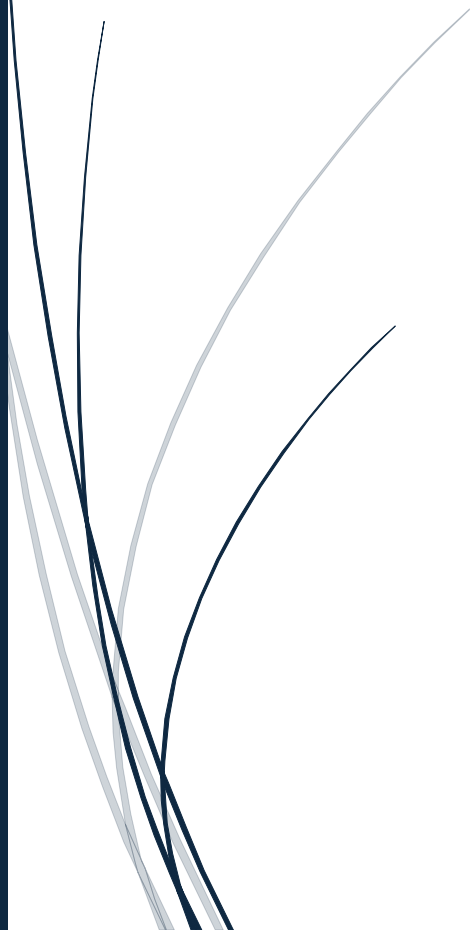




9.12.2024

# SkiServiceDB

ICT Modul 295



Sebastian Thurnbauer, Maximilian Lapple  
IPSO

## Inhaltsverzeichnis

Abbildungsverzeichnis .....	1
Einleitung .....	2
Verwendete Tools.....	2
Projektumfeld .....	3
Projektziele.....	3
Planen.....	4
Entscheiden .....	4
Realisieren .....	5
Projektübersicht.....	5
Technologien.....	5
Controller .....	5
AuthController.....	5
OrdersController .....	5
ServicesController.....	6
Services.....	6
AuthService.....	6
OrderService .....	6
Modelle.....	7
Order.....	7
Service .....	7
Employee .....	7
Datenbankzugriff.....	8
Swagger .....	8
SQL Management Studio .....	8
Kontrollieren.....	9
Auswertung .....	10
Fazit .....	11

## Abbildungsverzeichnis

Abbildung 1:AuthController .....	5
Abbildung 2: OrdersController .....	6
Abbildung 3: ServicesController.....	6
Abbildung 4: AuthService.....	6
Abbildung 5: OrderService .....	7
Abbildung 6: ClassOrder .....	7
Abbildung 7: ClassService .....	7
Abbildung 8: ClassEmployee .....	7

Abbildung 9: DBContext.....	8
Abbildung 10: Swagger .....	8
Abbildung 11: DB.....	9

## Einleitung

Die Firma Jetstream-Service, ein KMU für Winterskiservice, plant die Digitalisierung ihrer Auftragsverwaltung mit einer web- und datenbankbasierten Anwendung. Die bestehende Online-Anmeldung wird um Funktionen für das Auftragsmanagement erweitert. Bis zu zehn Mitarbeiter sollen in der Hauptsaison über einen passwortgeschützten Zugang Aufträge einsehen, bearbeiten und aktualisieren können.

Dieses Teilprojekt umfasst die Backend-Entwicklung gemäß IPERKA und beinhaltet:

- Erstellung einer Web-API mit Authentifikation und OpenAPI-Dokumentation
- Datenbankdesign und -implementierung
- Unit-Tests und Testpläne
- Umsetzung und Test der Anwendung (z. B. mit Postman)

Lesende Datenoperationen sind ohne Authentifikation möglich, Änderungen nur für autorisierte Nutzer.

## Verwendete Tools

**JetBrains Rider:** Für die Entwicklung wurde JetBrains Rider genutzt, da die integrierte Entwicklungsumgebung (IDE) eine hervorragende Unterstützung für .NET-Projekte bietet und die Entwicklung effizient gestaltet. Die umfangreichen Debugging- und Refactoring-Tools erleichtern die Arbeit und verbessern die Codequalität.

**Postman:** Zum Testen der API wurde Postman eingesetzt. Dieses Tool bietet eine intuitive Oberfläche, um API-Endpunkte zu testen, Anfragen zu senden und die Antworten detailliert zu analysieren. Es war besonders hilfreich bei der Durchführung und Dokumentation von Testszenarien.

**Swagger (OpenAPI):** Zur Dokumentation und Visualisierung der API wurde Swagger verwendet. Damit konnten die Endpunkte automatisch dokumentiert und ein benutzerfreundliches Interface für die API-Interaktion bereitgestellt werden, was die Zusammenarbeit zwischen Entwicklung und Testteams erleichterte.

## Informieren

Zu Beginn des Projekts lasen wir den Arbeitsauftrag gründlich durch und besprachen offene Punkte gemeinsam. Nachdem alle Unklarheiten geklärt waren, analysierten wir die im Auftrag beschriebenen Schritte und Anforderungen. Daraufhin überlegten wir uns verschiedene Ansätze für die Umsetzung des Projekts und informierten uns über passende Tools, die dabei helfen könnten. Zudem setzten wir uns intensiv mit den Bewertungskriterien auseinander, um

sicherzustellen, dass alle Anforderungen erfüllt werden. Abschließend recherchierten wir im Internet nach weiteren Ideen und Best Practices für die erfolgreiche Umsetzung des Projekts.

## Projektumfeld

Die Jetstream-Service GmbH ist ein KMU, das in der Wintersaison Skiservicearbeiten durchführt. Zur Digitalisierung der internen Verwaltung möchte das Unternehmen die Aufträge künftig über eine Web- und Datenbankanwendung abwickeln. Die bestehende Online-Anmeldung wird beibehalten und um Funktionen für das Auftragsmanagement erweitert. In der Hauptsaison arbeiten bis zu 10 Mitarbeiter an den Aufträgen, die über einen passwortgeschützten Zugang Änderungen vornehmen können.

- Das Projekt umfasst den Backend-Teil und beinhaltet:
- Entwicklung einer Web-API mit Authentifizierung (inkl. OpenAPI-Dokumentation).
- Datenbankdesign und -implementierung (Code- oder Database First).
- Testplanung und Unit-Tests.
- Durchführung der Tests (Postman).

Ziel ist es, eine sichere und effiziente Lösung zur Verwaltung der Aufträge zu schaffen, bei der Änderungen nur nach erfolgreicher Authentifizierung durch Mitarbeiter erfolgen können.

## Projektziele

### 1. Digitale Verwaltung der Aufträge:

Implementierung einer Web- und Datenbankanwendung, die eine effiziente und papierlose Verwaltung der Skiservice-Aufträge ermöglicht.

### 2. Erweiterung der bestehenden Online-Anmeldung:

Integration zusätzlicher Funktionen für das Auftragsmanagement in die bereits existierende Online-Anmeldung.

### 3. Sichere Authentifizierung:

Entwicklung eines passwortgeschützten Zugangs für Mitarbeiter, um Auftragsdaten sicher bearbeiten und ändern zu können.

### 4. Optimierung der Arbeitsprozesse:

Verbesserung der internen Abläufe durch eine benutzerfreundliche und schnelle Zugriffsmöglichkeit auf Auftragsdaten in der Hauptsaison.

### 5. Dokumentation und Qualitätssicherung:

Sicherstellung der Funktionalität und Stabilität durch umfassende Tests (Unit-Tests, Postman) und eine detaillierte OpenAPI-Dokumentation.

### 6. Nachhaltige Systemintegration:

Gewährleistung einer flexiblen Backend-Lösung, die zukünftige Erweiterungen und die Anbindung an mögliche Frontend-Systeme problemlos ermöglicht.

## Planen

Nr.	Beschreibung	SOL-Zeit	IST-Zeit
<b>1.</b>	<b>Informieren</b>	<b>3</b>	<b>2.5</b>
1.1	Situationsanalyse	2h	2h
1.2	Tool-Installation	1h	0.5h
<b>2</b>	<b>Planen</b>	<b>6</b>	<b>6</b>
2.1	Zeitplan	1.5h	1h
2.2	Datenbankdesign	1h	1h
2.3	Systemarchitektur	1.5h	1h
2.4	Spezifizierung API-Endpunkte	2h	3h
<b>3</b>	<b>Entscheiden</b>	<b>1</b>	<b>0.5</b>
3.1	Code- oder Database-First	1	0.5
<b>4</b>	<b>Realisieren</b>	<b>25</b>	<b>16.5</b>
4.1	Grundlegendes Projekt	1h	1h
4.2	Git-Repository	1h	0.5h
4.3	Implementierung DB	3h	2h
4.4	Entwicklung Backend	12h	7h
4.5	Entwicklung und Integration Frontend	8h	6h
<b>5</b>	<b>Kontrollieren</b>	<b>3</b>	<b>2</b>
5.1	Tests	2h	1.5h
5.2	Anforderungen vergleichen	1h	0.5h
<b>6</b>	<b>Auswerten</b>	<b>5</b>	<b>3</b>
6.1	Finalisierung der Dokumentation	3h	2h
6.2	Lessons-Learned	1.5h	0.5h
6.3	Abgabe One-Note	0.5h	0.5h
<b>Gesamt</b>		<b>43</b>	<b>30.5</b>

## Entscheiden

### Entscheidung für JetBrains Rider

Wir haben uns für JetBrains Rider entschieden, da es im Vergleich zu Visual Studio eine schnellere und ressourcenschonendere Entwicklungsumgebung bietet. Rider unterstützt eine Vielzahl von Programmiersprachen und Technologien und ist besonders leistungsfähig bei der Arbeit mit .NET-Projekten. Zudem gefällt uns die benutzerfreundliche Oberfläche und die effektiven Tools zur Code-Navigation und -Analyse, die eine effizientere Entwicklung ermöglichen.

### Entscheidung für Database-First Ansatz

Wir haben uns für den Database-First Ansatz entschieden, da wir etwas Neues ausprobieren wollten, anstatt wie gewohnt den Code-First Ansatz zu verwenden. Mit Database-First können wir direkt mit einer bestehenden oder geplanten Datenbankstruktur arbeiten, was uns eine präzisere Kontrolle über das Datenbankschema und die Tabellenstrukturen ermöglicht. Dieser Ansatz bietet uns zudem eine gute Grundlage für die Integration und spätere Erweiterung der Anwendung.

# Realisieren

## Projektübersicht

Das Projekt "Serviceauftrag Management" ist eine Webanwendung, die es Mitarbeitern ermöglicht, Serviceaufträge zu verwalten. Die Anwendung nutzt ASP.NET Core für die Backend-Entwicklung und implementiert eine RESTful API, die CRUD-Operationen für Aufträge und Dienstleistungen bereitstellt. Die Authentifizierung erfolgt über JWT (JSON Web Tokens), und die API-Dokumentation wird mit Swagger bereitgestellt.

## Technologien

- **ASP.NET Core:** Framework für die Entwicklung der Webanwendung.
- **Entity Framework Core:** ORM (Object-Relational Mapping) für den Datenzugriff.
- **SQL Server:** Datenbankmanagementsystem zur Speicherung der Daten.
- **Swagger:** Tool zur Dokumentation und Visualisierung der API.
- **JWT:** Methode zur Authentifizierung und Autorisierung.

## Controller

Die Controller sind für die Verarbeitung von HTTP-Anfragen verantwortlich und implementieren die Logik für die CRUD-Operationen.

### *AuthController*

**Zweck:** Handhabt die Authentifizierung von Benutzern.

**Methoden:**

- Login: Überprüft die Anmeldedaten eines Mitarbeiters und gibt ein JWT zurück.

```
1 [HttpPost("login")]|
2 public IActionResult Login([FromBody] LoginRequest request) {
3     // Logik zur Überprüfung der Anmeldedaten
4 }
```

Abbildung 1:AuthController

### *OrdersController*

**Zweck:** Verwalten von Serviceaufträgen.

**Methoden:**

- GetOrders: Gibt eine Liste aller Aufträge zurück.
- GetOrderByid: Gibt einen spezifischen Auftrag anhand der ID zurück.
- CreateOrder: Erstellt einen neuen Auftrag.
- UpdateOrderStatus: Aktualisiert den Status eines Auftrags.

- DeleteOrder: Markiert einen Auftrag als gelöscht.

```
[HttpGet]
public async Task<ActionResult<IEnumerable<OrderDto>>> GetOrders() {
    // Logik zum Abrufen aller Aufträge
}
```

Abbildung 2: OrdersController

### ServicesController

**Zweck:** Verwalten von Dienstleistungen.

**Methoden:**

- GetServices: Gibt eine Liste aller verfügbaren Dienstleistungen zurück.

```
[HttpGet]
public async Task<ActionResult<IEnumerable<Service>>> GetServices() {
    // Logik zum Abrufen aller Dienstleistungen
}
```

Abbildung 3: ServicesController

## Services

Die Services enthalten die Geschäftslogik und kommunizieren mit der Datenbank über den DbContext.

### AuthService

**Zweck:** Handhabt die Authentifizierung und Token-Generierung.

**Methoden:**

- VerifyPassword: Überprüft das eingegebene Passwort.
- GenerateJwtToken: Generiert ein JWT für den authentifizierten Benutzer.

```
public string GenerateJwtToken(Employee employee) {
    // Logik zur Token-Generierung
}
```

Abbildung 4: AuthService

### OrderService

**Zweck:** Verarbeitet die Logik für Aufträge.

**Methoden:**

- GetOrdersAsync: Holt alle Aufträge.

- GetOrderByIDAsync: Holt einen Auftrag anhand der ID.
- CreateOrderAsync: Erstellt einen neuen Auftrag.
- MarkOrderAsDeleted: Markiert einen Auftrag als gelöscht.

```
public async Task<Order> CreateOrderAsync(OrderCreateRequest request) {
    // Logik zur Erstellung eines neuen Auftrags
}
```

Abbildung 5: OrderService

## Modelle

Die Modelle repräsentieren die Datenstrukturen in der Anwendung.

### Order

- **Eigenschaften:** OrderID, CustomerName, Email, Phone, Priority, ServiceID, Status, IsDeleted, DateCreated, DateModified.

```
public class Order {
    public int OrderID { get; set; }
    public string CustomerName { get; set; }
    // Weitere Eigenschaften
}
```

Abbildung 6: ClassOrder

### Service

- **Eigenschaften:** ServiceID, ServiceName.

```
public class Service {
    public int ServiceID { get; set; }
    public string ServiceName { get; set; }
}
```

Abbildung 7: ClassService

### Employee

- **Eigenschaften:** EmployeeID, Username, Password, IsAdmin.

```
public class Employee {
    public int EmployeeID { get; set; }
    public string Username { get; set; }
    // Weitere Eigenschaften
}
```

Abbildung 8: ClassEmployee



## Datenbankzugriff

Die Anwendung verwendet Entity Framework Core, um mit der SQL Server-Datenbank zu interagieren. Der **ApiDbContext** verwaltet die Datenbankverbindung und die DbSet für die Modelle.

```
public class ApiDbContext : DbContext {  
    public ApiDbContext(DbContextOptions<ApiDbContext> options) : base(options) {  
        // DbSet für die Modelle  
        public DbSet<Order> Orders { get; set; }  
        public DbSet<Service> Services { get; set; }  
        public DbSet<Employee> Employees { get; set; }  
    }  
}
```

Abbildung 9: DbContext

## Swagger

Swagger wird verwendet, um die API-Dokumentation bereitzustellen. Die Swagger-Konfiguration wird in der **Startup.cs-Datei** vorgenommen.

Auth	
POST	/api/auth/login
Orders	
GET	/api/Orders
POST	/api/Orders
GET	/api/Orders/{id}
DELETE	/api/Orders/{id}
GET	/api/Orders/priority/{priority}
PUT	/api/Orders/{id}/status
Services	
GET	/api/Services

Abbildung 10: Swagger

## SQL Management Studio

SQL Management Studio wird verwendet, um die SQL Server-Datenbank zu verwalten und zu überwachen. Es ermöglicht das Erstellen von Tabellen, das Ausführen von Abfragen und das Überprüfen von Daten.

Employees			Orders			Services		
Column Name	Data Type	Allow Nulls	Column Name	Data Type	Allow Nulls	Column Name	Data Type	Allow Nulls
EmployeeID	int	<input type="checkbox"/>	OrderID	int	<input type="checkbox"/>	ServiceID	int	<input type="checkbox"/>
Username	nvarchar(255)	<input type="checkbox"/>	CustomerName	nvarchar(100)	<input type="checkbox"/>	ServiceName	nvarchar(100)	<input type="checkbox"/>
Password	nvarchar(255)	<input type="checkbox"/>	Email	nvarchar(100)	<input type="checkbox"/>			<input type="checkbox"/>
FailedLoginAttempts	int	<input checked="" type="checkbox"/>	Phone	nvarchar(20)	<input checked="" type="checkbox"/>			
IsLocked	Employees	<input checked="" type="checkbox"/>	Priority	nvarchar(50)	<input type="checkbox"/>			
IsAdmin	bit	<input checked="" type="checkbox"/>	ServiceID	int	<input type="checkbox"/>			
Role	nvarchar(50)	<input checked="" type="checkbox"/>	Status	nvarchar(20)	<input type="checkbox"/>			
		<input type="checkbox"/>	IsDeleted	bit	<input type="checkbox"/>			
			DateCreated	datetime	<input type="checkbox"/>			
			DateModified	datetime	<input checked="" type="checkbox"/>			
					<input type="checkbox"/>			

Abbildung 11: DB

## Kontrollieren

Testfall-ID	Beschreibung	Eingabewerte	Erwartetes Ergebnis	Tatsächliches Ergebnis	Status	Anmerkung
1	Benutzeranmeldung mit gültigen Anmeldedaten	Username, Passwort	JWT-Token wird generiert und zurückgegeben	JWT-Token generiert	Bestanden	-
2	Benutzeranmeldung mit ungültigen Anmeldedaten	Username: "testuser", Passwort: "invalidpassword"	Unauthorized-Fehler	Unauthorized-Fehler	Bestanden	-
3	Erstellung eines neuen Auftrags	CustomerName: "Max Mustermann", Email: "max@example.com", Phone: "123456789", Priority: "Hoch", ServiceID: 1	Auftrag wird erfolgreich erstellt und zurückgegeben	Auftrag wird erfolgreich erstellt	Bestanden	-
4	Abrufen aller Aufträge	-	Liste der Aufträge wird zurückgegeben	Liste der Aufträge wird zurückgegeben	Bestanden	-
5	Abrufen eines Auftrags mit gültiger ID	ID: 1	Auftrag mit ID 1 wird zurückgegeben	Auftrag mit ID 1 wird zurückgegeben	Bestanden	-
6	Abrufen eines Auftrags mit ungültiger ID	ID: 999	NotFound-Fehler	NotFound-Fehler	Bestanden	-

7	Markieren eines Auftrags als gelöscht	ID: 1	Auftrag wird als gelöscht markiert	Auftrag wird als gelöscht markiert	Bestanden	-
8	JWT-Token-Überprüfung bei geschützten Endpunkten	JWT-Token	Zugriff auf geschützten Endpunkt	Zugriff nicht gewährt	Nicht Bestanden	Fehler
9	Zugriff auf geschützten Endpunkt ohne Token	-	Unauthorized-Fehler	Unauthorized-Fehler	Bestanden	-
10	Filtern von Aufträgen nach Priorität	Priority: "Hoch"	Liste der Aufträge mit Priorität "Hoch" wird zurückgegeben	Liste der Aufträge mit Priorität "Hoch" wird zurückgegeben	Bestanden	-

### Zusammenfassung der Testergebnisse

- Anzahl der durchgeführten Tests: 10
- Bestanden: 9
- Nicht Bestanden: 1
- Gesamtstatus: 9 Tests bestanden einer nicht

## Auswertung

Das Projekt wurde erfolgreich abgeschlossen, auch wenn wir während der Umsetzung auf einige Herausforderungen gestoßen sind. Eines der größten Probleme war die anfängliche Verbindung zur SSMS-Datenbank. Ein Mitglied unseres Teams hatte Schwierigkeiten, sich mit der Datenbank zu verbinden, was uns vorübergehend daran hinderte, effizient zusammenzuarbeiten. Nach eingehender Analyse und gemeinsamen Bemühungen gelang es uns jedoch, dieses Problem zu lösen und sicherzustellen, dass alle Teammitglieder uneingeschränkten Zugriff auf die Datenbank hatten.

Ein weiteres Hindernis war die Implementierung des JWT-Tokens. Dieses stellte sich als technisch anspruchsvoll heraus, da es bestimmte Anforderungen an die Authentifizierung und Sicherheitsmechanismen gab, die zunächst nicht wie erwartet funktionierten. Durch Recherche, Tests und Teamarbeit konnten wir jedoch auch diese Schwierigkeiten überwinden und eine stabile Lösung entwickeln.

Trotz dieser Herausforderungen war die Zusammenarbeit innerhalb des Teams hervorragend. Die Kommunikation war klar und zielorientiert, was maßgeblich dazu beitrug, die Probleme rechtzeitig zu bewältigen.

## Fazit

Das Projekt war eine lehrreiche und bereichernde Erfahrung. Besonders stolz sind wir darauf, dass wir trotz der anfänglichen Schwierigkeiten und des hohen technischen Anspruchs eine funktionierende Lösung entwickeln konnten. Das Projekt hat nicht nur unsere fachlichen Fähigkeiten, sondern auch unsere Team- und Problemlösungskompetenzen gestärkt.

Ein wichtiger Lerneffekt war die Erkenntnis, dass ein frühzeitiger Start und eine bessere Zeitplanung entscheidend für den Projekterfolg sind. Leider haben wir die ersten Phasen des Projekts unterschätzt und standen am Ende unter erheblichem Zeitdruck, da wir viele Aufgaben erst in letzter Minute erledigten.

Nichtsdestotrotz haben wir alle Anforderungen erfüllt und ein Ergebnis erzielt, auf das wir stolz sein können. Besonders wertvoll war die Möglichkeit, praktische Erfahrungen zu sammeln und unser Wissen in den Bereichen Datenbankmanagement, API-Implementierung und Token-basierte Authentifizierung zu vertiefen. Diese Erfahrungen werden uns in zukünftigen Projekten zweifellos von großem Nutzen sein.