Die Firma Jetstream-Service führt als KMU in der Wintersaison Skiservice Arbeiten durch, will im Zuge der Digitalisierung die interne Verwaltung der Ski-Service Aufträge komplett über ein Web und Datenbank basierten Anwendung abwickeln. Die bereits existierende Online-Anmeldung 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.

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, in Arbeit 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 (Angebot), siehe nachfolgende Auflistung. Pro Serviceauftrag kann immer nur eine Dienstleistung zugeordnet werden. Die Firma bietet folgende Dienstleistungen (Angebot) an: - Kleiner Service - Grosser Service - Rennski-Service - Bindung montieren und einstellen - Fell zuschneiden – Heisswachsen»

Die Abgabe ist am: 13.11.2022

1.2. Was ist verlangt?

Zusammenfassung der Anforderungen

NR.	Beischreibung
A1	Login Dialog mit Passwort für den autorisierten Zugang der Mitarbeiter (Datenänderungen).
A2	In der Datenbank müssen die Informationen des Serviceauftrags und die Login-Daten der Mitarbeiter verwaltet sein.
A3	Erfasste Mitarbeiter können eine Statusänderung eines Auftrages vornehmen. Abrufbar sein.
A4	Die erfassten Serviceaufträge müssen selektiv nach Priorität abrufbar sein.
A5	Mitarbeiter können eine Statusänderung eines Auftrages vornehmen.
A6	Mitarbeiter können Aufträge löschen (z.B. bei Stornierungen)
A7	Die aufgerufenen API-Endpoints müssen zwecks Fehlerlokalisierung protokolliert sein (DB oder Protokolldatei).
A8	Datenbankstruktur muss normalisiert in der 3.NF sein inkl. referenzieller Integrität
A9	Für die Web-API Applikation muss ein eigener Datenbankbenutzerzugang mit eingeschränkter Berechtigung (DML) zur Verfügung gestellt werden (Benutzer root bzw. sa ist verboten).
A10	Die Web-API muss vollständig nach Open-API (Swagger) dokumentiert sein.
A11	Das Softwareprojekt ist über ein Git-Repository zu verwalten.
A12	Ganzes Projektmanagement muss nach IPERKA dokumentiert sein.

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.	Beischreibung
AO1	Die Mitarbeiter können zu einem Auftrag einen Freitext bzw. Kommentar hinterlegen
AO2	Ein Auftrag kann mit sämtlichen Datenfeldern geändert werden
AO3	Das Login des Mitarbeiters wird nach drei nachfolgenden Falschanmeldungen automatisch gesperrt.
AO4	Personalisierte Auftragsliste des eingeloggtten Mitarbeiters. Der Mitarbeiter kann sich zusätzlich zur gesamten Auftragsliste nur die von ihm übernommenen Aufträge ansehen.
AO6	Eingeloggte Mitarbeiter können ein gesperrtes Login zurücksetzen.
AO7	Gelöscht Aufträge werden nicht aus der Datenbank entfernt, sondern nur als gelöscht markiert.

1.3. Was muss ich mich Informieren

Am Anfang des Moduls wusste ich nicht viel, der Dozent hat und jeden Tag mehr über das gelernt. Aber wie man es richtig macht, musste ich viel nachschauen.

Weil es ein Projekt ist, wo ich noch nie so etwas Ähnliches gemacht habe, dachte ich mir, ich muss das Problem nachschauen, wen es geschieht, einfach genauer nachschauen. Ich habe bei der planen Genauen angeben, was ich informiert habe.

2. Planen

2.1. MSSQL oder MySQL

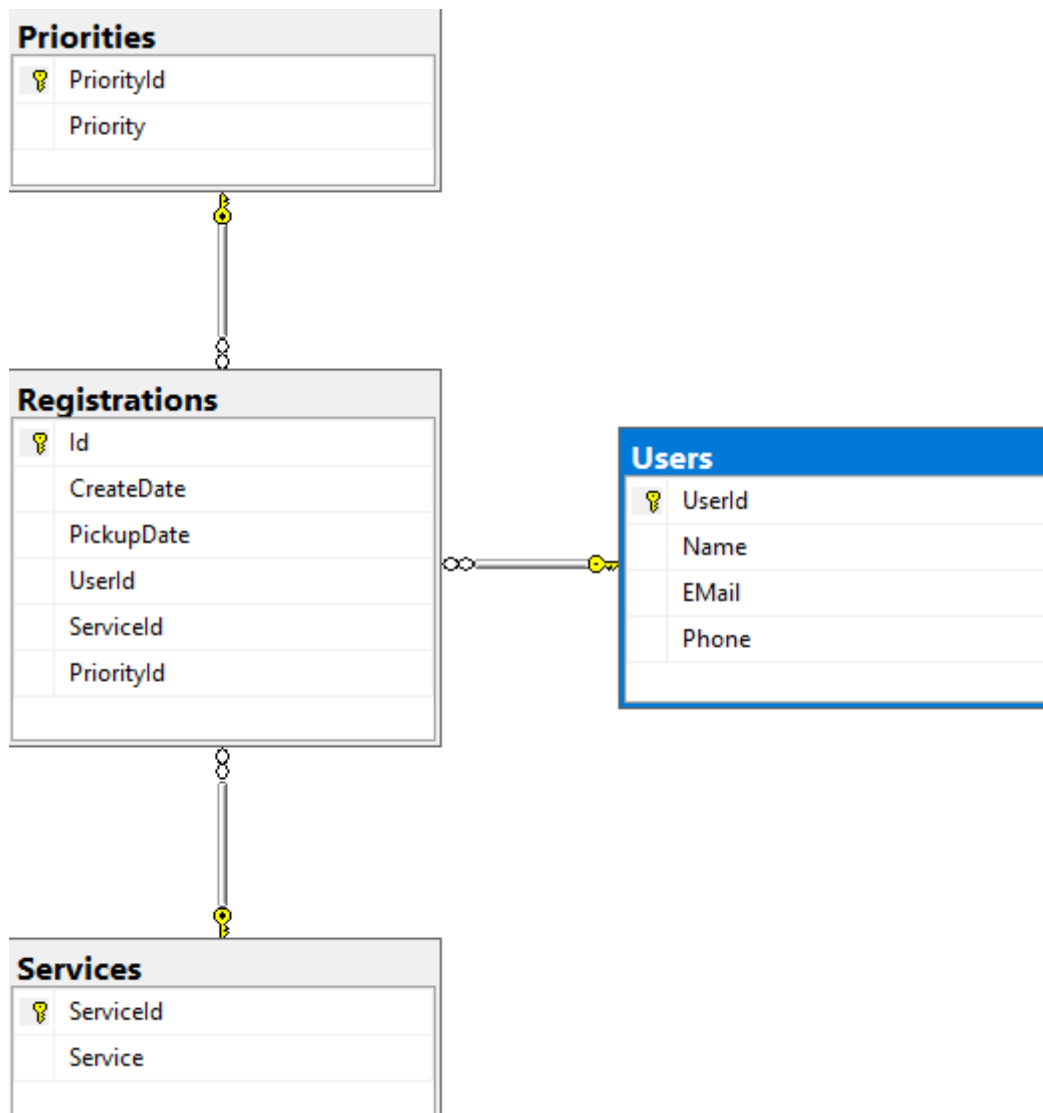
Ich habe einen MySQL-Server auf einem Raspberry und ich hatte einfach einen MSSQL Lokal auf meinem PC. Da werde ich informiere und dann entscheiden, welcher Server ich nehme.

In der Schule benutzen wir immer MSSQL und da ich ein MySQL-Server zu Hause habe, wollte ich mich genauer informieren.

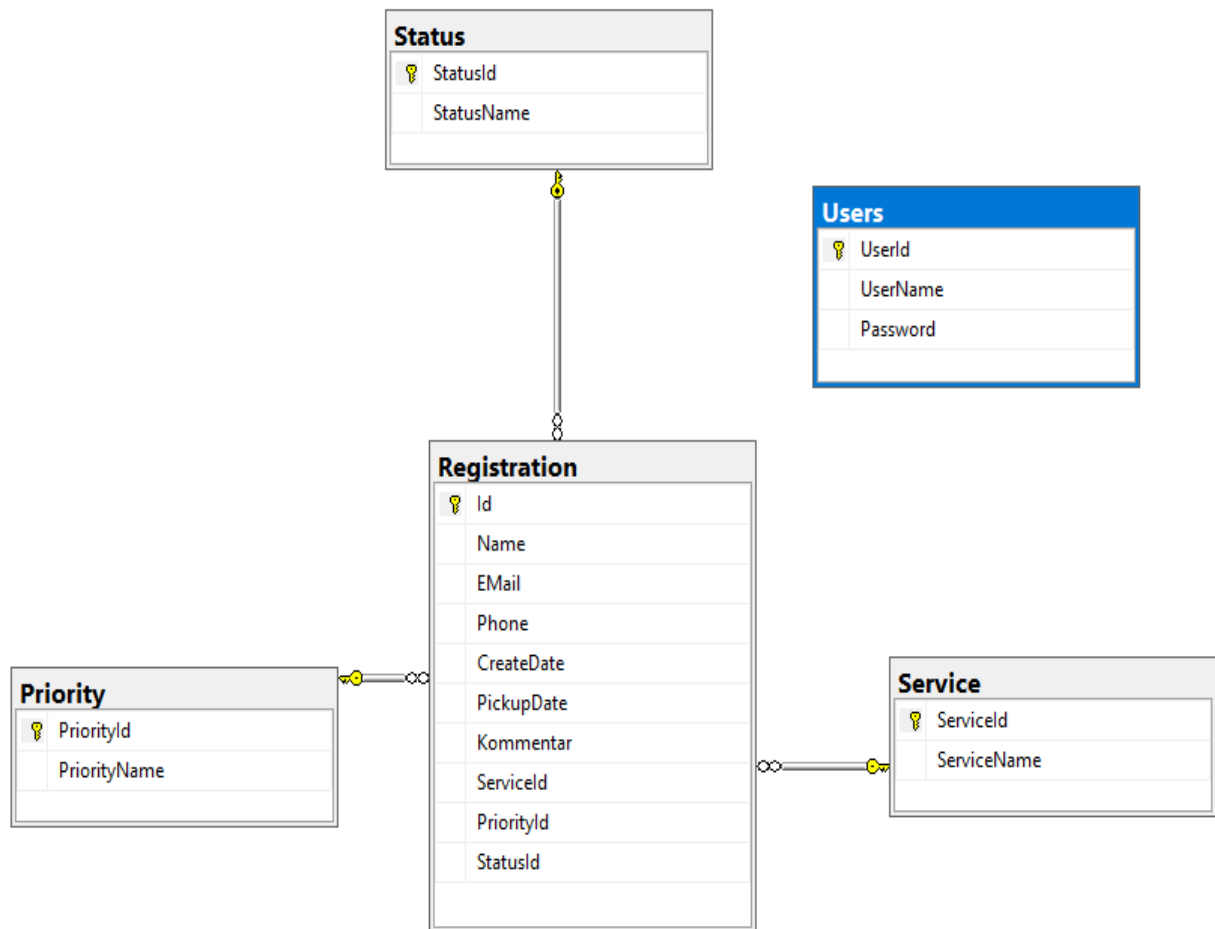
2.2. Datenbanken wie aufbauen

Also Zuerst habe ich ein Diagramm erstellt, um zu entscheiden, wie ich das ganze aufbauen will.

Bei diesem Model hat es 5 verbundene Tabellen und E-Mail, Name, Telefon sind separiert.



Bei diesem Model hat der User nur noch Username und Passwort eingefügt



Ich habe mir diese 2 Formate ausgesucht und werde später entscheiden, welche ich nehme.

2.3. Web-API Authentication

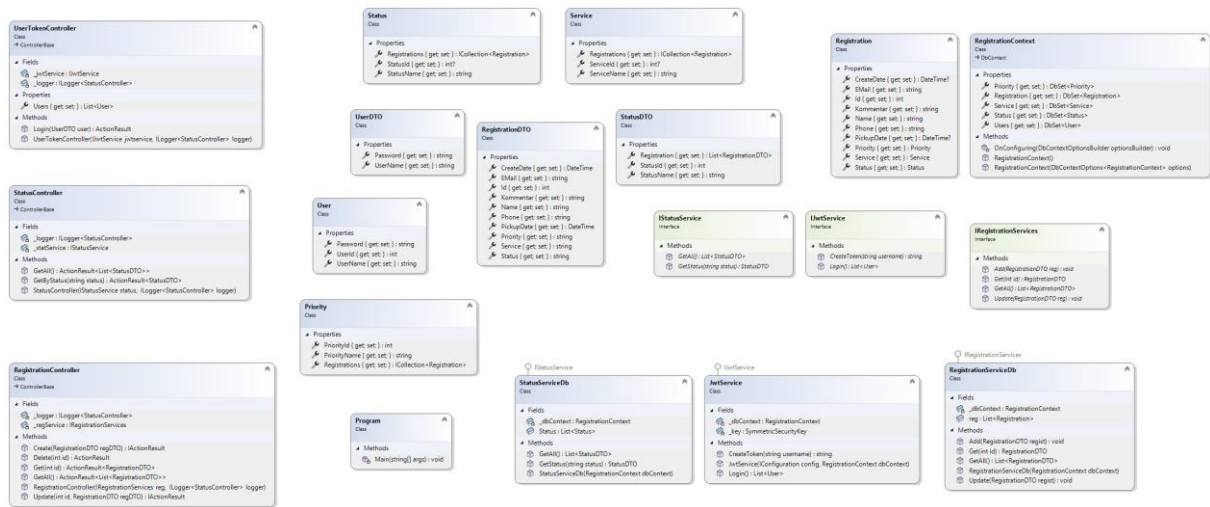
Der Dozent hat uns 3 verschiedene Modelle vorgestellt.

- JWT-Token
- API-Key
- Basic Authentication

Ich werde mir alles genaues Informieren, das ich mich kann entscheiden.

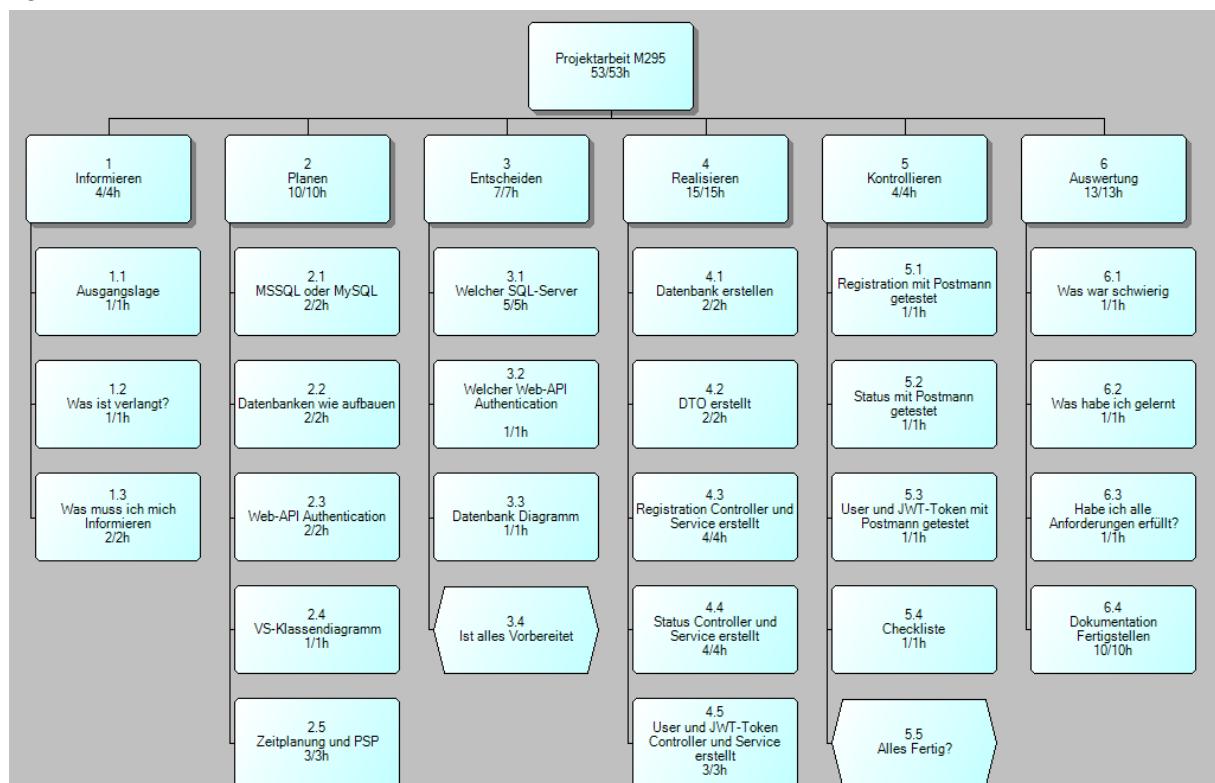
2.4. VS-Klassendiagramm

Da habe ein Fixes-Design.

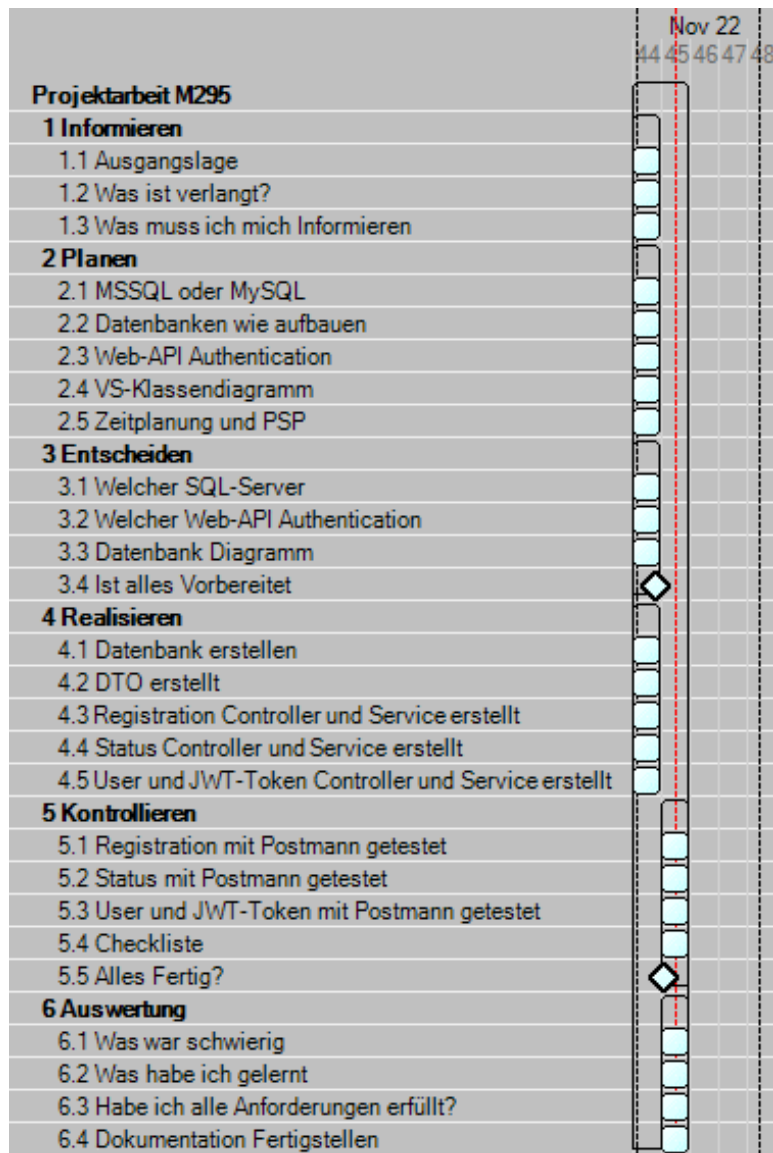


2.5. Zeitplanung und PSP

PSP:



GANT:



3. Entscheiden

3.1. Welcher SQL-Server

Ich habe mich für MSSQL entschieden, weil ich wollte, Code First machen, wir haben in der Schule dieses mich MSSQL gemacht. Ich wollte ich noch Ausprobieren in meiner einem Projekt das mit MySQL zu machen, aber bis jetzt konnte hat es noch nicht funktioniert. OR-Mapper wollte schon benutzen für Projektarbeit von der Schule.

Eigentlich wollte ich mit MySQL (MariaDB) die Tabellen erstellen und bei der Präsentation eine VPN zu mir zu Hause zu verbinden, um die Verbindung zu Datenbank zu erstellen. C# mit Visual Studio 2022 ist so Kompatible miteinander und da ich sehr viel Zeit verloren habe wegen Probieren Code First, mit MySQL musste ich fast MSSQL benutzen.

Aber ich werde, wenn ich Ferien oder genügend Freizeit haben, das auch mit MySQL schaffen. Ist halt sehr viel Recherche benötigt.

3.2. Welcher Web-API Authentication

Ich habe mich für JWT entschieden, es ist eine gute Authentication Methode und ist sehr gängig. JWT brauche ich auch eine Datenbank, aber der Schlüssel kann man die Zeit einstellen.

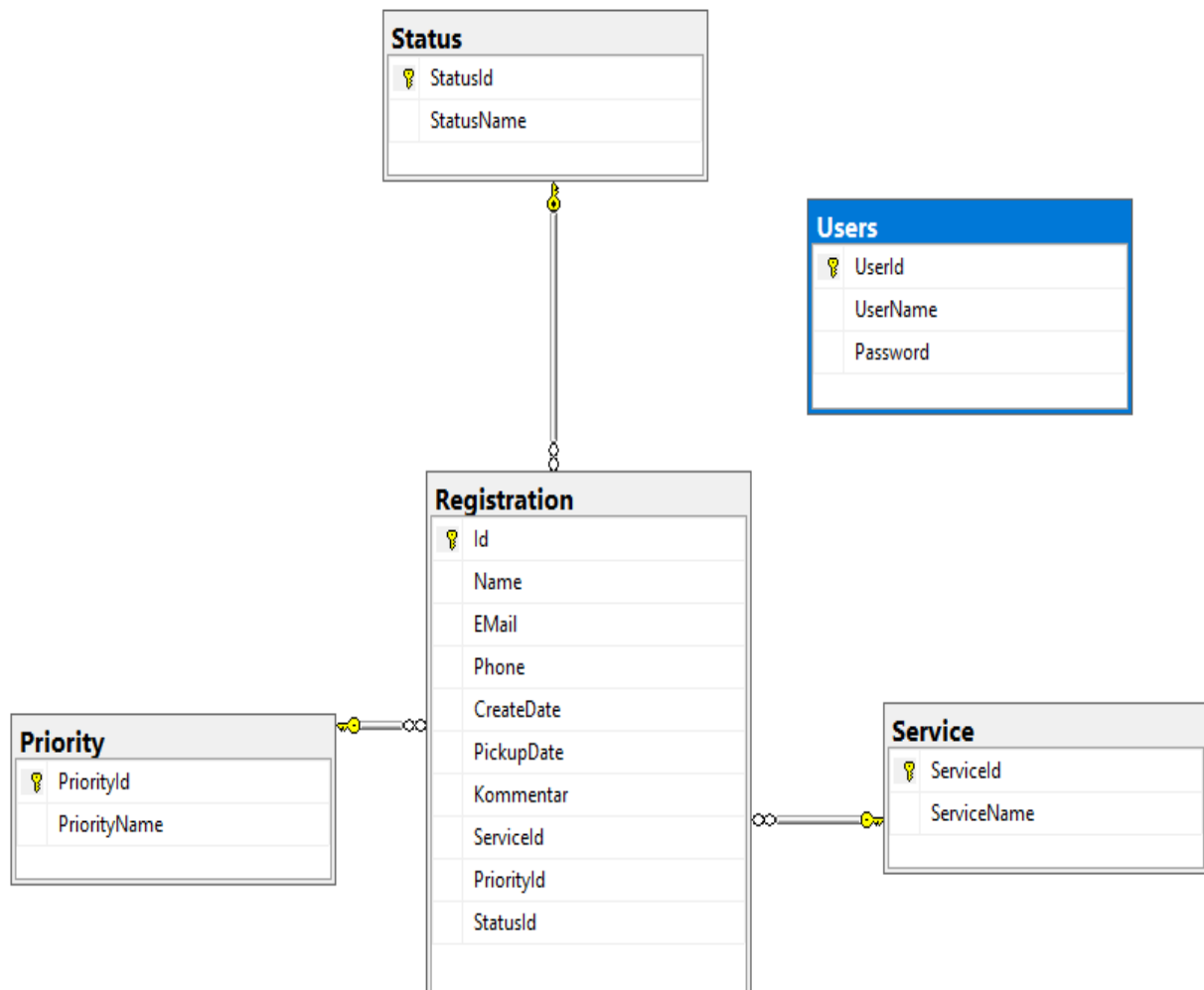
Wieso ich nicht Basic genommen habe, ist sehr leicht zu entschlüsseln und sollte nur intern in einer Firma benutzt werden.

Ich wollte eine sichere Methode programmieren.

Wieso ich nicht API-Key genommen habe, ist es ist auch eine gängige Authentication Methode, aber der Grund ist brauche eine Datenbank, wo der Key gespeichert.

3.3. Datenbank Diagramm

Ich habe dieses Model gewählt, weil es perfekt zu meinen vorherigen Entscheidungen passt.



4. Realisieren

Ich habe sehr spät mit dem Projekt angefangen, weil ich mein eigenes Projekt noch auch noch dran bin, aber habe es wegen der Schule pausiert.


4.1. Datenbank erstellen







Ich habe Code First ausgewählt und habe diese in Ordner Models eingefügt und diese programmiert, mit dem verschiedenen Klassen Registration, Status, Priority, Service und User eingebaut und den FK gemacht, wo alles in der Registration drin ist.

```

11 references | raphi, 1 day ago | 2 authors, 4 changes
public class Registration
{
    [Key]
    3 references | Raphael Hug, 6 days ago | 1 author, 1 change
    public int Id { get; set; }
    [StringLength(255)]
    4 references | raphi, 4 days ago | 1 author, 1 change
    public string? Name { get; set; }
    [StringLength(100)]
    4 references | raphi, 4 days ago | 1 author, 1 change
    public string? EMail { get; set; }
    [StringLength(10)]
    4 references | raphi, 4 days ago | 1 author, 1 change
    public string? Phone { get; set; }
    4 references | raphi, 4 days ago | 2 authors, 2 changes
    public DateTime? CreateDate { get; set; }
    4 references | raphi, 4 days ago | 2 authors, 2 changes
    public DateTime? PickupDate { get; set; }
    4 references | raphi, 4 days ago | 1 author, 1 change
    public string? Kommentar { get; set; }
    4 references | raphi, 3 days ago | 2 authors, 3 changes
    public Service? Service { get; set; }
    4 references | raphi, 3 days ago | 2 authors, 3 changes
    public Priority? Priority { get; set; }
    4 references | raphi, 3 days ago | 1 author, 2 changes
    public Status? Status { get; set; }
}

```

 Models

- ▷  Priority.cs
- ▷  Registration.cs
- ▷  RegistrationContext.cs
- ▷  Service.cs
- ▷  Status.cs
- ▷  User.cs

Danach hatte ich noch eine Klasse erstellt mit dem Namen RegistrationContext und dort habe ich die nötigen Angaben gemacht, um mit der Datenbank verbinden

```

10 references | raphi, 1 day ago | 2 authors, 3 changes
public partial class RegistrationContext : DbContext
{
    //PowerShell
    //Add-Migration InitialCreate
    //Update-Database

    0 references | raphi, 1 day ago | 2 authors, 3 changes
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlServer(@"Server=.;Database=SkiServicePA;Trusted_Connection=True");
    }

    2 references | raphi, 3 days ago | 1 author, 1 change
    public DbSet<Priority> Priority { get; set; }
    2 references | raphi, 3 days ago | 1 author, 1 change
    public DbSet<Registration> Registration { get; set; }
    2 references | raphi, 3 days ago | 1 author, 1 change
    public DbSet<Service> Service { get; set; }
    1 reference | Raphael Hug, 6 days ago | 1 author, 1 change
    public DbSet<User> Users { get; set; }
    3 references | raphi, 3 days ago | 1 author, 2 changes
    public DbSet<Status> Status { get; set; }

    0 references | Raphael Hug, 6 days ago | 1 author, 1 change
    public RegistrationContext()
    {
    }

    0 references | Raphael Hug, 6 days ago | 1 author, 1 change
    public RegistrationContext(DbContextOptions<RegistrationContext> options)
        : base(options)
    {
    }
}

```

4.2. DTO erstellt

DTO habe ich gemacht, um Fehler zu vermeiden, die Daten. Unabhängig von der Datenbank zu erstellen DTO wird gebraucht für die JSON-Datei Ich habe eins für Registration, Status und User gemacht.

```

25 references | raphi, 4 days ago | 1 author, 1 change
public class RegistrationDTO
{
    [JsonPropertyName("id")]
    7 references | raphi, 4 days ago | 1 author, 1 change
    public int Id { get; set; }

    [JsonPropertyName("name")]
    8 references | raphi, 4 days ago | 1 author, 1 change
    public string? Name { get; set; }

    [JsonPropertyName("email")]
    8 references | raphi, 4 days ago | 1 author, 1 change
    public string? EMail { get; set; }

    [JsonPropertyName("phone")]
    8 references | raphi, 4 days ago | 1 author, 1 change
    public string? Phone { get; set; }

    [JsonPropertyName("priority")]
    8 references | raphi, 4 days ago | 1 author, 1 change
    public string? Priority { get; set; }

    [JsonPropertyName("service")]
    8 references | raphi, 4 days ago | 1 author, 1 change
    public string? Service { get; set; }

    [JsonPropertyName("status")]
    8 references | raphi, 4 days ago | 1 author, 1 change
    public string? Status { get; set; }

    [JsonPropertyName("kommentar")]
    8 references | raphi, 4 days ago | 1 author, 1 change
    public string? Kommentar { get; set; }

    [JsonPropertyName("create_date")]
    8 references | raphi, 4 days ago | 1 author, 1 change
    public DateTime CreateDate { get; set; }

    [JsonPropertyName("pickup_date")]
    8 references | raphi, 4 days ago | 1 author, 1 change
    public DateTime PickupDate { get; set; }
}

```

```

11 references | raphi, 3 days ago | 1 author, 1 change
public class StatusDTO
{
    [JsonPropertyName("status_id")]
    1 reference | raphi, 3 days ago | 1 author, 1 change
    public int StatusId { get; set; }
    [JsonPropertyName("status")]
    2 references | raphi, 3 days ago | 1 author, 1 change
    public string StatusName { get; set; }
    1 reference | raphi, 3 days ago | 1 author, 1 change
    public List<RegistrationDTO> Registration { get; set; } = new List<RegistrationDTO>();
}

```

```

1 reference | raphi, 1 day ago | 1 author, 2 changes
public class UserDTO
{
    [JsonPropertyName("user_name")]
    2 references | raphi, 3 days ago | 1 author, 1 change
    public string UserName { get; set; }

    [JsonPropertyName("password")]
    1 reference | raphi, 3 days ago | 1 author, 1 change
    public string Password { get; set; }
}

```

4.3. Interface

Ich habe für alle Services ein Interface erstellt, um DI zu verwirklichen. Ich habe IRegistrationService, IStatusService und IJwtService.

4.4. Registration Controller und Service erstellt

Zuerst habe ich mal Registration Controller gemacht den werden HttpGet, HttpPost, HttpPut und HttpDelete dort habe ich den Service eingefügt über das Interface. HttpDelete habe ich einfach den Wert beim Status «Gelöscht» eingefügt.

Danach habe ich den Controller erstellt und die dort die folgende Methode eingefügt
Registration Service DB

Alle Daten werden von der Datenbank herausgelesene.

- GetAll
Habe ich einprogrammiert das alles ausliest ausser die den Status gelöscht hat.
- Get (id)
Dort habe ich einprogrammiert das alles ausliest mit der Gewählten id.
Wen man eines Gelöschten will, auslassen geht es nicht, aber man muss das beim Put den Status ändern.
- Add
Dort kann man einen Service-Antrag beantragen. Und fügt sie in der Datenbank hinzu.
- Put (id)
Dort kann man einen Service-Antrag verändern.

4.5. Status Controller und Service erstellt

Dort habe ich nur GetAll und Get (status) im Controller eingefügt.
Dort sieht man auch den gelöschten Einträgen auch
Sonst habe ich es genau gleich gemacht wie bei der Registration.

4.6. User und JWT-Token Controller und Service erstellt

Beim User habe ich einen User in der Datenbank eingefügt und die Vorlage vom JWT bekommen.

5. Kontrollieren

5.1. Registration mit Postmann getestet

Ich habe dieses mit Postmann getestet alle Registration zuerst ohne Auth

Ich konnte alles auslesen und auch per Get (id) ausgelesen wird.

Mit Post kann man einen Eintrag herstellen.

Man kann auch die Einträge verändern mit Put.

Beim Löschen wird es wie gewünscht gibt, den Status «Gelöscht» eintragen.

5.2. Status mit Postmann getestet

Beim Status kann man alle Einträge auslassen, auch die Gelöschten.

Auch man kann Status Name angeben dann gibt es nur noch die Status an wo man gewählt hat.

5.3. User und JWT-Token mit Postmann getestet

Das funktioniert gut man kann einen JWT-Token bekommen und so kann man alles machen.

5.4. Checkliste

NR.	Beischreibung	Check
A1	Login Dialog mit Passwort für den autorisierten Zugang der Mitarbeiter (Datenänderungen).	x
A2	In der Datenbank müssen die Informationen des Serviceauftrags und die Login Daten der Mitarbeiter verwaltet sein.	x
A3	Erfasste Mitarbeiter können eine Statusänderung eines Auftrages vornehmen. abrufbar sein.	x
A4	Die erfassten Serviceaufträge müssen selektiv nach Priorität abrufbar sein.	x
A5	Mitarbeiter können eine Statusänderung eines Auftrages vornehmen.	x
A6	Mitarbeiter können Aufträge löschen (z.B. bei Stornierungen)	x
A7	Die aufgerufenen API-Endpoints müssen zwecks Fehlerlokalisierung protokolliert sein (DB oder Protokolldatei).	x
A8	Datenbankstruktur muss normalisiert in der 3.NF sein inkl. referenzieller Integrität	x
A9	Für die Web-API Applikation muss ein eigener Datenbankbenutzerzugang mit eingeschränkter Berechtigung (DML) zur Verfügung gestellt werden (Benutzer root bzw. sa ist verboten).	x
A10	Das Web-API muss vollständig nach Open-API (Swagger) dokumentiert sein.	x
A11	Das Softwareprojekt ist über ein Git-Repository zu verwalten.	x
A12	Ganzes Projektmanagement muss nach IPERKA dokumentiert sein.	x
AO1	Die Mitarbeiter können zu einem Auftrag einen Freitext bzw. Kommentar hinterlegen	x
AO7	Gelöscht Aufträge werden nicht aus der Datenbank entfernt, sondern nur als gelöscht markiert.	x

6. Auswertung

6.1. Was war schwierig

Das ganze Implantieren von der Datenbank mit mehreren Tabellen war am Anfang sehr schwer. Status anzeigen zu lassen und dass auch Registration auch noch angezeigt wird, hatte ich Kopfzerbrechen.

6.2. Was habe ich gelernt

Wie man in C# eine ASP.Net API erstellt und wie man OR-Mapper richtig macht.
Was die Vorteile von DTO sind, so ist es viel leichter zu verbinden, um die Daten auszutauschen.

6.3. Habe ich alle Anforderungen erfüllt?

Ich denke habe alle Punkte verlangtem Punkte erfüllt, wen etwas nicht so stimmt, bin ich auf Feedback vom Dozenten gespannt, dass ich mich verbessern kann und mir diesen Punkt hinter den Ohren streichen kann.