Ski-Service Management

Projektdokumentation nach IPERKA

IBZ - Basel

Fokko Vos

Inhaltsverzeichnis

[1 Informieren 2](#_Toc150930879)

[1.1 Ausgangsituation 2](#_Toc150930880)

[1.2 Anforderungsanalyse 2](#_Toc150930881)

[1.2.1 Funktionale Anforderungen 2](#_Toc150930882)

[1.2.2 Nicht-Funktionale Anforderungen 2](#_Toc150930883)

[1.3 Technische Anforderungen 2](#_Toc150930884)

[2 Planen 3](#_Toc150930885)

[2.1 Zeitplanung 3](#_Toc150930886)

[2.2 Datenbankentwurf 4](#_Toc150930887)

# Informieren

## Ausgangsituation

**Unternehmen**Jetstream-Service, ein KMU, spezialisiert auf Skiservicearbeiten in der Wintersaison.

**Ziel**Das Unternehmen möchte die interne Verwaltung von Ski-Service-Aufträgen digitalisieren, um Effizienz und Kundenzufriedenheit zu steigern. Aktuell ist noch keinerlei Backend Technologie in Verwendung.

**Integration**Das Jet-Stream Frontend habe ich damals nicht gemacht da ich einen Sonderauftrag aufgrund meiner Vorerfahrung bearbeitet habe. Ich werde für die Integration die Seite von Bobby und Mahir als Grundlage nehmen.

## Anforderungsanalyse

### Funktionale Anforderungen

* Request Protokollierung (Serilog)
* Übersicht der ausstehenden Service-Aufträge
* Login für Änderungen an einem Auftrag (Status, Notiz, Löschen)

### Nicht-Funktionale Anforderungen

* Starke Authentifizierung und Autorisierung.
* Schnelle und effiziente Verarbeitung von Anfragen.
* Einfache und intuitive Bedienung für Mitarbeiter.

## Technische Anforderungen

**Technologie**Backend Entwicklung mit ASP.NET Web API

**Datenbankanforderungen**Robustes Datenbank Design für die Verwaltung von Auftrags- und Kundendaten in der 3NF

**Integration**Kompatibilität und reibungslose Integration mit der bestehenden Infrastruktur.

# Planen

## Zeitplanung

|  |  |  |  |
| --- | --- | --- | --- |
| **Nr.** | **Beschreibung** | **SOLL-Zeit** | **IST-Zeit** |
| **1** | **Informieren** | **3** | **4** |
| 1.1 | Situationsanalyse | 2 | 3 |
| 1.2 | Tool-Installation (MSSQL & MSSMS) | 1 | 0.5 |
| **2** | **Planen** | **6.5** |  |
| 2.1 | Erstellung eines Zeitplans | 1 | 1.5 |
| 2.2 | Entwurf des Datenbankdesigns | 2 | 2 |
| 2.3 | Entwurf der Systemarchitektur | 1 | 1 |
| 2.4 | Spezifizierung der API-Endpunkte | 1 | 1.5 |
| 2.5 | Mockup für die Verwaltung-Oberfläche | 1.5 | 2.5 |
| **3** | **Entscheiden** | **1** |  |
| 3.1 | Test-Strategie definieren | 0.5 | 0.5 |
| 3.2 | Code- oder Database-First entscheiden | 0.5 | 0.5 |
| **4** | **Realisieren** | **28.5** |  |
| 4.1 | Grundlegendes Projekt erstellen | 2 | 1 |
| 4.2 | Git-Repository aufsetzen | 0.5 | 0 |
| 4.3 | Implementierung der Datenbank | 1 | 1 |
| 4.4 | Entwicklung des Backend | 15 | 12 |
| 4.5 | Entwicklung & Integration des Frontends | 10 |  |
| **5** | **Kontrollieren** | **3** |  |
| 5.1 | Test-Strategie ausführen | 1 |  |
| 5.2 | Anforderungen mit dem Produkt abgleichen | 2 |  |
| **6** | **Auswerten** | **4.5** |  |
| 6.1 | Finalisierung der Dokumentation | 2 |  |
| 6.2 | Lessons-Learned identifizieren | 1 |  |
| 6.3 | Abschlusspräsentation vor der Klasse | 2 |  |
| 6.4 | Übergabe an Herrn Müller | 0.5 |  |
| **Gesamt** | | **46.5** |  |

- PSP Zeitplanung

## Datenbankentwurf

1 - Datenbankentwurf

Um eine hohe Skalierbarkeit und Performance zu gewährleisten, werden zentrale Werte wie Services, Prioritäten und Status in dedizierten Tabellen verwaltet. Diese Struktur ermöglicht nicht nur eine effiziente Indizierung und Suchfunktion, sondern auch eine problemlose Erweiterbarkeit bei Bedarf. Jedes Attribut innerhalb der Tabellen wird mit der NOT NULL-Bedingung versehen, wenn nicht anders spezifiziert, um Datenkonsistenz zu gewährleisten.

Die Einzigartigkeit der Nutzernamen wird durch einen UNIQUE CONSTRAINT garantiert, der die Integrität des nutzernamenbasierten Login-Systems sicherstellt. Dieser Ansatz verhindert doppelte Einträge und unterstützt die Authentifizierungseffizienz. Für eine Feinere Kontrolle über den Zugriff hat jeder Nutzer eine Rolle wo «Mitarbeiter» der Standard ist und «SuperAdmin» für den Geschäftsführer um seine Mitarbeiter zu verwalten

In der Auftragsverwaltung kann jeder Auftrag entweder keinen oder genau einen Nutzer haben, was eine optionale 1:1-Beziehung von Aufträgen zu Nutzern bedeutet. Auf der anderen Seite kann ein Nutzer für mehrere Aufträge zuständig sein, was eine 1:n-Beziehung von Nutzern zu Aufträgen darstellt. Zudem ist es obligatorisch, dass jeder Auftrag einen Service, eine Priorität und einen Status zugewiesen bekommt. Es ist dabei möglich, dass verschiedene Aufträge denselben Service, dieselbe Priorität und denselben Status teilen können.

## Systemarchitekturentwurf

2- Systemarchitekturentwurf

Das System ist auf eine Drei-Schichten-Architektur ausgelegt, die eine klare Trennung zwischen Benutzeroberfläche, Geschäftslogik und Datenpersistenz ermöglicht. Diese Strukturierung erlaubt es, jede Schicht unabhängig von den anderen zu entwickeln und zu warten, was eine effiziente und zielgerichtete Entwicklung fördert.

**Web Browser (Frontend UI)**  
Die Präsentationsschicht ist als Webanwendung realisiert, die im Browser des Benutzers läuft. Sie ist verantwortlich für die Darstellung der Benutzeroberfläche und die Interaktion mit dem Nutzer.

**Web Server (ASP.NET API)**  
Die Geschäftslogikschicht, implementiert durch eine ASP.NET-basierte API, kümmert sich um die Verarbeitung von Benutzeranfragen, das Ausführen von Geschäftsregeln sowie die Datenkommunikation.

**Datenbank (MSSQL)**  
Die Datenhaltungsschicht wird durch Microsoft SQL Server, ein leistungsfähiges relationales Datenbankmanagementsystem, implementiert. Die Datenbank speichert alle erforderlichen Daten und stellt sie auf Anfrage bereit. Ihre Struktur ist optimiert für schnelle Abfragen und Transaktionsintegrität

Die Kommunikation zwischen den Schichten erfolgt über HTTPS, wobei JSON als Datenaustauschformat verwendet wird, um die Interoperabilität und das leichte Parsing der Daten zu gewährleisten. Diese Architektur bietet eine solide Basis für die Realisierung unseres Projekts und unterstützt dessen Wachstum und Anpassungsfähigkeit an zukünftige Anforderungen.

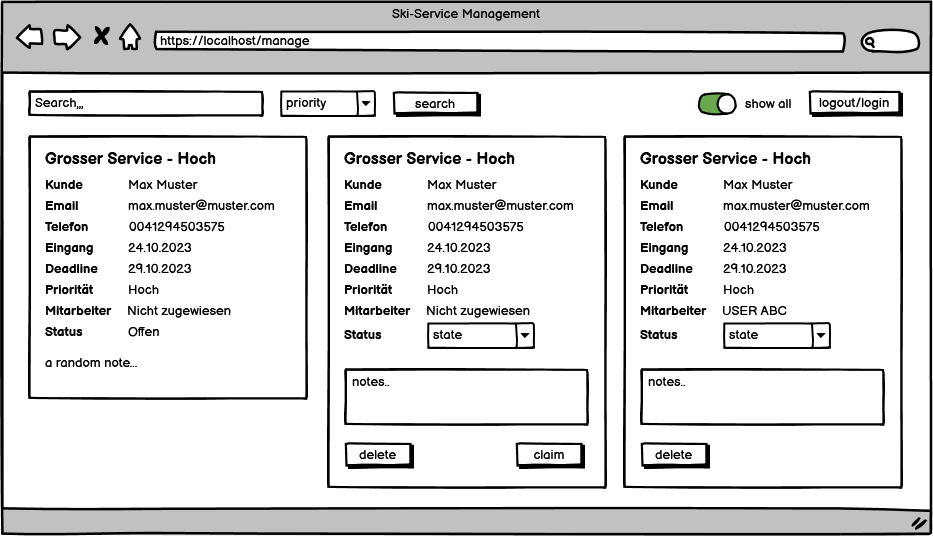
## API-Endpunkte

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Endpunkt** | **Methode** | **Beschreibung** | **AUTH** | **Rolle** |
| /users | GET | Abfrage aller Nutzer | X | superadmin |
| /users | POST | Erstellung eines neuen Nutzers | X | superadmin |
| /users/login | POST | Login Endpunkt |  | - |
| /users/me | GET |  |  |  |
| /users/<id> | GET | Nutzer abfragen | X | superadmin, owner |
| /users/<id> | PATCH | Nutzer ändern | X | superadmin |
| /users/<id> | DELETE | Nutzer löschen | X | superadmin, owner |
| /users/<id>/orders | GET | Alle zugewiesenen Aufträge für diesen Nutzer |  |  |
| /orders | GET | Abfrage aller Aufträge |  | - |
| /orders | POST | Neuen Auftrag erstellen |  | - |
| /orders/<id> | GET | Auftrag abfragen |  | - |
| /orders/<id> | PATCH | Auftrag ändern | X | mitarbeiter |
| /orders/<id> | DELETE | Auftrag löschen | X | mitarbeiter |
| /services | GET | Abfrage aller Services |  |  |
| /services/<id> | GET | Service abfragen |  |  |
| /states | GET | Abfrage aller Stati |  |  |
| /states/<id> | GET | Status abfragen |  |  |
| /priority | GET | Abfrage aller Prioritäten |  |  |
| /priority/<id> | GET | Priorität abfragen |  |  |

- API Endpunkte

Dies ist eine Grundlegende Initial-Definition der API-Endpunkte, kann sich bei der Entwicklung leicht ändern.

## Mockup



In der Entwicklung des Self-Service-Managementsystems wurde ein besonderes Augenmerk auf eine intuitive Nutzerführung gelegt, um eine optimale User Experience zu gewährleisten. Das Design orientiert sich an bewährten Panel-Layouts, welche die Benutzerfreundlichkeit und Zugänglichkeit in den Vordergrund stellen.

Das Wireframe präsentiert drei Kernzustände der Benutzerinteraktion:

**Unauthentifizierter Zugriff**  
Die erste Karte zeigt die Ansicht für Nutzer, die nicht eingeloggt sind. Hier wird ein klar strukturiertes Layout verwendet, um sofortige Einblicke zu ermöglichen, ohne dass vertrauliche Informationen preisgegeben werden.

**Authentifizierter Zugriff ohne Claim**  
In der zweiten Karte wird die Ansicht dargestellt, die ein eingeloggter Nutzer ohne Beanspruchung eines Services sieht. Die Interaktionselemente sind so gestaltet, dass sie auf die Aktionen des Nutzers reagieren.

**Authentifizierter Zugriff mit Claim**  
Die dritte Karte illustriert die Ansicht, nachdem ein Service durch den Nutzer beansprucht wurde. Die Veränderungen im Interface signalisieren deutlich den neuen Status, und der Nutzer erhält visuelles Feedback über die erfolgreiche Übernahme des Services.

Die Gestaltung der visuellen Hierarchie lenkt die Aufmerksamkeit gezielt auf wichtige Informationen wie Fristen und Prioritäten, und interaktive Elemente wie Suchfunktionen und Status-Auswahlmöglichkeiten fördern eine effiziente Aufgabenbearbeitung. Ein konsistentes Design über die verschiedenen Zustände hinweg stellt sicher, dass die Nutzer eine verlässliche und vorhersagbare Erfahrung haben.

Zugänglichkeitsstandards werden berücksichtigt, um das System für Menschen mit unterschiedlichen Fähigkeiten nutzbar zu machen.

# Entscheiden

## Test-Strategie

Für das Projekt habe ich geplant, für die verschiedenen API-Endpoints spezifische Unit Tests zu implementieren. Diese Tests werden sich darauf konzentrieren, die Funktionalität der einzelnen Komponenten der API zu überprüfen. Dabei wird überprüft, ob die Endpoints die erwarteten Ergebnisse liefern, korrekt mit Eingabeparametern umgehen und geeignete Reaktionen auf unterschiedliche Anfrageszenarien zeigen.

Aufgrund des schulischen Projektumfangs werde ich mich auf Unit Tests konzentrieren und komplexere Testformen wie Integrationstests und End-to-End-Tests aussparen. Für die Implementierung der Unit Tests wird das in Visual Studio integrierte Testframework verwendet.

## Code- oder Database-First

Ich habe mich für den Code-First-Ansatz entschieden, da er eine hohe Flexibilität und eine enge Integration mit der Anwendungsentwicklung ermöglicht. Besonders weil keine komplexe vorbestehende Datenbankstruktur vorhanden ist, erlaubt dieser Ansatz ein intuitives Design und eine schrittweise Entwicklung des Datenmodells direkt aus dem Anwendungscode heraus.

# Realisierung