

Modul 295

Projektdokumentation nach IPERKA



17. Februar 2025

Even Stuck

Inhalt

[1 Informieren 2](#_Toc190717866)

[1.1 Projektbeschreibung 2](#_Toc190717867)

[1.2 Verwendete Technologien 2](#_Toc190717868)

[1.3 Aufgabenstellung / Anforderungen 3](#_Toc190717869)

[2 Planen 4](#_Toc190717870)

[2.1 Hauptkomponenten der «Lösung» 4](#_Toc190717871)

[2.2 Essenzielles 4](#_Toc190717872)

[2.3 Herausforderungen 4](#_Toc190717873)

[3 Entscheiden 5](#_Toc190717874)

[3.1 Alternativen und Begründungen 5](#_Toc190717875)

[3.2 Berechtigungsmatrix 5](#_Toc190717876)

[4 Realisieren 6](#_Toc190717877)

[4.1 Implementierung der Hauptfunktionen 6](#_Toc190717878)

[4.2 6](#_Toc190717879)

[5 Kontrollieren 8](#_Toc190717880)

[5.1 API-Tests mit Postman 8](#_Toc190717881)

[6 Auswerten 9](#_Toc190717882)

[6.1 Projektbewertung 9](#_Toc190717883)

# Informieren

## Projektbeschreibung

Das Ziel dieses Projekts ist die Entwicklung einer RESTful Web-API für eine SkiService-Website. Damit sollte man, Serviceaufträge digital zu verwalten können, damit die Skiservice-Mitarbeiter ihre Aufträge effizient abrufen und bearbeiten können.

**Hauptanforderungen**

* Benutzerverwaltung (RBAC; Admin, Mitarbeiter, Benutzer)
* JWT-Authentifizierung
* Die CRUD-Funktionen
* Datenbank mit Entity Framework Core designen
* Passwort-Hashing mit BCrypt
* API-Doku mit Swagger

## Verwendete Technologien

|  |  |
| --- | --- |
| **Technologie** | **Einsatzbereich** |
| C# (.NET Core 8) | Backend-Entwicklung des API |
| Entity Framework Core | Datenbankzugriff mit ORM |
| MS SQL Server | Datenbank-Management |
| JWT | Authentifizierung |
| BCrypt | Passwort-Hashing |
| Swagger | API-Doku |
| Postman | API-Testing |
| Serilog | Logging und Fehlerprotokollierung |
| GitHub | Versionskontrolle |

## Aufgabenstellung / Anforderungen

Die Aufgabe ist das bestehende SkiService-Management, um eine digitale Verwaltung zu erweitern. Dafür entwickle ich eine Web-API[[1]](#footnote-1) welche folgenden Anforderungen erfüllen muss:

**Funktionale Anforderungen**  
Benutzerverwaltung mit drei Rolllen:

* Admin: Hat Vollzugriff …. Ist halt der Admin
* Mitarbeiter: Aufträge abrufen, bearbeiten und löschen
* Benutzer: Kann sich selbst und seine Aufträge ansehen

**Auftragsmanagement**

* Aufträge nach Priorität filtern
* Statusänderungen vornehmen
* Kunden können Aufträge stornieren -> löschen

**Authentifizierung und Sicherheit**

* Anmeldung erfolgt per JWT-Token
* Passwörter werden mit BCrypt gehasht gespeichert

**Protokollierung und Fehlerbehandlung**

* Serilog für das Logging
* Datenvalidierung für die Eingaben (also keine 10^99 Zeichen Benutzernamen)

**Datenbank und API-Doku**

* Datenbank in normalisiert in der 3NF …
* Swagger-Doku für die Endpoints
* SQL-Querries

# Planen

## Hauptkomponenten der «Lösung»

* Datenbank -> Für die Speicherung der Benutzer und Aufträge
* Web-API-> Schnittstelle für Benutzerinteraktion
* Authentifizierung -> JWTs für die Sicherheit
* Datenvalidierung und Logging -> Fehlerprotokollierung und Serilog

## Essenzielles

* Passwort-Hashing: Die Passwörter dürfen natürlich auf keinen Fall als Klartext in der DB gespeichert sein. Deshalb verwende ich BCrypt.
* Authentifizierung: Da ich ja eine RBAC einrichte muss eine Authentifizierung her. Ich habe mich darauf entschlossen JWTs zu verwenden.

## Herausforderungen

Wenn ich mir dieses Projekt so ansehe, entdecke ich folgende mögliche Probleme:

* Passwörter richtig mit BCrypt speichern.
* JWTs richtig einrichten
* Unique Constraints richtig definieren und auch einhalten
* Kleine Fehler welche mir unterlaufen sind (ich hatte ein «s» vergessen …) und mich einige Stunden kosten diese zu fixen

# Entscheiden

## Alternativen und Begründungen

Dieser Punkt ist schnell abgeschlossen, da relativ viel vorgegeben wurde. Deshalb nenne ich hier ein paar wenige Sachen, welche ich wirklich selbst entschieden habe.

* PBKDF2 statt BCrypt: Die Implementierung von PBKDF2 ist um einiges Komplexer, deshalb habe ich mich für BCrypt entschieden.
* Basic Auth statt JWT: Unsicher, da Benutzername/Passwort bei jeder Anfrage gesendet wird
* MySQL statt MS SQL: Meine Mutter hat mal gesagt man soll Markenprodukte nicht mischen und einfach bei einer bleiben.
* Serviceklassen verwenden ?: Meiner Meinung nach ist es einfacher für die Entwicklung keine Serviceklassen zu verwenden. Deshalb werde ich es lassen.

## Berechtigungsmatrix

Folgende Berechtigungsmatrix zeigt die getroffenen Entscheidungen für die Berechtigungen. Diese wurde **mit Chat-GPT generiert**:  
Ein Bild, das Text, Screenshot, Software enthält.

KI-generierte Inhalte können fehlerhaft sein.

# Realisieren

## Implementierung der HauptfunktionenHauHaupt

## 

Ich habe die API schrittweise entwickelt -> Ein Modul nach dem anderen. Als erstes einfach Orders returnen. Danach die Authentifizierung mit JWTs usw.

**Benutzerverwaltung und Authentifizierung**

* Benutzer können sich **registrieren & anmelden**.
* Passwörter werden mit **BCrypt gehasht**.
* JWT-Token wird für **sichere Sitzungen** genutzt.

Eine vereinfachte Version einer **Registrierung mit BCrypt** mit meiner API sieht folgendermassen aus (Jedoch als Screenshot. Ich möchte dir gepasteten Code in Word ersparen, da es wahrlich Augenkrebs ist):  
Ein Bild, das Text, Screenshot, Software, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

**JWT-Authentifizierung**

* Jeder Benutzer bekommt nach dem Login einen JWT
* Dieser Token wird dann für die Endpoints benötigt

Hier wieder eine vereinfachte Version meines Codes. Diesmal jedoch für die **Token Erstellung**:

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

Naja, den richtigen Code kann man sich im GitHub ansehen :)

**Datenbank mit Hilfe des Entity Framework**

Hierfür habe ich zwei Model und eine DBContext Klasse erstellt -> Code-First-Ansatz. Danach muss man lediglich diese Commands im Terminal (mit richtigem Dateipfad) eingeben:

dotnet ef migrations add InitialCreate

Um die Migration zu erstellen.

dotnet ef database update

Hiermit wird die Datenbank auch effektiv mit dieser Migration geupdated. Diesen Schritt nicht vergessen!

Vorher auch sicherstellen, dass die nötigen Libraries installiert sind.

# Kontrollieren

## API-Tests mit Postman

Ich habe alle API-Funktionen mithilfe von Postman getestet (Postman > REST-Client)

* Registrierung eines neuen Benutzers
* Anmeldung mit gültigen und ungültigen anmelde Daten
* Zugriff auf geschützte API-Endpoints mit JWT
* CRUD-Operationen, mit jeder Rolle (User, Mitarbeiter, Admin)
* Rollensystem funktioniert(z.B. User kann keine Benutzerrollen ändern)

Die Postman Collection ist ebenfalls im GitHub hinterlegt

# Auswerten

## Projektbewertung

**Positives**:

* Benutzerverwaltung mit JWTs und BCrypt funktioniert Einwand frei
* API ist sauber mit Swagger dokumentiert
* Logging ist mit Serilog vorhanden

**Negatives**:

* Viel Zeit bei unnötigen Bugs verloren (z.B. Schreibfehler)
* Immer viel am Stück gearbeitet, Zeit besser einteilen

1. Es sollte eigentlich das API sein -> (Application Programming Interface). Die klingt halt besser. [↑](#footnote-ref-1)