



13.11.2022

# Dokumentation: Ski-Service Management

Modul 295 –Backend für Applikationen realisieren



David Mangold  
IBZ

## Inhaltsverzeichnis

Einleitung.....	2
Verwendet Tools: .....	2
Dokumentation nach dem IPERKA-Modell.....	2
Informieren.....	2
Planen.....	3
Entscheiden .....	4
Realisieren .....	5
Kontrollieren.....	8
Auswerten .....	8
Fazit .....	8
Testprotokoll .....	9

## Abbildungsverzeichnis

Abbildung 1 GANT und PSP .....	3
Abbildung 2 Einblick in das Models Ordner .....	4
Abbildung 3 Einblick in das Attribut Ordner .....	5
Abbildung 4 Überblick vom Explorer im Visual Studio .....	5
Abbildung 5: Übersicht des Interface .....	6
Abbildung 6: Zuweisung der Interfaces .....	6

## Tabellenverzeichnis

Tabelle 1: Testprotokoll .....	9
--------------------------------	---

## Abkürzungen

Abkürzung	Beschreibung
API	Application Programming Interface
ASP	Active Server Pages
MVC	Model View Control
ORM	Object-Relational Mapping
DB	Danten Bank
JWT	JSON Web Token
SQL	„Structured Query Language

## Einleitung

Die Projektarbeit «Ski-Service Management» lehnt sich an das vorherige Projekt «Ski-Service Anmeldung» aus dem Modul 294. Nun zielt dieses Projekt, «Ski-Service Management», darauf ab, eine API zu erstellen. Diese API soll als Auftragsmanagement für die Mitarbeiter dienen, um die anstehenden Aufträge der Kunden zu verwalten, und ist somit eine Schnittstelle zwischen der Webseite (welche die Kunden verwenden) und Datenbank.

Dieses Projekt sollte folgende Punkte beinhalten:

Das Auftragsmanagement muss folgende Funktion zu Verfügung stellen:

- Login mit Benutzername und Passwort
- Anstehende Serviceaufträge anzeigen
- Bestehende Serviceaufträge mutieren.
- Auftrag löschen (ggf. bei Stornierung)

Das Projekt wurde von David Mangold realisiert und dokumentiert. Die Dokumentation sowie die Umsetzung des Ski-Service Management basieren auf dem IPERKA-Modell. Das Vorgehen wurde phasenweise dokumentiert.

## Verwendet Tools:

Als Quelltext-Editor habe ich Visual Studio Code verwendet, in welchen ich das ASP.NET MVC Webframework angewendet habe. Für die Verbindung mit der Datenbank und das ORM habe ich Entity Framework angewendet. Für das Entwerfen, Erstellen, Testen und Iterieren der APIs wurde POSTMAN verwendet. Um das Projekt zu verwalten wurde ein GitHub Repositorien erstellt und tägliche Änderung am Projekt hochgeladen.

## Dokumentation nach dem IPERKA-Modell

Die Dokumentation erfolgt unterteilt nach den sechs Phasen des IPERKA-Modells.

### Informieren

Als Erstes habe ich den Arbeitsauftrag gründlich durchgelesen und habe vorhandene Unklarheiten notiert, welche ich jeweils dann, in der Schule, mit Herrn Müller besprechen konnte. Die Unklarheiten waren unter anderem:

- Welchen Art von ORM sollte man verwenden ? DB First oder Code First ?
- Welche DB sollte man verwenden? Ist SQL in Ordnung?
- Endpoint müssen Protokolliert sein mit logger?
- Wie soll mit Swagger dokumentiert werden ?
- Wird mit Remove Methode die Id auch gelöscht?

Nachdem die vorhandenen Unklarheiten geklärt waren, inspizierte ich mich noch zusätzlich im Internet, um die im Arbeitsauftrag beschriebenen Schritte zur ermöglichen.

Folgendes habe ich noch recherchiert:

- Wie kann ich eine API Key in meinem Projekt implementieren?
- Wie fügt man ein neuen User in der SQL DB hinzu, sodass der User einer eigene DBbenutzer Zugang hat?
- Wie füge ich ein Middleware ein? Und wofür brauche ich diese?

Weiter haben wir in diesem Modul verschiedene Tools/ Möglichkeiten gelernt, die wir für die Umsetzung des Projekts benutzen konnten, unter anderem Postmann.

Ich habe mich zusätzlich von verschiedenen Webseiten inspirieren lassen und habe dadurch Ideen für dieses Projekt gesammelt. Wo ich dann darauf die Möglichkeit hatte anfangen mein Projekt konkret zu planen.

### Planen

Als Erstes habe ich in der Planungsphase festgelegt, wann ich welchen Teil des Projekts mache und wie lange ich dafür rechne. Dafür habe ich mit dem WBS Tool ein PSP und GANT erstellt.

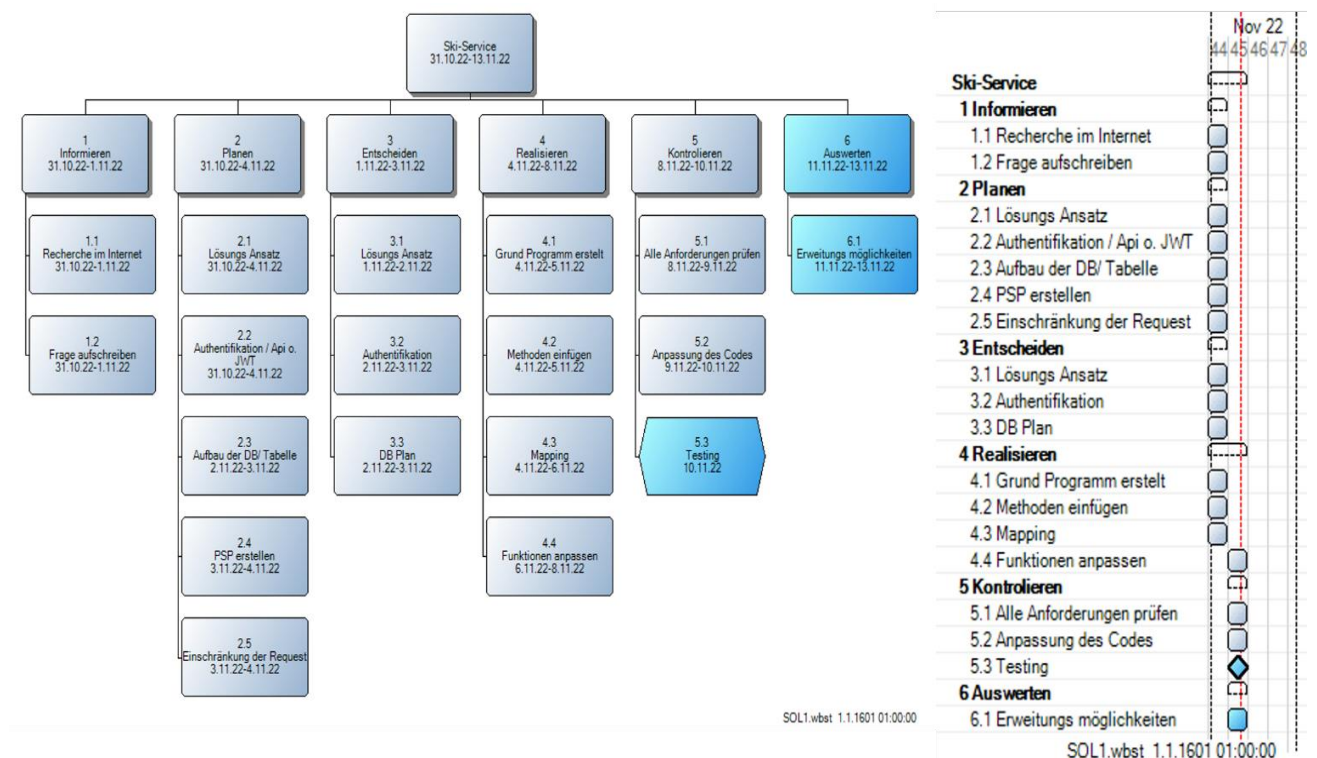


Abbildung 1 GANT und PSP

In einem zweiten Schritt habe ich mir Gedanken gemacht, welche Ansätze für dieses Projekt am geeignetsten ist.

Zum Beispiel:

- welche Lösung Ansatz möchte ich machen? Code First oder DB First?
- Aufbau der DB: wie viel Tabelle braucht es? Und wie sind die Relationen?
- Soll ich für die Authentifikation API Key oder JWT verwenden?
- Wie soll die Request einschränken, das nur Mitarbeiter die Möglichkeit haben alle Request durchzuführen?

Um alle diese Fragen zu beantworten, musste ich auf den letzten Modultag abwarten, da wir am letzten Tag die notwendigen Informationen bekommen haben.

### Entscheiden

Nach dem letzten Tag von diesem Modul habe ich in der Entscheidungsphase die finale Auswahl über die Ansätze getroffen, welche ich in der Planung bereits vorgesehen haben.

Ich habe für das Mapping mich für das Codefirst Ansatz entschieden, da es mir die Möglichkeit gegeben hat die Tabellen in der Datenbank automatisch in der C# Programmiersprache zu generieren. Dies hat mir einen besseren Überblick gegeben, was eingefügt wurde.

Bezüglich die Tabelle in der Datenbank habe ich mich für 5 Tabelle entschieden. 1 für Client (welches alle Attribut erhalten hat, die verlangt wurde). Eine weitere Tabelle für Facility. In dieser Tabelle wurden Angebote von Hand eingetragen und jedes Angebot hat eine ID bekommen. Eine Tabelle mit den Mitarbeitern (diese besitzt Name und die dazugehörige API Key). Eine Tabelle mit Priority (welche Priorität und die ID enthält). Und die fünfte Tabelle ist die Statustabelle (diese besitzt den Status, Name und eine ID).

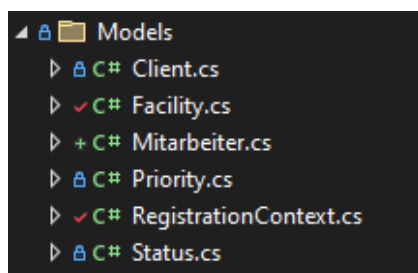


Abbildung 2 Einblick in das Models Ordner

Für die Authentifikation habe ich mich für die API Key entschieden. Der Grund dafür war, dass es leicht ist zum Implementieren und gewährt eine ausreichende Sicherheit. Der Nachteil ist hier, dass man solche Request über ein HTTPS durchführen soll.

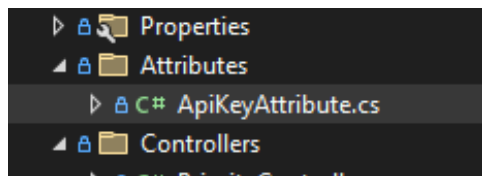


Abbildung 3 Einblick in das Attribut Ordner

Die Einschränkung der Request läuft über eine Attribut Classen ab. Die Klasse wird über jedes Request als Attribut eingefügt, um das API Key abzurufen. Dies wird in der Realisierungsphase genauer erklärt.

## Realisieren

In dieser Phase begann ich mit der Erstellung der API, dazu nutze ich Visual Studio. Und SQL für die Datenbank. Im Visual Studio musste ich für dieses Projekt zuerst noch folgende Nuggets installieren:

- *MicrosoftEntityFrameworkCore*
- *MicrosoftEntityFrameworkCore SqlServer*
- *MicrosoftEntityFrameworkCore.Tools*
- *Newtonsoft.Json*
- *Serilog*
- *Serilog.AspNetCore*

Als Erstes habe ich mein Projekt so aufgebaut das ich ein Grundgerüst bekomme. Das heisst ich habe alle Klasse und Controller wie Services erstellt und anschliessend die Models erstellt (welche als Tabelle in der DB migriert werden)

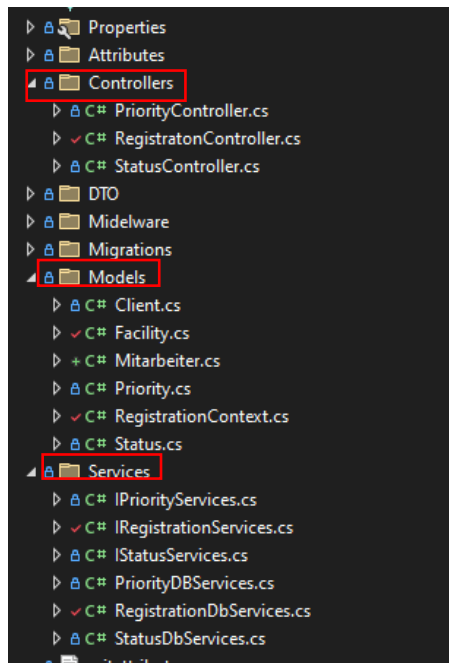


Abbildung 4 Überblick vom Explorer im Visual Studio

In einem nächsten Schritt habe ich mich auf die Methoden konzentriert, welche ich in den Services eingebaut habe. Die Methoden sollen von den jeweiligen Request aufgerufen werden können. Es handelt sich um folgende Methoden zum Beispiel für die Registrierung Services:

```
public interface IRegistrationServices
{
    IEnumerable<ClientDTO> GetAll();
    ClientDTO? Get(int id);
    void Add(ClientDTO registration);
    public void Delete(int id);
    public void Update(ClientDTO registration);
    //void Add(ClientDTO newClient);
}
```

Abbildung 5: Übersicht des Interface

Diese werden im RegistrationDbServices aufgerufen und die Logging eingebaut. Dies habe ich für vier weitere Services gemacht.

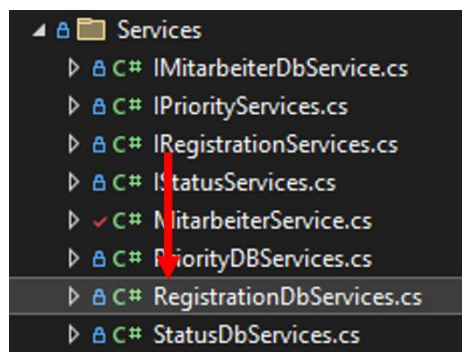


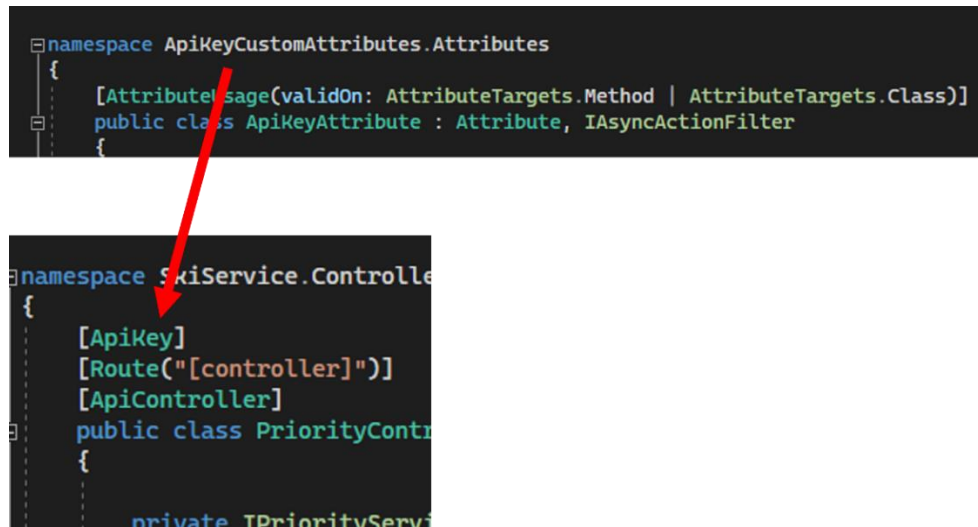
Abbildung 6: Zuweisung der Interfaces

Nach der Implementierung der Methode habe ich mich auf das DPI konzentriert, in dem ich Interfaces in meine Services implementiert habe sodass die Methoden im Controller aufrufbar sind.

```
public RegistrationController(IRegistrationServices registration, ILogger<RegistrationController> logger)
{
    _registrationService = registration;
    _logger = logger;
}
```

Abbildung 7: Anwendung des DPI

Als nächstes habe ich mir mit der Authentifikation auseinandergesetzt und das API Key implementiert. Zuerst habe ich gedacht ich mache es in einer Middleware, aber danach wurde mir bewusst, dass ich die einzelnen Requests einschränken muss. Also habe ich eine Attribut Klasse erstellt, welche in der Methode das API Key verlangt. Und als Target die Methoden und Klassen angewendet.

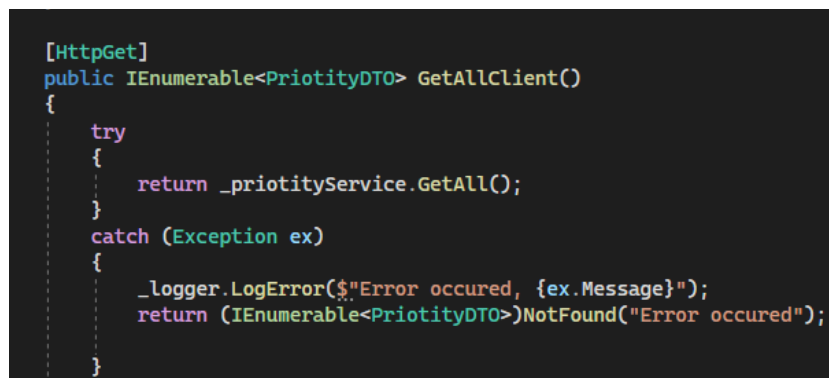


```
namespace ApiKeyCustomAttributes.Attributes
{
    [AttributeUsage(validOn: AttributeTargets.Method | AttributeTargets.Class)]
    public class ApiKeyAttribute : Attribute, IAsyncActionFilter
    {
    }
}

namespace SkiService.Controllers
{
    [ApiKey]
    [Route("[controller]")]
    [ApiController]
    public class PriorityController
    {
        private IPriorityService
    }
}
```

Abbildung 8: Attribut Übergabe

Als ich die Authentifikation fertig hatte, wurde mir bewusst, die Fehlermeldung noch nicht gemacht zu haben. Also habe ich auf jeder Methode ein Try Catch Method über gemacht. So dass wenn es im Catch geht, soll die Fehlermeldung in eine Log File gespeichert werden.



```
[HttpGet]
public IEnumerable<PriorityDTO> GetAllClient()
{
    try
    {
        return _priorityService.GetAll();
    }
    catch (Exception ex)
    {
        _logger.LogError($"Error occurred, {ex.Message}");
        return (IEnumerable<PriorityDTO>)NotFound("Error occurred");
    }
}
```

Abbildung 9: Try Catch Methode

Nachdem ich alle Anforderungen, die in diesem Projekt gefragt waren, gemacht habe, habe ich mich auf die Zusatzanforderung konzentriert. Leider ist mir nur 2 davon gelungen. Und zwar, erstens dass ein Mitarbeiter ein Kommentar hinterlegen kann. Zweitens, dass ein Auftrag mit sämtlichen Datenfelder geändert werden.

Diese Schritte der Realisierung wurde immer mit Postman getestet und überprüft, ob sie funktionieren. Und somit komme ich zu der Kontrolle Phase.



## Kontrollieren

Diese Phase nahm sicherlich am meisten Zeit in Anspruch, da sie das Testen Programms beinhaltete. Es traten fortlaufend Fehler auf, die ich korrigieren mussten. Um die Probleme lokalisieren zu können, benutze ich den Debugger. Wie auch Postman welche mir jeweils angezeigt hat, welche Fehlermeldung aufgetreten ist.

Durch das Auftreten von Fehlern entstanden wiederholt neue Ideen, welche ich implementieren wollte. Daher bin ich wieder zurück in die Entscheidungsphase und überlegte mir, wie ich die neuen Ideen umsetzen kann. Anschliessend hab ich diese in einer erneuten Realisierungsphase umgesetzt. Diesen Vorgang wiederholten ich, bis die API funktionsfähig und zufriedenstellend war.

Zum Schluss habe ich, mit Hilfe unserm Testprotokoll, das API geprüft, ob alle Anforderungen erfüllt sind.

## Auswerten

Nach der Fertigstellung der API habe ich mir Gedanken darüber gemacht, was in diesem Projekt bereits gut funktioniert hat und was ich für ein zukünftiges Projekt verbessern können. Ich bin zum Schluss gekommen, diese Arbeit viel Geduld und viel Wissen verlangt hat.

Das Einzige, was ich in Zukunft verbessern werden, ist das Dokumentieren. Beim Dokumentieren hätte ich bereits zu Beginn des Projekts eine Vorlage erstellen können, um meine Arbeit gleich bei Beginn zu dokumentieren.

In einem weiteren Schritt könnte man diese API auf eine richtige Webseite einbauen, welche ein Client die http POST Request aufrufen kann, in dem der User in einem Dialog die Angabe eingeben kann und die Daten in der DB übertragen werden.

## Fazit

Mir hat dieses Projekt sehr viel Spass bereitet und ich haben dabei enorm viel gelernt. Ich bin begeistert vom Schreiben von C# Programme und die jeweiligen Erfolgserlebnisse, welche ich verspürten, sobald der Code funktioniert.

Gerne hätte ich noch mehr Zeit, um diese Arbeit noch verbessern und komplett vervollständigen und alle zusätzlichen Anforderungen erfüllen. Wie zum Beispiel das eingeloggte Mitarbeiter können, ein gesperrtes Login zurücksetzen oder auch: Gelöschte Aufträge werden nicht aus der Datenbank entfernt, sondern nur als gelöscht markiert.

Ein weiterer Punkt ist, dass es für mich eine grosse Herausforderung war, da wir vieles in kurzer Zeit lernen und umsetzen mussten. Dabei war für mich der Umfang dieses Projektes die grösste Herausforderung. Trotz allem macht C# immer noch Spass! Und ich werde sicherlich in Zukunft weiter an diesem Projekt daran arbeiten.

## Testprotokoll

ID	Was ich Testen	Wie ich testen	Erwartetes Ergebnis	Tatsächliches Ergebnis	Testergebnis	Wurde es gelöst
1	Loggin File	Falsche Eingabe im Body	Fehlermeldung wird im File gespeichert	Fehlermeldung wurde immer im File gespeichert	Fehlerfrei	-
2	Request	Alle Arten von Request gemacht	Alle Request funktionieren	Die Request haben nicht immer Funktionier	Fehlerhaft	JA
3	Andere Attribut im body eingetragen	Im Body vom request ein zusätzliches Attribut mitgegeben	Fehlermeldung	Fehlermeldung	Fehlerfrei	-
4	Api Key	Request senden mit und ohne API Kay	Bei Key Anforderung das keine Daten gezeigt werden	Daten wurden trotz Api Key gezeigt	Fehlerhaft	JA
5	Url im Postmann	Eine falsche URL eingeben	Error Meldung	404Not Found	Fehlerfrei	-

Tabelle 1: Testprotokoll