

16. NOVEMBER 2023

# Entity Framework



## WEB-API | JETSTREAM

M295

ARDA DURSUN  
IPSO BILDUNG AG  
Elisabethenanlage 9

## Inhalt

<b>1</b>	<b>Einleitung</b>	<b>3</b>
1.1	Projektübersicht .....	3
1.2	Zielsetzung .....	3
<b>2</b>	<b>Informieren</b>	<b>3</b>
2.1	Anforderungsanalyse .....	3
2.2	Technologie Stack .....	4
<b>3</b>	<b>Planen</b>	<b>5</b>
3.1	PSP (Projektstrukturplan) .....	5
3.2	Frontend-Architektur .....	6
3.2.1	Installation der Nugets .....	7
3.3	API-Design und Endpunkte .....	7
3.4	Sicherheitskonzepte (JWT, Benutzerrolle) .....	8
3.4.1	JSON Web Tokens (JWT).....	8
3.4.2	Benutzerrollen.....	9
3.4.3	Sicherheitsüberlegungen für Entwicklung und Dozenten .....	9
3.4.4	Zusammenfassung .....	9
3.5	Datenmodellierung und Datenbankstruktur .....	9
3.5.1	Entitäten und ihre Struktur .....	9
3.5.2	Beziehungen zwischen den Entitäten .....	10
3.5.3	Datenbankkonfiguration und Sicherheitsaspekte .....	10
3.5.4	Zusammenfassung .....	10
<b>4</b>	<b>Entscheiden</b>	<b>11</b>
4.1	Technologieauswahl .....	11
4.1.1	Datenbanktechnologie: .....	11
4.1.2	Backend-Entwicklung: .....	11
<b>5</b>	<b>Realisieren</b>	<b>11</b>
5.1	Implementierung der Backend-Logik .....	11
5.1.1	Mitarbeiterverwaltung.....	11
5.1.2	Serviceauftragsverwaltung .....	12
5.1.3	Datenbankinteraktionen und Entity Framework .....	12
5.1.4	Sicherheit und Authentifizierung.....	12
5.1.5	Fehlerbehandlung und Logging .....	13
5.2	Entwicklung der Datenbank mit Code First-Ansatz .....	13
5.3	Einrichtung von Benutzerberechtigungen und Sicherheitsmechanismen .....	13
5.4	Integration von Logging mit Serilog.....	13
<b>6</b>	<b>Kontrollieren</b>	<b>14</b>
<b>7</b>	<b>Auswerten</b>	<b>15</b>
7.1	Analyse der erreichten Ziele und Anforderungen .....	15
7.1.1	Ausgangssituation und Ziele .....	15
7.1.2	Erreichte Ziele und Anforderungen .....	15
7.1.3	Herausforderungen und Verbesserungsmöglichkeiten .....	15
7.1.4	Fazit.....	16
7.2	Reflektion über Herausforderungen und Lösungsansätze .....	16
7.2.1	Herausforderung: Implementierung von Login und User-Management im Code First-Ansatz .....	16
7.2.2	Lösungsansatz und Unterstützung durch einen Kollegen .....	16
7.2.3	Fazit.....	16

7.3	Lessons Learned und mögliche Verbesserungen .....	17
7.3.1	Lessons Learned .....	17
7.3.2	Mögliche Verbesserungen .....	17
<b>8</b>	<b>Abschluss-Fazit + Screenshots</b>	<b>18</b>
8.1	Screenshots der Anwendung .....	18
8.2	Datenbank-Schemata .....	19
8.3	Fazit .....	19

## 1 Einleitung

### 1.1 Projektübersicht

Das Projekt "Ski Service Management" ist im Rahmen des Moduls 295 angesiedelt. Es konzentriert sich auf die Entwicklung des Backend für Applikationen. Das Herzstück des Projektes ist die Firma Jetstream-Service. Es handelt sich dabei um einen Klein- und Mittelbetrieb, der sich auf den Skiservice spezialisiert hat. Das Unternehmen möchte seine Prozesse während der Wintersaison durch die Digitalisierung der internen Verwaltung mittels einer web- und datenbankbasierten Anwendung optimieren. Das Projekt umfasst die Entwicklung einer Web-API, das Design der Datenbank, die Erstellung eines Testprojekts und die Durchführung von Tests mit Postman mit Schwerpunkt auf der Backend-Entwicklung.

### 1.2 Zielsetzung

Hauptziel des Projekts ist die Entwicklung einer leistungsfähigen und sicheren Backend-Lösung, die die Verwaltung von Skiserviceaufträgen für Jetstream Service vereinfacht. Während der Hochsaison sollen bis zu zehn Mitarbeiter über einen autorisierten und passwortgeschützten Zugang die Möglichkeit haben, Aufträge zu erfassen, zu bearbeiten und zu ändern. Es ist geplant, die bestehende Online-Registrierung, um Funktionen für das Auftragsmanagement zu erweitern. Die Anforderungen umfassen die Anzeige von Serviceaufträgen, die Änderung von bestehenden Aufträgen, die Aktualisierung des Status von Aufträgen und das Löschen von Aufträgen im Falle einer Stornierung. Die Dokumentation der Web-API nach Open API (ehemals Swagger) sowie eine umfassende Dokumentation des Gesamtprojektes nach IPERKA sind weitere wesentliche Aspekte dieses Projektes.

## 2 Informieren

### 2.1 Anforderungsanalyse

Die Anforderungsanalyse bildet die Grundlage für das Projekt "Ski-Service-Management". Die spezifischen Bedürfnisse und Erwartungen des Unternehmens Jetstream Service wurden im Rahmen dieser Phase ermittelt. Die Hauptanforderungen beinhalten

- **Entwicklung einer Web-API im Backend:**  
Die API soll die Verwaltung von Serviceaufträgen ermöglichen und Funktionen zum Erstellen, Anzeigen, Bearbeiten, Ändern des Status und Löschen von Aufträgen bereitstellen.
- **Datenbankdesign:**  
Erstellung eines Datenbankschemas zur effizienten Speicherung und Abfrage von Serviceaufträgen, Kundeninformationen und Servicearten. Die Datenbankstruktur muss normalisiert und nach der dritten Normalform (3.NF) strukturiert sein, um Redundanzen zu vermeiden und die Datenintegrität zu gewährleisten.
- **Benutzerauthentifizierung und -autorisierung:**  
Implementierung eines sicheren Authentifizierungs- und Autorisierungssystems, das es den Mitarbeitern ermöglicht, sich in die Anwendung einzuloggen und je nach Berechtigungsstufe verschiedene Aktionen durchzuführen.
- **API-Dokumentation:**  
Dokumentation der API mit Swagger/OpenAPI, um die Endpunkte, Anforderungen und möglichen Antworten der API klar darzustellen.

- **API-Test:**

Erstellen einer Testumgebung und Ausführen von Tests mit Postman, um die Funktionalität und Sicherheit der API zu gewährleisten.

Die Ermittlung der Anforderungen erfolgte auf Basis von Interviews, der Analyse der vorhandenen Prozesse und der Prüfung der technischen Möglichkeiten. Sie sind die Grundlage für die folgenden Phasen des Projekts und die Garantie für die Übereinstimmung der endgültigen Lösung mit den Bedürfnissen des Unternehmens.

## 2.2 Technologie Stack

### Entwicklungsumgebung:

- **Visual Studio Enterprise 2022:** Verwendet für die gesamte Entwicklung; bietet integrierte Unterstützung für die C#-Entwicklung und Datenbankmanagement. Aktuelle Version im Einsatz: 17.7.5.
- **Programmiersprache und Framework:**
  - **C#:** Die primäre Programmiersprache für das Backend.
  - **.NET 7.0:** Das aktuelle Framework, das für die Entwicklung der Webanwendung verwendet wird. Dies ermöglicht moderne Webentwicklung und vereinfacht die Verwendung von APIs und Datenbanken.

### Datenbankmanagement:

- **Entity Framework Core:** Ein modernes Datenbank-Framework, das einen Code-First-Ansatz ermöglicht. Es handelt sich um ein ORM (Object-Relational Mapper), dass die Datenbankentwicklung erleichtert. Version: 7.0.14.
- **Microsoft SQL Server:** Das Datenbanksystem, das für die Speicherung aller Anwendungsdaten verwendet wird.

### Authentifizierung und Sicherheit:

- **JWT (JSON Web Token):** Implementiert mit Microsoft.AspNetCore.Authentication.JwtBearer, Version 7.0.14, für sichere Authentifizierung und Autorisierung.
- **Sicherheitsschlüssel:** Verwaltet durch Microsoft.IdentityModel.Tokens, Version 7.0.3, um die Sicherheit der Token-Infrastruktur zu gewährleisten.

### API-Design und Dokumentation:

- **Swagger/OpenAPI:** Eingesetzt durch Swashbuckle.AspNetCore, Version 6.5.0, für das Design und die Dokumentation der API-Endpunkte.
- **Logging:**
  - **Serilog:** Ein umfangreiches Logging-Framework, das in dieser Anwendung für das Protokollieren von Systemereignissen verwendet wird. Es wurde konfiguriert, um Logs sowohl in Konsolenausgaben als auch in Dateien zu schreiben. Die zentralen Pakete sind Serilog, Version 3.1.1, Serilog.AspNetCore, Version 7.0.0 und Serilog.Sinks.File, Version 5.0.0.

### Frontend-Entwicklung:

- **Bootstrap:** Ein Frontend-Framework, das für das Styling der Webseiten verwendet wird. Aktuelle Version im Einsatz: 5.3.2.
- **API-Testing:**
  - **Postman:** Ein Werkzeug zum Testen von APIs, das es ermöglicht, Anfragen zu senden und Antworten der Backend-API zu überprüfen.

### 3 Planen

#### 3.1 PSP (Projektstrukturplan)

##### Informieren (**13.11.2023**)

- Projektdefinition und Scope-Abgrenzung
- Anforderungsanalyse durch Dokument
- Technologie- und Tool-Auswahl

##### Planen (**14.11.2023**)

- Entwicklung des Projektzeitplans
- Entwurf der Datenbankarchitektur
- Festlegung des API-Designs

##### Entscheiden (**14.11.2023**)

- Entscheidung über Datenbankmanagementsystem

##### Realisieren (**15.11.2023 - 19.11.2023**)

###### •15.11.2023

- Implementierung der Datenbank
- Integration von Sicherheitsfeatures wie JWT

###### •15.11.2023 - 18.11.2023

- Entwicklung der API-Endpunkte

###### •15.11.2023 - 17.11.2023

- Vorhandenes Frontend angepasst
- Datenbank in 3. NF umcodieren
- Unit + Postman Tests erstellen

###### •19.11.2023

- Letzte Anpassungen auf API-Endpoints und Datenbank Login & User

##### Kontrollieren (**20.11.2023**)

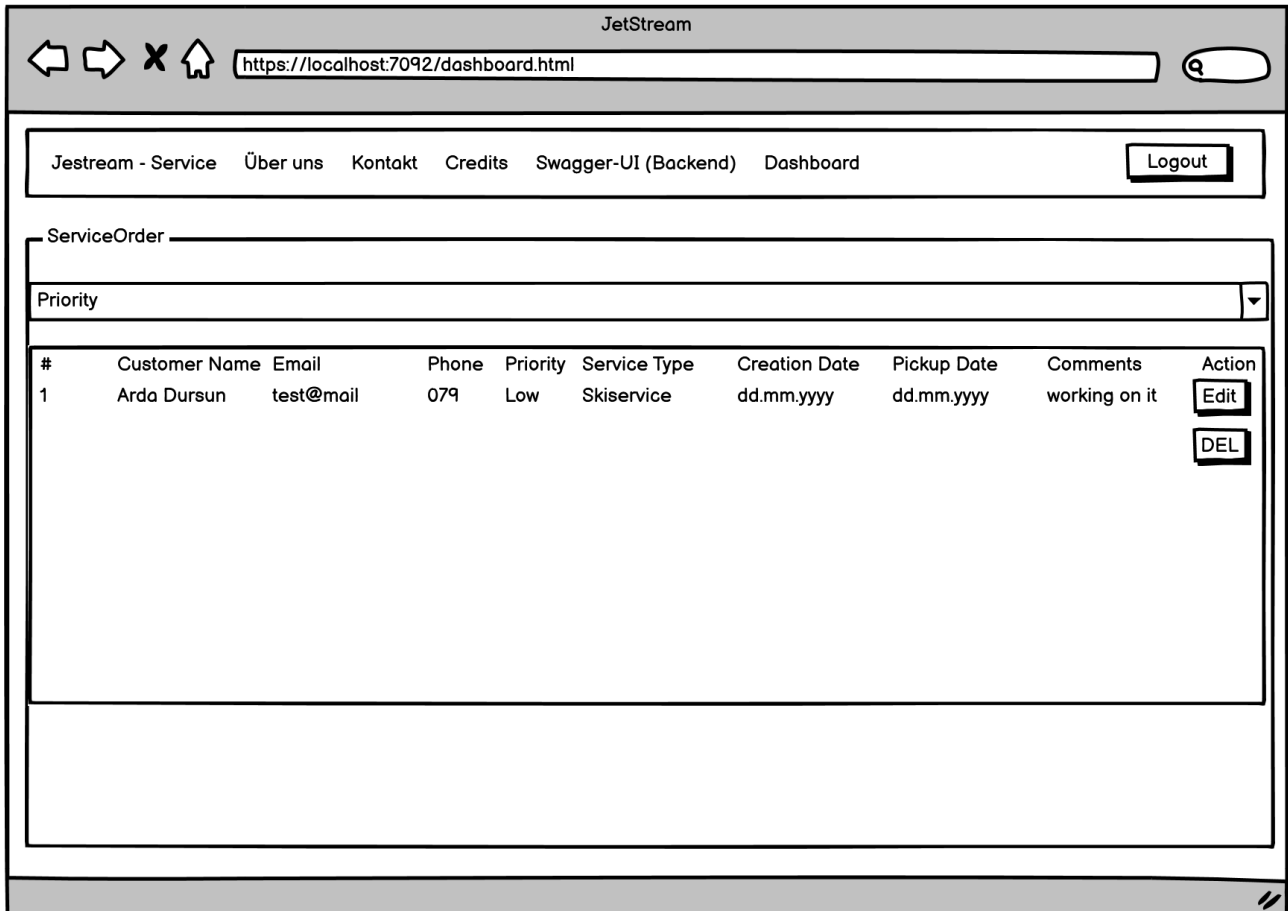
- Überprüfung des Codes und Optimierungen
- Sicherheits- und Leistungstests
- Postman (Integrität) Tests

##### Auswerten (**21.11.2023**)

- Zusammenstellung der Dokumentation
- Abgabe aller Dokumente

### 3.2 Frontend-Architektur

Die Frontend-Architektur des Dashboards "Jetstream - Service" soll den Benutzern eine klare und interaktive Schnittstelle für die Verwaltung von Serviceaufträgen zur Verfügung stellen. Das Design konzentriert sich auf Benutzerfreundlichkeit und Funktionalität. Es ist wie folgt aufgebaut:



- **Navigationsleiste (Kopfzeile):**

Die obere Leiste bietet schnellen Zugriff auf verschiedene Bereiche der Anwendung wie "Über uns", "Online-Anmeldung", "Kontakt" und "Credits". Zusätzlich ist ein direkter Link zum Backend über das "Swagger-UI" verfügbar. Dies erhöht die technische Transparenz und Zugänglichkeit für Entwickler. Die Option zum Ausloggen ist ein Hinweis auf das Vorhandensein einer Authentifizierungsfunktion in der Anwendung, was für die Sicherheit der Anwendung von Bedeutung ist.

- **Dropdown-Menü für die Filterung:**

Die Benutzer können Serviceaufträge nach Priorität filtern, was zeigt, dass das Frontend darauf ausgelegt werden sollte, mit dynamischen Datenmengen zu arbeiten und diese entsprechend den Benutzeranforderungen darzustellen.

- **Tabellarische Darstellung der Daten:**

Die Tabelle ist das zentrale Element des Dashboards. Sie dient der Darstellung der Serviceaufträge. Sie ist in mehrere Spalten unterteilt, in denen wichtige Informationen zu jedem Auftrag angezeigt werden. Dazu gehören Kundenname, Kontaktdaten, Priorität, Servicetyp, Erstellungs- und Abholdatum und Kommentare.

- **Interaktive Elemente:**

Die Struktur der Tabelle deutet auf interaktive Elemente hin, wie z.B. "Bearbeiten"- und "Löschen"-Schaltflächen, die es dem Benutzer ermöglichen, Aktionen direkt vom Dashboard aus zu initiieren, auch wenn dies in der Skizze nicht sichtbar ist.

- **Responsive Design:**

Die klar strukturierte Anordnung der Elemente suggeriert, dass das Frontend für verschiedene Endgeräte optimiert ist. So wird unabhängig von der Bildschirmgröße eine konsistente Benutzererfahrung gewährleistet.

- **Benutzererfahrung (UX):**

Das Design des Dashboards ist schlicht gehalten, mit einem weissen Hintergrund und einer einfachen Typografie, um die Lesbarkeit und Benutzerführung zu unterstützen.

- **Technologie-Stack für das Frontend:**

Es kann davon ausgegangen werden, dass Frontend-Technologien wie HTML, CSS, JavaScript und Frameworks wie Bootstrap eingesetzt werden, um eine reaktive und dynamische Benutzeroberfläche zu schaffen, ohne den Fokus zu sehr auf das Frontend zu legen.

### 3.2.1 Installation der Nugets

Um alle Benötigten Nugets runterladen zu können kann folgende Befehle in Visual Studio Package Manager Console eingeben:

```
dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer --version 7.0.14
dotnet add package Microsoft.AspNetCore.OpenApi --version 7.0.12
dotnet add package Microsoft.EntityFrameworkCore --version 7.0.14
dotnet add package Microsoft.EntityFrameworkCore.Proxies --version 7.0.14
dotnet add package Microsoft.EntityFrameworkCore.SqlServer --version 7.0.14
dotnet add package Microsoft.EntityFrameworkCore.Tools --version 7.0.14
dotnet add package Microsoft.IdentityModel.Tokens --version 7.0.3
dotnet add package Serilog --version 3.1.1
dotnet add package Serilog.AspNetCore --version 7.0.0
dotnet add package Serilog.Sinks.File --version 5.0.0
dotnet add package Swashbuckle.AspNetCore --version 6.5.0
dotnet add package System.IdentityModel.Tokens.Jwt --version 7.0.3
```

### 3.3 API-Design und Endpunkte

Im Rahmen der Planung für das JetStreamBackend-Projekt werden verschiedene API-Endpunkte entworfen, die eine umfassende Verwaltung von Serviceaufträgen und Mitarbeiterdaten ermöglichen sollen. Diese Endpunkte sind so konzipiert, dass sie eine effiziente und sichere Interaktion mit der Datenbank ermöglichen.

Spezielle Endpunkte für den Dozenten und Entwickler

In der Planungsphase wurden zwei spezifische Endpunkte berücksichtigt, die ausschliesslich für den Dozenten und Entwickler vorgesehen sind:

1. **Löschen aller Serviceaufträge (DeleteAllServiceOrders):** Dieser geplante Endpunkt soll die Möglichkeit bieten, alle Serviceaufträge aus der Datenbank zu entfernen. Dies ist besonders für das Zurücksetzen der Datenbank im Rahmen von Tests und Entwicklungen gedacht. Dieser Endpunkt wird in der Produktionsumgebung aus Sicherheitsgründen nicht verfügbar sein, um unbeabsichtigten Datenverlust zu verhindern.



2. **Erstellen eines Benutzers (CreateEmployee):** Geplant ist ebenfalls ein Endpunkt zum Anlegen neuer Mitarbeiter. Während dieser Endpunkt für das Anlegen von Benutzerkonten wichtig ist, soll der Zugriff darauf in der Produktionsumgebung eingeschränkt werden, um unbefugtes Erstellen von Benutzerkonten zu verhindern.

### **Sicherheitsaspekte in der Planung**

Für beide speziellen Endpunkte sind in der Planung entsprechende Sicherheitsmassnahmen vorgesehen, wie die Implementierung von Authentifizierungs- und Autorisierungsverfahren, um den Zugang zu diesen sensiblen Funktionen zu regeln.

### **Geplante Kernfunktionalitäten**

Zusätzlich zu den speziellen Endpunkten beinhaltet die Planung eine Reihe von Standardfunktionalitäten:

- **Verwaltung von Serviceaufträgen:** Endpunkte zum Erstellen, Bearbeiten, Löschen und Abrufen von Serviceaufträgen.
- **Mitarbeiter-Management:** Funktionen für Mitarbeiter-Login, Statusüberprüfungen und Kontenverwaltung.
- **Suchfunktionen:** Endpunkte zur Suche von Serviceaufträgen basierend auf Kundenname oder Priorität.

Abschluss der API-Planungsphase

Die Planung der API-Endpunkte bildet eine solide Grundlage für die Entwicklung des JetStreamBackend-Projekts. Die speziellen Endpunkte für Dozenten und Entwickler sind dabei als wichtige Werkzeuge für die Verwaltung und Kontrolle der Anwendung gedacht. In der finalen Umsetzung werden diese Endpunkte entsprechend angepasst oder aus der Produktionsversion entfernt, um die Sicherheit und Integrität der Anwendung zu gewährleisten.

## **3.4 Sicherheitskonzepte (JWT, Benutzerrolle)**

In der Planungsphase des JetStreamBackend-Projekts ist das Sicherheitskonzept zentral auf die Rolle der Mitarbeiter ausgerichtet. Zwei Hauptkomponenten dieses Sicherheitskonzepts sind die Verwendung von JSON Web Tokens (JWT) für die Authentifizierung und eine rollenbasierte Zugriffskontrolle, die sich auf die Mitarbeiterrolle konzentriert.

### **3.4.1 JSON Web Tokens (JWT)**

JWTs spielen eine Schlüsselrolle in der Authentifizierung und Sicherheit:

1. **Authentifizierung:** Mitarbeiter erhalten nach erfolgreicher Anmeldung einen JWT, der für nachfolgende Anfragen zur Authentifizierung verwendet wird.
2. **Token-Sicherheit:** JWTs werden digital signiert und optional verschlüsselt, um die Sicherheit der übertragenen Informationen zu gewährleisten.
3. **Token-Verwaltung:** Die Ausgabe, das Ablaufen und die Überprüfung der Tokens werden sorgfältig verwaltet, um unbefugten Zugriff zu verhindern.

### 3.4.2 Benutzerrollen

Da das Projekt ausschliesslich die Rolle "Mitarbeiter" vorsieht, ist das rollenbasierte Zugriffskontrollsystem speziell auf diese Rolle ausgerichtet:

1. **Einheitliche Rolle – "Mitarbeiter"**: Jeder Benutzer des Systems wird als Mitarbeiter behandelt, mit Zugriff auf Funktionen, die für diese Rolle relevant sind.
2. **Zugriffsbeschränkungen**: Gewisse Funktionen, wie z.B. das Löschen von Serviceaufträgen, sind möglicherweise nur bestimmten Mitarbeitern vorbehalten oder werden in der finalen Version aus Sicherheitsgründen eingeschränkt.
3. **Verwaltung der Mitarbeiterrechte**: Ein System zur Verwaltung der Zugriffsrechte der Mitarbeiter wird eingerichtet, um sicherzustellen, dass jeder Mitarbeiter nur auf die für seine Aufgaben notwendigen Funktionen Zugriff hat.

### 3.4.3 Sicherheitsüberlegungen für Entwicklung und Dozenten

Spezielle Zugriffe für Entwickler und Dozenten, wie das Löschen aller Serviceaufträge oder das Anlegen neuer Mitarbeiter, werden sorgfältig kontrolliert und sind nur im Entwicklungskontext vorgesehen. Diese Funktionen werden in der Produktionsumgebung deaktiviert oder eingeschränkt, um Sicherheit und Datenintegrität zu gewährleisten.

### 3.4.4 Zusammenfassung

Das Sicherheitskonzept des JetStreamBackend-Projekts stellt die Authentifizierung und Autorisierung der Mitarbeiter in den Mittelpunkt und nutzt JWT für eine sichere und effiziente Authentifizierung. Die Rolle des Mitarbeiters wird dabei als zentraler Faktor in der Zugriffskontrolle betrachtet, wobei spezielle Sicherheitsmassnahmen für Entwicklungs- und Kontrollzwecke berücksichtigt werden. In der Entwicklungsphase werden diese Konzepte präzisiert und implementiert, um eine sichere und zuverlässige Backend-Lösung für das Management von Serviceaufträgen zu schaffen.

## 3.5 Datenmodellierung und Datenbankstruktur

Die Datenmodellierung für das JetStreamBackend-Projekt umfasst die Entitäten **Employee**, **ServiceOrder** und **ServiceType**. Die Implementierung erfolgt in Microsoft Management Studio, wobei die Sicherheit, insbesondere im Hinblick auf die Speicherung von Passwörtern, beachtet wird.

### 3.5.1 Entitäten und ihre Struktur

1. **Employee**:
  - **Id** (int, Primary Key): Eindeutige ID des Mitarbeiters.
  - **Username** (string): Benutzername des Mitarbeiters, beschränkt auf 100 Zeichen.
  - **Password** (string): Passwort des Mitarbeiters (unverschlüsselt gespeichert).
  - **IsLocked** (bool): Status, der anzeigt, ob der Mitarbeiteraccount gesperrt ist.
  - **FailedLoginAttempts** (int): Anzahl der fehlgeschlagenen Anmeldeversuche.

## 2. **ServiceOrder:**

- **Id** (int, Primary Key): Eindeutige ID des Serviceauftrags.
- **CustomerName** (string): Name des Kunden, beschränkt auf 100 Zeichen.
- **Email** (string): E-Mail-Adresse des Kunden.
- **PhoneNumber** (string): Telefonnummer des Kunden.
- **Priority** (string): Priorität des Serviceauftrags.
- **ServiceTypeId** (int, Foreign Key): Verweis auf **ServiceType**.
- **CreationDate** (DateTime): Datum der Auftragserstellung.
- **PickupDate** (DateTime): Voraussichtliches Datum der Auftragsabholung.
- **Comments** (string): Kommentare zum Auftrag.
- **Status** (string): Status des Auftrags.

## 3. **ServiceType:**

- **Id** (int, Primary Key): Eindeutige ID des Dienstleistungstyps.
- **Name** (string): Name des Dienstleistungstyps, beschränkt auf 100 Zeichen.
- **Cost** (decimal): Kosten der Dienstleistung.

### 3.5.2 **Beziehungen zwischen den Entitäten**

- Jeder **ServiceOrder** ist mit einem **ServiceType** verbunden, wobei jeder Auftrag genau einen Typ referenziert (1:n-Beziehung).

### 3.5.3 **Datenbankkonfiguration und Sicherheitsaspekte**

- Die Datenbankstruktur wird mit Entity Framework Core in einer SQL-Datenbank umgesetzt.
- Primärschlüssel werden für jede Entität automatisch generiert.
- Seed-Daten für **Employee** und **ServiceType** werden bereitgestellt.
- **Wichtiger Hinweis:** Das Passwort wird in der **Employee**-Entität unverschlüsselt gespeichert. Es wird empfohlen, diese Entscheidung zu überdenken und Passwörter sicher zu speichern, z.B. durch Hashing, um Sicherheitsrisiken zu minimieren.

### 3.5.4 **Zusammenfassung**

Die geplante Datenmodellierung und -struktur bildet das Fundament für das JetStreamBackend-Projekt und ist auf Effizienz und Funktionalität ausgerichtet. Während die aktuelle Planung das Passwort unverschlüsselt speichert, wird eine Überprüfung dieser Entscheidung im Hinblick auf die Sicherheitsbest Practices dringend empfohlen. Die Struktur ermöglicht die effektive Verwaltung von Mitarbeiter- und Serviceauftragsdaten und ist so konzipiert, dass sie zukünftige Erweiterungen unterstützen kann.

## 4 Entscheiden

### 4.1 Technologieauswahl

#### 4.1.1 Datenbanktechnologie:

Microsoft SQL Server und Management Studio

- **Begründung:** Microsoft SQL Server bietet robuste Funktionen für das Transaktionsmanagement, die Sicherheit und die Performance-Optimierung. SQL Server Management Studio (SSMS) ermöglicht eine effiziente Verwaltung der Datenbank, inklusive der Entwicklung von Datenbankschemata, der Durchführung von Abfragen und der Verwaltung von Sicherheitsrichtlinien.
- **Vorteile:** Hohe Zuverlässigkeit, gute Skalierbarkeit und starke Unterstützung durch Microsoft. SSMS bietet eine benutzerfreundliche Oberfläche für die Verwaltung der Datenbank.

#### 4.1.2 Backend-Entwicklung:

C# mit ASP.NET Core

- **Begründung:** Die Wahl von C# in Kombination mit ASP.NET Core für das Backend basiert auf der starken Performance, der Sicherheit und der Vielseitigkeit der Sprache und des Frameworks. ASP.NET Core unterstützt RESTful API-Entwicklung, was eine saubere Trennung zwischen Backend und Frontend ermöglicht.
- **Vorteile:** C# ist eine moderne, objektorientierte Sprache, die eine klare Strukturierung des Codes ermöglicht. ASP.NET Core ist für seine Effizienz und sein leichtgewichtiges Design bekannt und unterstützt die Entwicklung von hochperformanten Anwendungen.
- 

API-Kommunikation: REST

- **Begründung:** REST (Representational State Transfer) wird für die API-Kommunikation ausgewählt, da es sich um einen weit verbreiteten, standardisierten Ansatz handelt, der leichtgewichtig, verständlich und flexibel ist. REST APIs sind in der Regel einfach zu entwickeln und zu konsumieren, was die Integration mit verschiedenen Frontend-Technologien erleichtert.
- **Vorteile:** Hohe Kompatibilität mit verschiedenen Technologien, leicht zu testen und zu warten.

## 5 Realisieren

### 5.1 Implementierung der Backend-Logik

#### 5.1.1 Mitarbeiterverwaltung

- **Erstellung eines neuen Mitarbeiters:**
  - Der **EmployeesController** stellt einen POST-Endpunkt zur Verfügung, um neue Mitarbeiter zu erstellen (**CreateEmployee**).
  - Überprüfung, ob der Benutzername bereits existiert, um Duplikate zu vermeiden.
  - Erfolgreiches Hinzufügen neuer Mitarbeiter zum System und Speicherung in der **Employees**-Tabelle.

- **Benutzeranmeldung:**
  - Implementierung eines Anmeldeprozesses (**Login**) mit Überprüfung von Benutzername und Passwort.
  - Bei erfolgreicher Authentifizierung wird ein JWT-Token generiert und zurückgegeben.
  - Behandlung von Fehlversuchen und automatische Sperrung des Kontos bei wiederholten Fehlversuchen.
- **Benutzerentsperrung:**
  - Bereitstellung eines Endpunkts zur Entsperrung gesperrter Benutzer (**UnlockEmployee**), einschliesslich entsprechender Validierung und Fehlerbehandlung.

### 5.1.2 Serviceauftragsverwaltung

- **Erstellung von Serviceaufträgen:**
  - Der **RegistrationsController** ermöglicht das Erstellen neuer Serviceaufträge (**CreateServiceOrder**).
  - Erfassung wichtiger Informationen wie Kundenname, Kontaktinformationen, Dienstleistungstyp und Priorität.
  - Berechnung des Abholdatums basierend auf der Priorität.
- **Abrufen von Serviceaufträgen:**
  - Implementierung eines GET-Endpunkts, um alle Serviceaufträge abzurufen (**GetAllServiceOrders**).
  - Optional kann nach Priorität gefiltert werden (**GetAllServiceOrders** mit Query-Parameter **priority**).
- **Suche nach Serviceaufträgen:**
  - Bereitstellung einer Suchfunktion, um Serviceaufträge anhand des Kundennamens zu finden (**SearchServiceOrdersByName**).
- **Aktualisierung und Löschung von Serviceaufträgen:**
  - Aktualisierung spezifischer Auftragsdetails, wie z.B. Kommentare (**UpdateServiceOrderComment**).
  - Möglichkeit, einzelne Serviceaufträge zu löschen (**DeleteServiceOrder**) sowie alle Serviceaufträge auf einmal (**DeleteAllServiceOrders**).

### 5.1.3 Datenbankinteraktionen und Entity Framework

- **Entity Framework Core:**
  - Verwendung von Entity Framework Core für die Interaktion mit Microsoft SQL Server.
  - Definition von Datenmodellen (**Employee**, **ServiceOrder**, **ServiceType**) und deren Beziehungen.
  - Einsatz von Migrations für die Datenbankschema-Erstellung und -Aktualisierung.

### 5.1.4 Sicherheit und Authentifizierung

- **JWT-basierte Authentifizierung:**
  - Implementierung eines sicheren Systems zur Token-Generierung und -Validierung.
  - Schutz sensibler Endpunkte durch die **[Authorize]**-Annotation.

### 5.1.5 Fehlerbehandlung und Logging

- **Robuste Fehlerbehandlung:**
  - Umfassende Fehlerbehandlung in den Controllern, um die Systemstabilität zu gewährleisten.
  - Spezifische HTTP-Antwortcodes für verschiedene Fehlerzustände.
- **Logging:**
  - Implementierung eines Logging-Mechanismus, um wichtige Systemereignisse und Fehler zu protokollieren.

## 5.2 Entwicklung der Datenbank mit Code First-Ansatz

- **Beschreibung:**
  - Der Code First-Ansatz von Entity Framework Core wird verwendet, um die Datenbankstruktur basierend auf den definierten .NET-Modellklassen (**Employee**, **ServiceOrder**, **ServiceType**) zu erstellen.
- **Vorteile:**
  - Erleichtert die Änderungen am Datenmodell und deren Synchronisierung mit der Datenbank.
  - Fördert eine agile Entwicklungsumgebung, da Anpassungen schnell umgesetzt werden können.
- **Implementierung:**
  - Definition der Modelle und ihrer Beziehungen im Code.
  - Nutzung von Migrations, um Änderungen in der Datenbankstruktur zu verwalten und anzuwenden.
- **Datenintegrität und Validierung:**
  - Implementierung von Datenvalidierungen direkt in den Modellklassen durch Data Annotations und Fluent API.

## 5.3 Einrichtung von Benutzerberechtigungen und Sicherheitsmechanismen

- **Benutzerrollen und Zugriffskontrolle:**
  - Festlegung der verschiedenen Benutzerrollen (falls erforderlich) und Implementierung der Zugriffskontrolle auf die API-Endpunkte.
- **Authentifizierung und Autorisierung:**
  - Verwendung von JWTs für die Authentifizierung und Autorisierung.
  - Sicherstellung, dass nur autorisierte Benutzer auf geschützte Ressourcen zugreifen können.
- **Sicherheitsbest Practices:**
  - Einhaltung von Sicherheitsstandards wie der Verwendung sicherer Passwörter und der sicheren Handhabung von Benutzerdaten.

## 5.4 Integration von Logging mit Serilog

- **Einsatz von Serilog:**
  - Integration von Serilog für eine verbesserte Logging-Funktionalität.
- **Funktionalitäten:**
  - Detailliertes Logging von Systemereignissen, Fehlern und Warnungen.
  - Unterstützung verschiedener Ausgabeziele (z.B. Konsole, Datei, Datenbank).
- **Konfiguration:**
  - Einrichtung und Konfiguration von Serilog im Projekt.
  - Definition der Log-Level und der zu loggenden Informationen.

## 6 Kontrollieren

### Testprotokoll - API-Endpoints-Test Postman

TESTTITEL	PRIORITÄT	TESTFALL-ID	TESTNUMMER	TESTERSTELLUNGSDATUM
API-Endpoints-Test Postman	HOCH	1	1	19.11.2023
TESTBESCHREIBUNG	TEST DESIGNED BY		TEST AUSGEFÜHRT VON	AUSFÜHRUNGSDATUM
Sicherstellung der Endpoints-Funktionalität	Arda Dursun		Arda Dursun	20.11.2023
TESTBESCHREIBUNG	ABHÄNGIGKEITEN TESTEN		PRÜFBEDINGUNGEN	
Das Ziel besteht darin alle Endpoints aufzurufen und die Responses zu bekommen (Postman Collection wird mitgesendet).			Erfolgreicher API Aufruf und korrekter Response	

SCHRITT-ID	SCHRITTBESCHREIBUNG	TESTDATUM	ERWARTETE ERGEBNISSE	TATSÄCHLICHE ERGEBNISSE	BESTANDEN / NICHT BESTANDEN	ZUSÄTZLICHE HINWEISE
1	Get all orders	20.11.2023	Erwartet war HTTP-Statuscode: 200 und im Responsebody die Tabellen Attribute mit dem über Proxies Nuget aufgelösten Servicetype	Eins zu eins wie zuvor schon erwartet funktionierte alles	Bestanden	Vorausgesetzt man gibt valide Daten im Body mit, da es auch eine Validierung im Backend gibt
2	Get orders by priority	20.11.2023	Erwartet war HTTP-Statuscode: 200 und im Responsebody die Tabellen Attribute mit dem über Proxies Nuget aufgelösten Servicetype	Genau wie erwartet	Bestanden	
3	Search order by name	20.11.2023	Erwartet war HTTP-Statuscode: 200 und im Responsebody die Tabellen Attribute mit dem über Proxies Nuget aufgelösten Servicetype	Funktionierte genau wie zuvor schon beschrieben und das einwandfrei	Bestanden	
4	Post order	20.11.2023	Erwartet war HTTP-Statuscode: 200 und im Responsebody die zugeteilte Id	Alles verlief wie erwartet. Auch die Backend validierung macht seinen Job.	Bestanden	
5	Delete all orders (For Devs)	20.11.2023	Man sollte HTTP Statuscode 204 bekommen als bestätigung	204 ist aufgetreten und alle Bestellungen sind gelöscht	Bestanden	
6	Delete order by id	20.11.2023	Man sollte im Endpoint die ID angeben dann werden direkt alle Bestellungen gelöscht. Man sollte HTTP Statuscode 204 bekommen als bestätigung	Der ausgewählte Auftrag wurde gelöscht	Bestanden	
7	PATCH Comment	20.11.2023	HTTP Statuscode 204 sollte erscheinen	Kommentar wurde angepasst und HTTP Statuscode wurde wie erwartet zurückgegeben	Bestanden	
8	PATCH Status	20.11.2023	HTTP Statuscode 204 sollte erscheinen	Status wurde angepasst und HTTP Statuscode wurde wie erwartet zurückgegeben	Bestanden	
9	Create Employee	20.11.2023	Falls Namen vorhanden sollte im Response-Body erwähnt werden und somit nichts erstellt. Falls eine Erstellung erfolgreich sein sollte, sollte eine Nachricht kommen, dass es erfolgreich war und die ID, die der Mitarbeiter zugewiesen bekommt	Eins zu eins wie zuvor schon erwartet funktionierte alles	Bestanden	
10	Login	20.11.2023	Falls richtige Anmeldedaten wie Passwort usw., dann sollte man den HTTP-Statuscode 200 bekommen und einen JWT-Token. Falls die Anmelde-Daten falsch sind wie ein falsches Passwort, sollte HTTP-Statuscode 401 kommen und im Response-Body, dass es ungültige Anmeldedaten sind. Falls man es zum dritten Mal hintereinander falsch angeben sollte, sollte im Response-Body stehen, dass das Konto gesperrt worden ist. Falls man anschließend probiert sich anzumelden, auch mit korrekten Anmeldedaten, dann sollte im Response-Body stehen, dass das Konto gesperrt ist	Genau wie erwartet	Bestanden	
11	Unlock locked employee	20.11.2023	Erwartet ist HTTP-Statuscode: 200 und im Responsebody die Mitteilung dass der Mitarbeiter entsperrt worden ist	Funktionierte genau wie zuvor schon beschrieben und das einwandfrei	Bestanden	

Ist auf auch GitHub als PDF verfügbar mit hoher Auflösung.



Es ist ausserdem möglich über Postman KI generierte Tests durchzuführen. Dies spart eine Menge Zeit. Aber für die Übersicht habe ich die sicherheitshalber von Hand ausgeführt und KI Generiert, um es abzugleichen.

## 7 Auswerten

### 7.1 Analyse der erreichten Ziele und Anforderungen

#### 7.1.1 Ausgangssituation und Ziele

Das Projekt "Ski-Service Management" der Firma Jetstream-Service zielte darauf ab, die interne Verwaltung von Ski-Service-Aufträgen durch eine web- und datenbankbasierte Anwendung zu digitalisieren. Zu den Hauptzielen gehörten:

1. **Digitalisierung der Auftragsverwaltung:** Ersetzen manueller Prozesse durch ein automatisiertes System.
2. **Zugriffskontrolle für Mitarbeiter:** Implementierung eines passwortgeschützten Zugangssystems.
3. **Effiziente Auftragsbearbeitung:** Ermöglichung der Auftragsannahme, -bearbeitung und -änderung durch die Mitarbeiter.

#### 7.1.2 Erreichte Ziele und Anforderungen

Basierend auf den bereitgestellten Informationen wurden folgende Ziele und Anforderungen erfolgreich umgesetzt:

1. **Web-API mit Authentifikation:**
  - Eine RESTful Web-API wurde entwickelt, um die Interaktion mit der Anwendung zu ermöglichen.
  - Authentifizierung mittels JWT wurde implementiert, um sicherzustellen, dass nur autorisierte Mitarbeiter auf bestimmte Funktionen zugreifen können.
2. **Datenbankdesign und Implementierung:**
  - Das Datenbankdesign wurde erfolgreich umgesetzt, wobei der Code First-Ansatz von Entity Framework Core verwendet wurde.
  - Die Datenbankstruktur unterstützt die Verwaltung von Mitarbeitern, Serviceaufträgen und Servicearten.
3. **Funktionalitäten der Anwendung:**
  - Implementierung von Funktionen zur Anzeige und Mutation von Serviceaufträgen.
  - Ermöglichung der Statusänderung und Löschung von Aufträgen.
4. **Protokollierung:**
  - Protokollierung von API-Aufrufen zur Fehlerlokalisierung und Überwachung.
5. **Einrichtung eines sicheren Systems:**
  - Gewährleistung der Sicherheit durch den Einsatz von Authentifizierung und Autorisierung.
  - Einhaltung von Best Practices in Bezug auf die Sicherheit der Daten.

#### 7.1.3 Herausforderungen und Verbesserungsmöglichkeiten

- **Passwortsicherheit:** Die Passwörter werden aktuell unverschlüsselt gespeichert. Für zukünftige Verbesserungen wird empfohlen, Passwörter sicher zu speichern, beispielsweise durch Hashing.
- **Skalierbarkeit:** Während das aktuelle System die Grundbedürfnisse erfüllt, sollte für zukünftige Erweiterungen die Skalierbarkeit der Anwendung berücksichtigt werden.



#### 7.1.4 Fazit

Das Projekt hat seine Hauptziele der Digitalisierung der Auftragsverwaltung und der Einführung eines effizienten, sicheren Systems für Mitarbeiter erfolgreich erreicht. Es bietet eine solide Basis für die weitere Digitalisierung und Optimierung interner Prozesse. Zukünftige Entwicklungen sollten sich auf die Verbesserung der Sicherheitsaspekte und die Skalierbarkeit der Anwendung konzentrieren.

## 7.2 Reflektion über Herausforderungen und Lösungsansätze

### 7.2.1 Herausforderung: Implementierung von Login und User-Management im Code First-Ansatz

- **Problemstellung:** Eine der Hauptherausforderungen in diesem Projekt war die Implementierung eines effizienten und sicheren Login-Systems sowie die Verwaltung von Benutzerdaten unter Verwendung des Code First-Ansatzes mit Entity Framework Core. Dies umfasste die Erstellung von Benutzerkonten, die Authentifizierung und die Sicherstellung der Datensicherheit.
- **Spezifische Schwierigkeiten:** Die Komplexität lag insbesondere in der Integration der Authentifizierungsfunktionen mit dem bestehenden Datenmodell und der Datenbankstruktur. Dies erforderte ein tiefes Verständnis der Funktionsweise von Entity Framework Core und des Zusammenspiels zwischen den Datenmodellen, der Datenbank und den Authentifizierungsmechanismen.

### 7.2.2 Lösungsansatz und Unterstützung durch einen Kollegen

- **Kollaboration und Wissensaustausch:** Die Überwindung dieser Herausforderung wurde durch die Unterstützung eines erfahrenen Kollegen ermöglicht. Durch gemeinsame Brainstorming-Sitzungen und Code-Reviews konnte eine effektive Lösung erarbeitet werden.
- **Erkenntnisse und Weiterentwicklung:**
  - Diese Herausforderung hat zu einem vertieften Verständnis der Arbeit mit Entity Framework Core und der Implementierung von Sicherheitsmechanismen in einer .NET-Anwendung geführt.
  - Die Erfahrung zeigt die Bedeutung von Teamarbeit und Wissensaustausch, insbesondere bei der Lösung komplexer technischer Probleme.

### 7.2.3 Fazit

Die Bewältigung dieser Herausforderung hat nicht nur zur erfolgreichen Implementierung des Login-Systems und User-Managements beigetragen, sondern auch das technische Wissen und die Teamfähigkeit gestärkt. Diese Erfahrung unterstreicht den Wert von Kollaboration und kontinuierlichem Lernen in der Softwareentwicklung.

## 7.3 Lessons Learned und mögliche Verbesserungen

### 7.3.1 Lessons Learned

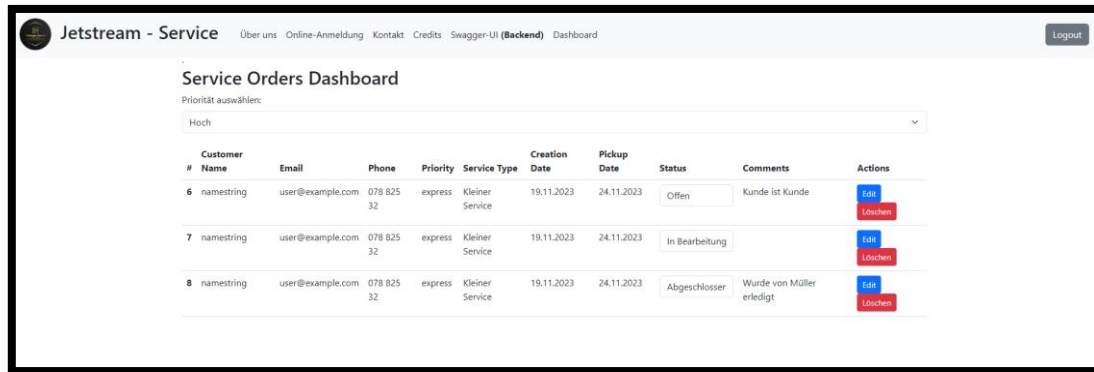
1. **Bedeutung von Teamarbeit und Kollaboration:**
  - Die Herausforderungen bei der Implementierung des Login- und User-Managements verdeutlichten, wie wertvoll der Austausch mit Kollegen und das gemeinsame Brainstorming für die Problemlösung sind.
  - Die Zusammenarbeit förderte kreative Lösungsansätze und den Austausch von Fachwissen.
2. **Tieferes Verständnis von Technologien:**
  - Die Arbeit mit dem Code First-Ansatz und Entity Framework Core bot die Gelegenheit, ein tieferes Verständnis für ORM-Tools und deren Anwendung in realen Projekten zu entwickeln.
  - Die Auseinandersetzung mit Authentifizierungsmechanismen erweiterte das Wissen über Sicherheitsaspekte in Web-Anwendungen.
3. **Flexibilität und Anpassungsfähigkeit:**
  - Die Notwendigkeit, sich neuen Herausforderungen schnell anzupassen, zeigte die Bedeutung von Flexibilität in der Softwareentwicklung.
  - Das Projekt lehrte, wie wichtig es ist, offen für Änderungen in den Anforderungen und im Entwicklungsprozess zu sein.
4. **Wichtigkeit von gründlichem Testing:**
  - Die Erfahrungen unterstrichen die Bedeutung von gründlichem Testing, um sicherzustellen, dass alle Komponenten wie erwartet funktionieren.
  - Es wurde deutlich, dass frühes und regelmässiges Testen hilft, Probleme frühzeitig zu erkennen und zu beheben.

### 7.3.2 Mögliche Verbesserungen

1. **Verbesserung der Passwortsicherheit:**
  - Die Implementierung eines sicheren Verfahrens zur Speicherung von Passwörtern, wie Hashing und Salting, sollte priorisiert werden, um die Sicherheit der Anwendung zu erhöhen.
2. **Erhöhung der Skalierbarkeit:**
  - Für zukünftige Erweiterungen sollte die Skalierbarkeit der Anwendung berücksichtigt werden, insbesondere im Hinblick auf die zunehmende Datenmenge und Benutzerzahl.
3. **Erweiterung des Testumfangs:**
  - Die Implementierung umfassenderer Testverfahren, einschliesslich Integrationstests und möglicherweise automatisierter UI-Tests, könnte die Qualität und Zuverlässigkeit der Anwendung weiter verbessern.
4. **Dokumentation und Wissensmanagement:**
  - Die Erstellung einer detaillierteren Dokumentation des Entwicklungsprozesses und der Systemarchitektur könnte zukünftigen Entwicklern helfen, schneller einen Überblick über das Projekt zu gewinnen.
5. **Feedback-System für Endnutzer:**
  - Ein formelles Feedback-System für Endnutzer könnte implementiert werden, um Rückmeldungen über die Anwendungsnutzung zu sammeln und auf dieser Basis Verbesserungen vorzunehmen.

## 8 Abschluss-Fazit + Screenshots

### 8.1 Screenshots der Anwendung

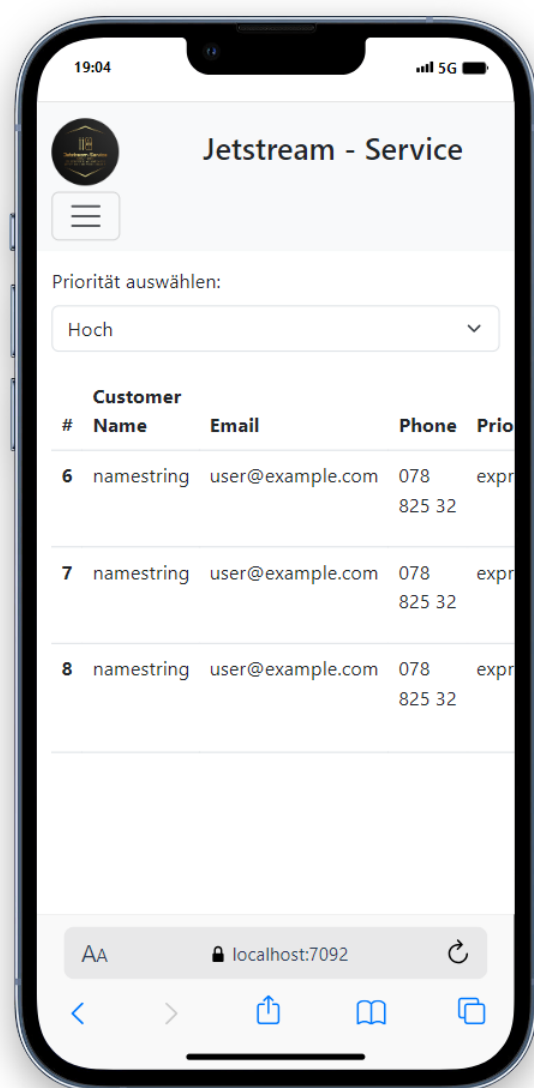
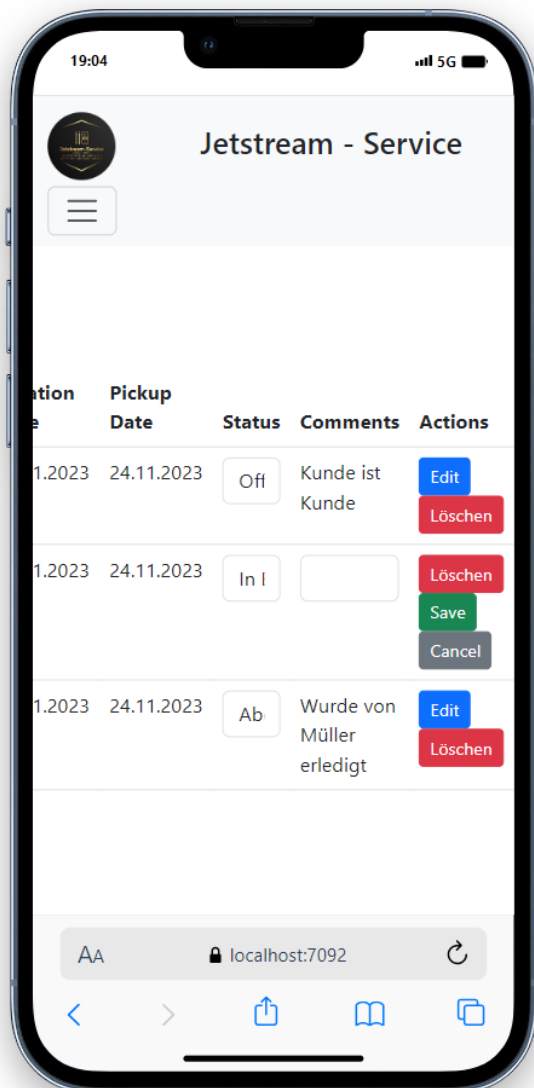


**Jetstream - Service** | Über uns | Online-Anmeldung | Kontakt | Credits | Swagger-UI (Backend) | Dashboard | Logout

### Service Orders Dashboard

Priorität auswählen:  
Hoch

Customer #	Name	Email	Phone	Priority	Service Type	Creation Date	Pickup Date	Status	Comments	Actions
6	namestring	user@example.com	078 825 32	express	Kleiner Service	19.11.2023	24.11.2023	Offen	Kunde ist Kunde	<a href="#">Edit</a> <a href="#">Löschen</a>
7	namestring	user@example.com	078 825 32	express	Kleiner Service	19.11.2023	24.11.2023	In Bearbeitung		<a href="#">Edit</a> <a href="#">Löschen</a>
8	namestring	user@example.com	078 825 32	express	Kleiner Service	19.11.2023	24.11.2023	Abgeschlossener	Wurde von Müller erledigt	<a href="#">Edit</a> <a href="#">Löschen</a>



## 8.2 Datenbank-Schemata



## 8.3 Fazit

Die gewonnenen Erkenntnisse und die identifizierten Verbesserungsmöglichkeiten zeigen, dass das Projekt nicht nur zur Erreichung seiner Ziele beigetragen hat, sondern auch eine wertvolle Lernerfahrung für das gesamte Entwicklungsteam darstellte. Diese Erfahrungen bilden eine solide Grundlage für zukünftige Projekte und deren Weiterentwicklung. Ich muss ehrlich sagen, dass dieses Projekt wirklich sehr anspruchsvolle Anforderungen hatte, für das wir einen Wissensstand von 5 Modultagen haben. Dennoch gab ich mein Bestes und probierte alle Anforderungen auf jegliche Art zu erfüllen.

Durch dieses Projekt nahm ich viel mit. Würde man mich erneut auffordern, ein Backend zu erstellen, bin ich überzeugt davon das jetzt noch besser umsetzen zu können.