# Doku Modul 295

Maric Lang

**IPERKA-Dokumentation: Service-Auftragsmanagement Backend (Web-API mit Authentifikation)**

## Inhalt



---

# 1. Informieren

**Projektbeschreibung**

Das Ziel des Projekts ist die Entwicklung einer Web-API für das Service-Auftragsmanagement eines Unternehmens. Mitarbeiter können sich anmelden, Serviceaufträge verwalten und bearbeiten. Kunden können Serviceaufträge online erfassen.

**Anforderungen**

- Web-API mit Authentifikation (JWT-Token)
- OpenAPI-Dokumentation (Swagger)
- Code-First-Datenbank mit Microsoft SQL Server
- Serviceaufträge erstellen, anzeigen, ändern und löschen
- Authentifikation erforderlich für Änderungen & Löschungen
- Bereitstellung über **ASP.NET Core (C#) mit Entity Framework Core**

---

# 2. Planen

**Technologien & Tools**

- **Programmiersprache:** C# (ASP.NET Core Web API)
- **Datenbank:** Microsoft SQL Server (EF Core Code-First)
- **Sicherheit:** BCrypt (Passwort-Hashing), JWT-Token (Authentifizierung)
- **Testen:** Postman (API-Tests)

- **Entwicklungsumgebung:** Visual Studio 2022
- **Versionskontrolle:** Git

**Projektstruktur & Meilensteine**

1. **Backend aufsetzen** (ASP.NET Core, EF Core, SQL Server)
2. **Modelle & Datenbank erstellen** (Code-First Migration)
3. **API-Endpunkte implementieren** (CRUD für Serviceaufträge)
4. **Authentifizierung & Autorisierung hinzufügen** (JWT, Login)
5. **Swagger-Dokumentation bereitstellen**

---

# 3. Entscheiden

**Architektur-Entscheidungen**

- **Code-First-Ansatz für die Datenbank** (keine manuelle SQL-Erstellung)
- **JWT-Token zur Authentifizierung** (keine Cookies, keine Sessions)
- **Entity Framework Core für ORM**
- **Dependency Injection für saubere Trennung von Services & Controllern**

**API-Endpunkte**

- **POST /api/auth/login** → Login mit JWT-Token
- **POST /api/auth/register** → Mitarbeiter registrieren
- **GET /api/serviceauftrag** → Serviceaufträge anzeigen (keine Auth erforderlich)
- **POST /api/serviceauftrag** → Neuen Serviceauftrag erstellen (Auth erforderlich)
- **PUT /api/serviceauftrag/{id}** → Auftrag aktualisieren (Auth erforderlich)
- **DELETE /api/serviceauftrag/{id}** → Auftrag löschen (Auth erforderlich)

---

# 4. Realisieren

**Implementierte Features**

- **Backend-Struktur aufgesetzt** (ASP.NET Core, EF Core)
- **Datenbank mit Code-First-Migration erstellt**
- **API-Endpunkte mit Repository-Pattern erstellt**
- **Authentifizierung & Autorisierung mit JWT implementiert**
- **CRUD-Operationen für Serviceaufträge fertiggestellt**

- **Swagger-Dokumentation integriert**

---

## 5. Kontrollieren

**Tests mit Postman**

- **Endpunkte manuell getestet**
- **Falsche Anfragen geprüft** (z. B. falsches Login, unautorisierte Änderungen)
- **Datenbank überprüft** (richtige Speicherung der Daten)
- **Fehler-Logs analysiert & Fehler behoben**

---

## 6. Auswerten

**Ergebnisse & Fazit**

- **Web-API funktioniert wie geplant**
- **Authentifikation mit JWT erfolgreich**
- **Serviceaufträge können erstellt, bearbeitet & gelöscht werden**
- **OpenAPI-Dokumentation ist verfügbar**

**Verbesserungsmöglichkeiten**

- **Frontend-Erweiterung:** Web-Oberfläche zur Interaktion mit der API
- **Automatische Tests hinzufügen**
- **Erweiterte Fehlerbehandlung & Logging**

---

## 7. Zeitplan

| Datum | Aufgabe |
| --- | --- |
| **Tag 1** | Planung und Architektur-Festlegung |
| **Tag 2-3** | Backend-Projekt aufsetzen, Datenbankstruktur definieren |
| **Tag 4-5** | API-Endpunkte implementieren (CRUD für Serviceaufträge) |
| **Tag 6** | Authentifizierung & Autorisierung implementieren (JWT) |
| **Tag 7** | Swagger-Dokumentation hinzufügen und manuelle Tests durchführen |
| **Tag 8** | Fehlerbehebung und finale Tests mit Postman |
| **Tag 9** | Bereitstellung und Abschlussdokumentation |

---

## Fazit

Das Projekt wurde erfolgreich umgesetzt. Die API ist voll funktionsfähig und erfüllt alle Anforderungen. Verbesserungsmöglichkeiten bestehen in der Einbindung von automatisierten Tests und einer erweiterten Fehlerbehandlung.

---