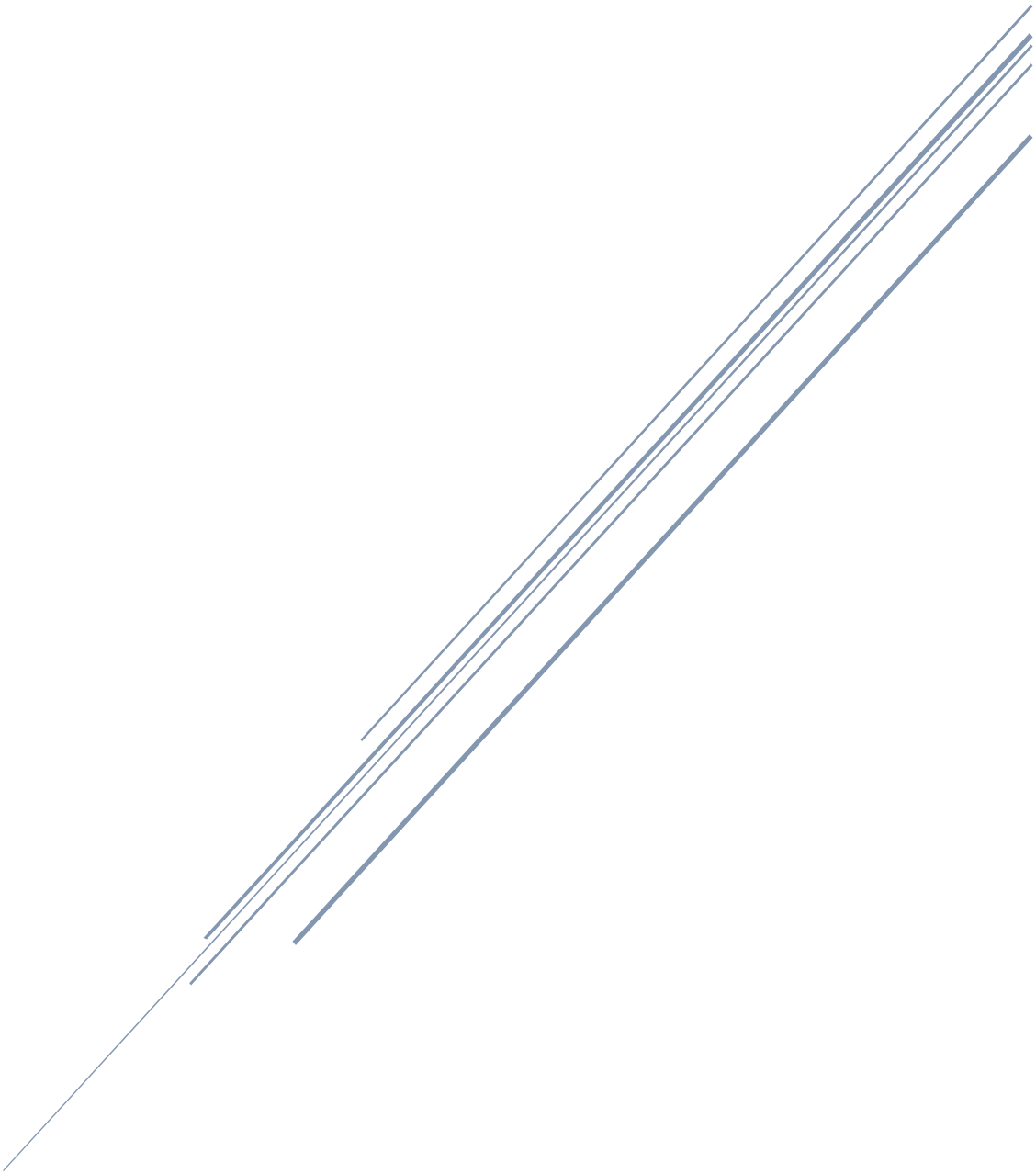
SKI-SERVICE MANAGEMENT

Projektdokumentation nach IPERKA



IBZ - Basel
Fokko Vos

Inhaltsverzeichnis

1 Versionsverlauf

Version	Autor	Datum	Änderung
1.0	Fokko Vos	15.07.2023	Erstellung des Dokumentes und der Planung
1.1	Fokko Vos	15.07.2023	Aktualisierung des Zeitplans
1.2	Fokko Vos	16.07.2023	Aktualisierung des Zeitplans
1.3	Fokko Vos	16.07.2023	Aktualisierung des Zeitplans
1.4	Fokko Vos	18.07.2023	Aktualisierung des Zeitplans
1.5	Fokko Vos	20.07.2023	Hinzufügen der fehlenden Abschnitte
1.6	Fokko Vos	21.07.2023	Finalisierung der Dokumentation

2 Informieren

2.1 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.

2.2 Anforderungsanalyse

2.2.1 Funktionale Anforderungen

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

2.2.2 Nicht-Funktionale Anforderungen

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

2.3 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.

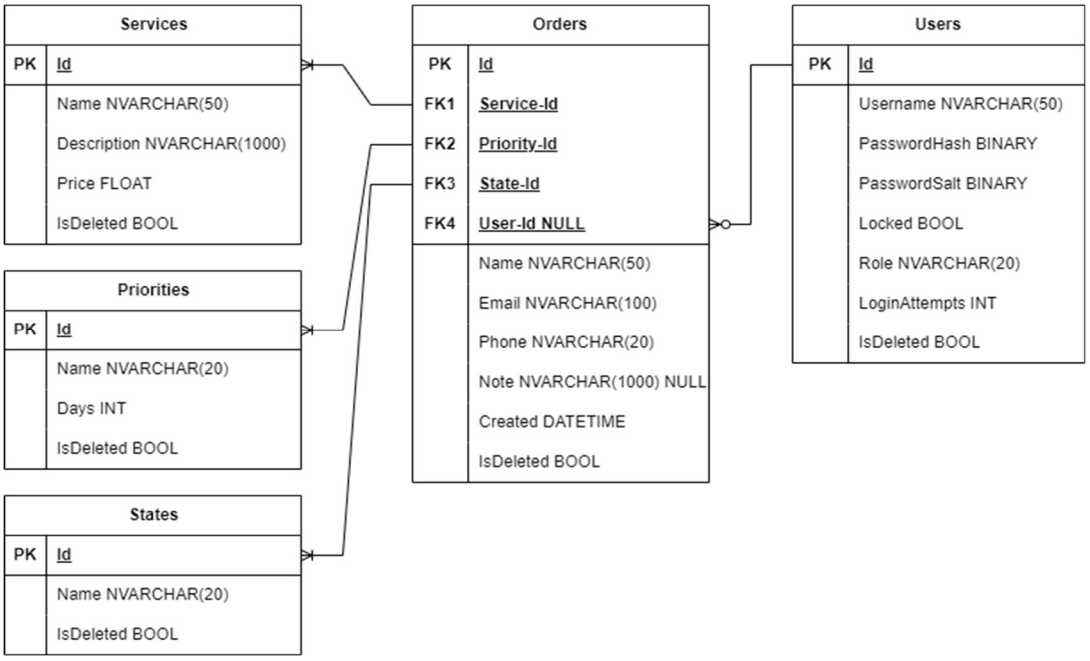
3 Planen

3.1 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	17
4.6	Entwicklung der Tests	5	7
~~4.7~~	~~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**	

1 - PSP Zeitplanung

3.2 Datenbankentwurf



1 - Datenbankentwurf

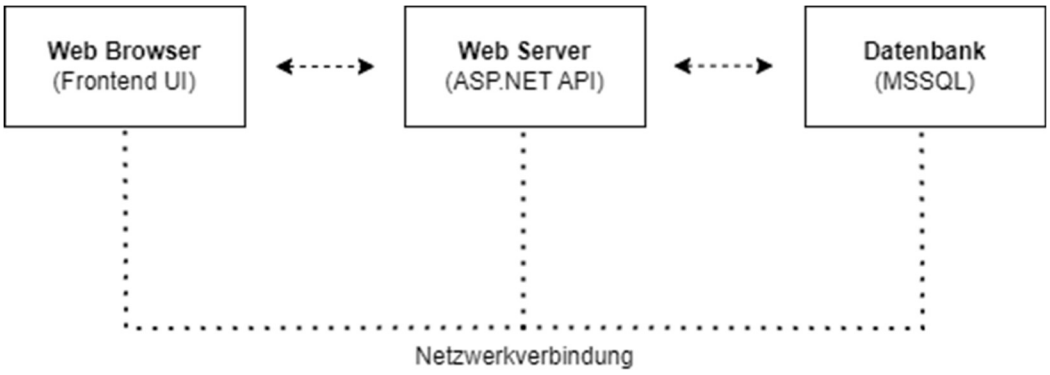
Um eine hohe Skalierbarkeit und Performance zu gewährleisten, werden zentrale Werte wie Services, Prioritäten und Status in dedizierten Tabellen verwaltet. Diese Struktur ermöglicht nicht nur eine effiziente Indizierung und Suchfunktion, sondern auch eine problemlose Erweiterbarkeit bei Bedarf. Jedes 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.

Jede Tabelle in dem Datenbankschema beinhaltet ein IsDeleted-Attribut für die Umsetzung der Soft-Delete-Funktionalität. Diese Eigenschaft ermöglicht es, Einträge als gelöscht zu markieren, ohne sie tatsächlich zu entfernen, was die Datenintegrität für Audits und mögliche Wiederherstellungen sichert.

3.3 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.

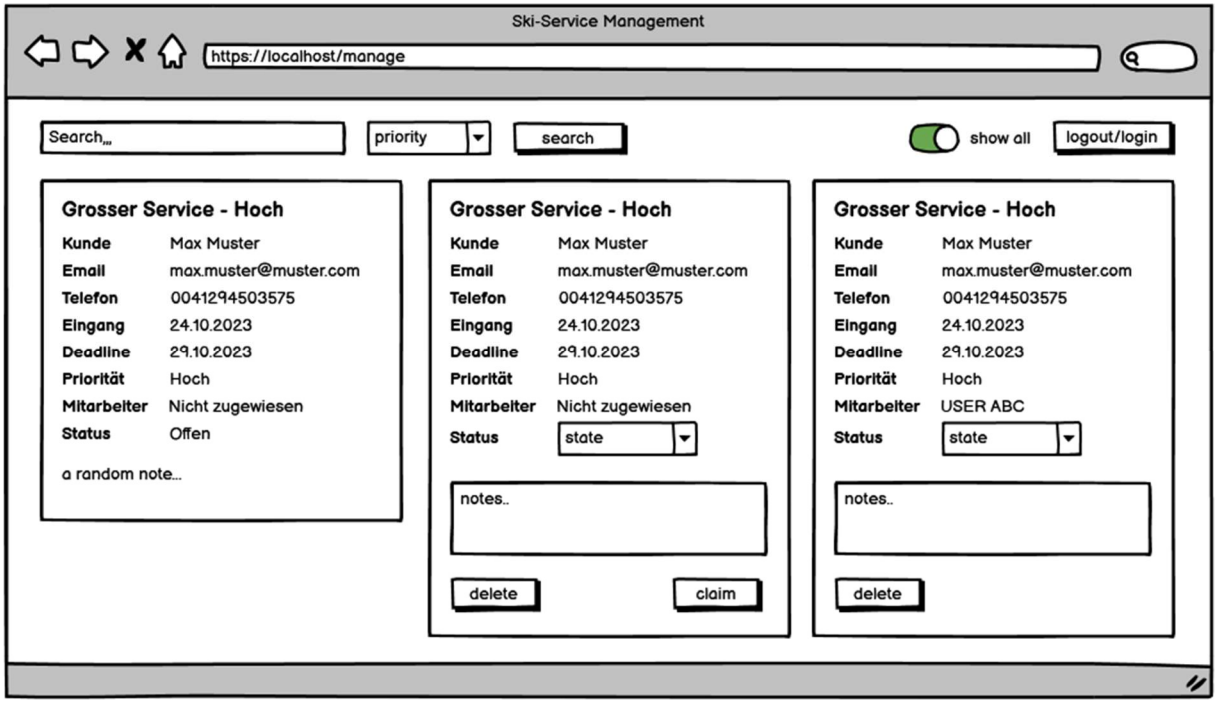
3.4 API-Endpunkte

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	Informationen des eingeloggten Nutzers	X	-
/users/<id>	GET	Nutzer abfragen	X	superadmin
/users/<id>	PATCH	Nutzer ändern	X	superadmin
/users/<id>	DELETE	Nutzer löschen	X	superadmin
/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		

2 - API Endpunkte

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

3.5 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.

4 Entscheiden

4.1 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.

Im Rahmen des schulischen Projekts liegt der Schwerpunkt auf Unit Tests, um die Funktionalität der einzelnen Komponenten zu validieren. Dabei wird das in Visual Studio integrierte Testframework genutzt. Zusätzlich werden grundlegende Integrationstests mit Postman durchgeführt, um die Interaktionen zwischen verschiedenen Komponenten zu überprüfen. Dies bietet einen realistischeren Kontext für die Tests, ohne den Rahmen des Projekts durch umfassende End-to-End-Tests zu überschreiten.

4.2 Code- oder Database-First

Für die Datenbankentwicklung habe ich mich für den Code-First-Ansatz entschieden, der durch die Verwendung von Entity Framework ermöglicht wird. Diese Methode bietet große Flexibilität und eine nahtlose Integration mit dem Entwicklungsprozess der Anwendung. Da keine komplexe, vorbestehende Datenbankstruktur zu berücksichtigen ist, ermöglicht dieser Ansatz ein intuitives Design und die schrittweise Weiterentwicklung des Datenmodells direkt im Anwendungscode. Entity Framework erleichtert dabei die Erstellung und Migration von Datenbankschemata, was die Effizienz und Agilität im Entwicklungsprozess deutlich steigert.

5 Realisierung

5.1 Grundlegendes Projekt erstellen

Ich habe mit Visual Studio ein ASP.NET Web API Projekt für das Ski-Service-Management-System erstellt. Dieses Projekt ist nicht nur mit grundlegenden Bausteinen ausgestattet, sondern auch mit erweiterten Funktionen für eine robuste und sichere Anwendung:

JWT-Authentifizierung
Implementiert im TokenService, um sichere und effiziente Benutzerauthentifizierung und -autorisation zu gewährleisten.
Serilog Logging
Konfiguriert im appsettings für fortgeschrittene Logging-Fähigkeiten, einschließlich der Rotation von Logdateien.
Datenbankanbindung
Verwendung von Entity Framework Core für die Datenbankinteraktion, konfiguriert in Programm.cs und appsettings.
Automapper
Eingesetzt für die effiziente Umwandlung zwischen Datenmodellen und Data Transfer Objects (DTOs), konfiguriert im MappingProfile.
Swagger Integration
Ermöglicht eine interaktive Dokumentation und einfache Testmöglichkeiten der Web API.
Docker-Unterstützung
Bereitstellung von Dockerfiles und Skripten für die Containerisierung der Anwendung und der Datenbank.
Testabdeckung
Einrichtung von Unit-Tests im Verzeichnis SkiServiceAPI.Tests zur Sicherstellung der Codequalität.

5.2 Git-Repository Aufsetzen

Für die Versionskontrolle habe ich GitHub verwendet. Das Initial Repository wurde direkt aus Visual Studio heraus erstellt. Wichtige Dateien und Konfigurationen, die im Repository enthalten sind:

.gitignore
Eine sorgfältig konfigurierte Datei, die sicherstellt, dass nur relevante Dateien und Verzeichnisse im Repository verfolgt werden. Sie schließt unnötige oder vertrauliche Dateien aus, wie z.B. lokale Konfigurationsdateien und Build-Artefakte.
README.md
Eine zentrale Dokumentationsdatei, die eine Übersicht über das Projekt, Installationsanweisungen, Nutzungshinweise und andere wichtige Informationen enthält.
Lizenzdatei
Die LICENSE-Datei definiert, wie andere das Projekt verwenden dürfen. Dies ist wichtig für die Festlegung von Urheberrechten und Nutzungsbedingungen.

5.3 Datenbank Implementierung

In diesem Abschnitt habe ich die Datenbank für das Ski-Service-Management-System implementiert, wobei ich mich auf Robustheit, Effizienz und Skalierbarkeit konzentriert habe:

Datenbanktechnologie und Containerisierung
Verwendung von Microsoft SQL Server, wie im Dockerfile und entrypoint.sh ersichtlich. Die Containerisierung ermöglicht eine einfache und konsistente Bereitstellung der Datenbank.
Initialisierungsskript
Ein init.sql Skript wurde erstellt, um die Datenbank und erforderlichen Benutzer bei der ersten Ausführung zu initialisieren.
Entity Framework Core
Der Einsatz von Entity Framework Core wird durch die Klasse ApplicationDBContext im ApplicationDBContext.cs dargestellt. Dies ermöglicht eine effiziente und sichere Datenbankinteraktion.
Datenbankmodelle
Die Datenbankmodelle, wie User, Order, Priority, Service und State, sind in den entsprechenden Modellklassen definiert. Diese Modelle repräsentieren die Struktur der Datenbanktabellen und deren Beziehungen.
Datenbankmigrationen
Die Verwendung von Migrationen zur Verwaltung von Datenbankschemaänderungen wird durch das Dockerfile.Migrate und das entrypoint.Migration.sh Skript demonstriert. Dies gewährleistet, dass die Datenbankstruktur immer aktuell und mit dem Code synchronisiert ist.
Dateninitialisierung
Standardwerte für Tabellen wie Service, Priority und State werden in ApplicationDBContext definiert.
Seed-User
In der Program.cs wird eine Initialisierungsroutine für Seed-User implementiert. Diese Routine stellt sicher, dass bei der ersten Ausführung der Anwendung Standardbenutzerkonten in der Datenbank vorhanden sind, was für die erste Inbetriebnahme und Tests von großer Bedeutung ist.

5.4 Entwicklung des Backend

In diesem Abschnitt habe ich die Haupt-Logik des Programms geschrieben:

Architektur und Design
In der Entwicklung des Backends meines ASP.NET Web API Projekts habe ich eine Architektur geschaffen, die generische Controller und Services nutzt. Diese ermöglichen eine effiziente und flexible Handhabung verschiedener Entitätstypen. Die Architektur unterstützt die Wiederverwendbarkeit des Codes und erleichtert die Wartung, während sie gleichzeitig eine solide Grundlage für die Sicherheits- und Autorisierungsmechanismen bietet.

API-Entwicklung
Ich habe mich bei der API-Entwicklung streng an REST-Prinzipien gehalten. Durch die Verwendung von AutoMapper, implementiert in MappingProfile.cs, konnte ich eine effiziente Konvertierung zwischen Datenmodellen und DTOs erreichen. Die Controller, wie OrdersController und UsersController, sind konsistent in ihrer Struktur und bieten eine klare und intuitive Schnittstelle für CRUD-Operationen.

Authentifizierung mit JWT
Die Implementierung der JWT-basierten Authentifizierung erfolgte durch eine Kombination von Anpassungen in Program.cs, dem TokenService und dem UserService, sowie durch die Integration in den UsersController. Diese Implementierung stellt eine sichere und effiziente Authentifizierung sicher und ist ein Kernbestandteil der Sicherheitsarchitektur der Anwendung.

Fehlerbehandlung und Logging
Für das Logging habe ich Serilog integriert, um detaillierte und aussagekräftige Log-Informationen zu erfassen. Die selbst entwickelte ExceptionHandlingMiddleware spielt eine zentrale Rolle in der Fehlerbehandlung, indem sie Ausnahmen abfängt und in einer benutzerfreundlichen Form zurückgibt.

Unit-Tests und Qualitätssicherung
Die Unit-Tests, die ich erstellt habe, decken eine Vielzahl von Funktionen und Szenarien ab. Durch automatisierte Tests und regelmäßige Code-Reviews stelle ich sicher, dass die Qualität der Anwendung kontinuierlich überwacht und verbessert wird.

Dokumentation und Swagger
Die Integration von Swagger/OpenAPI in das Projekt ermöglicht eine klare und interaktive Dokumentation der API. Die Response-DTOs sind detailliert definiert, um eine strukturierte und verständliche Rückgabe von Daten zu gewährleisten.

Performance und Skalierung
Ich habe mich auf generische Implementierungen, effiziente Datenbankzugriffe und asynchrone Programmierung konzentriert, um eine hohe Performance und Skalierbarkeit des Backends zu erreichen. Regelmäßiges Performance-Monitoring und gezielte Optimierungen tragen zur langfristigen Leistungsfähigkeit des Systems bei.

5.5 Entwicklung der Tests

In der Testphase unseres ASP.NET Web API Projekts haben wir eine umfassende Teststrategie implementiert, um die Funktionalität und Zuverlässigkeit der Anwendung zu gewährleisten:

Generische Controller-Tests
In GenericControllerTests.cs wurden Tests für CRUD-Operationen durchgeführt, um die Funktionalität der generischen Controller zu überprüfen.
Spezifische Controller-Tests
OrdersControllerTests.cs und UsersControllerTests.cs fokussieren sich auf die Überprüfung der Bestell- und Benutzerverwaltungsfunktionen.
Mock-Daten
Realistische Testdaten wurden verwendet, um eine breite Abdeckung und realitätsnahe Testszenarien zu ermöglichen.
Postman Collection
Die SkiService-Management.postman_collection.json im files-Ordner dient zur Überprüfung der API-Endpunkte und zur Demonstration der Anwendungsfälle.
Automatisierte Tests
Diese gewährleisten eine kontinuierliche Überwachung der Anwendungsqualität und helfen, Regressionen schnell zu identifizieren.

Diese Tests stellen sicher, dass unser System den Anforderungen entspricht und eine solide Basis für zukünftige Erweiterungen bietet.

5.6 Entwicklung & Integration des Frontends

In meinem ASP.NET Web API Projekt habe ich mich entschieden, die Entwicklung eines dedizierten Frontends zu überspringen und stattdessen die Funktionalitäten und das Management des Backends über Swagger und Postman zu demonstrieren und zu testen. Diese Entscheidung basiert auf folgenden Überlegungen:

Projektanforderungen
Das Hauptziel des Projekts liegt in der Entwicklung und Demonstration eines robusten und funktionsreichen Backends. Ein separates Frontend ist für diese Anforderungen nicht notwendig.
Swagger UI
Durch die Integration von Swagger/OpenAPI biete ich eine interaktive Dokumentation und Testumgebung für die API. Dies ermöglicht es mir, alle Endpunkte direkt zu testen und zu dokumentieren, was eine klare und benutzerfreundliche Schnittstelle für die API darstellt.
Postman
Die Verwendung von Postman für das Testen der API-Endpunkte ermöglicht eine effiziente und flexible Testumgebung. Die bereitgestellte Postman Collection im files-Ordner illustriert die Nutzung der API und erleichtert das Verständnis der Funktionalitäten.
Fokus auf Backend-Entwicklung
Durch den Verzicht auf ein separates Frontend konnte ich mich vollständig auf die Entwicklung, Optimierung und Testung des Backends konzentrieren. Dies hat zu einer tieferen und spezialisierteren Arbeit an der Backend-Logik geführt.

6 Kontrollieren

6.1 Test-Strategie ausführen

In dieser Phase des Projekts lag der Fokus auf der Durchführung und Dokumentation von Tests, um die Funktionalität und Zuverlässigkeit der entwickelten API sicherzustellen. Die Teststrategie umfasste sowohl Unit Tests als auch grundlegende Integrationstests.

Unit-Tests
Die Unit Tests zielten darauf ab, die Funktionalität der einzelnen Komponenten der API isoliert zu überprüfen. Dies beinhaltete die Validierung der Kernfunktionalitäten, das korrekte Handling von Eingabeparametern und die angemessene Reaktion auf verschiedene Anfrageszenarien.

Die Ergebnisse der Unit Tests wurden in Visual Studio dokumentiert und sind im beigefügten Testbericht detailliert aufgeführt. Sie zeigen eine hohe Abdeckung und erfolgreiche Validierung der API-Komponenten.

Ort: files/Berichte/test_results.trx

Integrationstests mit Postman
Die Integrationstests dienten dazu, die Interaktionen zwischen verschiedenen Komponenten der API zu überprüfen. Dies bot einen realistischeren Kontext für die Funktionsweise der API.

Die Ergebnisse der Integrationstests bestätigten die korrekte Funktionsweise der API bei der Verarbeitung integrierter Anfragen. Details zu diesen Tests sind ebenfalls im Testbericht enthalten.

Ort: files/Berichte/SkiService-Management.postman_test_run.json

Testbericht
Die Testergebnisse wurden sorgfältig dokumentiert und sind im angehängten Testbericht einsehbar. Dieser Bericht enthält detaillierte Informationen über die durchgeführten Tests, einschließlich der Testfälle, der erzielten Ergebnisse und der festgestellten Probleme.

6.2 Anforderungen mit dem Produkt abgleichen

Bei der detaillierten Überprüfung des Ski-Service-Management-Projekts habe ich festgestellt, dass alle vorgegebenen Anforderungen erfolgreich umgesetzt wurden. Diese umfassende Implementierung deckt sowohl die Kernfunktionalitäten als auch zusätzliche optionale Features ab.

Login und Authentifizierung (A1, AO3)
Ein sicherer Login-Dialog mit Passwortauthentifizierung wurde implementiert, inklusive einer Funktion zur Sperrung des Zugangs nach mehrfachen Fehlversuchen.
Datenbankverwaltung (A2, A3, A4, A8, A9, AO1, AO2, AO7)
Die Datenbank verwaltet effektiv Serviceaufträge und Benutzerinformationen, unterstützt das Abrufen und Sortieren von Aufträgen, ermöglicht das Hinzufügen von Kommentaren und Änderungen an Auftragsdaten, und gewährleistet die Normalisierung sowie eingeschränkte Benutzerzugänge.
Auftragsmanagement (A5, A6, AO4, AO6)
Mitarbeiter können den Status von Aufträgen ändern, Aufträge löschen, personalisierte Auftragslisten einsehen und gesperrte Logins zurücksetzen.
Protokollierung und Dokumentation (A7, A10, A11, A12)
Die API-Endpoints werden protokolliert, die Web-API ist vollständig nach Open-API dokumentiert, das Projekt wird über ein Git-Repository verwaltet, und das gesamte Projektmanagement ist nach IPERKA dokumentiert.

7 Auswerten

7.1 Fazit

In diesem Projekt habe ich wertvolle Erfahrungen und Kenntnisse im Bereich der Softwareentwicklung, insbesondere in C# und ASP.NET Web API, gesammelt. Durch die intensive Auseinandersetzung mit diesen Technologien konnte ich mein Verständnis für die zugrundeliegenden Konzepte und die Funktionsweise moderner Web-APIs vertiefen.

Ein zentraler Aspekt meiner Lernerfahrung war die Arbeit mit generischen Typen in C#. Diese ermöglichten es mir, flexiblere und wiederverwendbare Codekomponenten zu entwickeln, was zu einer effizienteren und wartungsfreundlicheren Codebasis führte. Die Anwendung von generischen Typen hat mein Verständnis für Typsicherheit und Polymorphismus in der Programmierung erheblich erweitert.

Ein weiterer wichtiger Lernbereich war das SOLID-Prinzip. Die praktische Anwendung dieser Designprinzipien hat mir geholfen, die Bedeutung einer sauberen, modularen und erweiterbaren Architektur zu erkennen. Insbesondere die Prinzipien der Einzelverantwortung und der offenen/geschlossenen Architektur waren in meinem Projekt von großer Bedeutung. Sie leiteten mich bei der Entwicklung von Komponenten, die spezifische Aufgaben erfüllen, ohne die Notwendigkeit häufiger Änderungen bei Erweiterungen oder Anpassungen.

Die Interaktion zwischen den verschiedenen Komponenten einer API war ebenfalls ein Schlüsselelement meines Lernprozesses. Die Entwicklung von Endpunkten, die effizient mit der Datenbank interagieren, die Implementierung von Authentifizierungsmechanismen und das Verständnis für die Behandlung von Anfragen und Antworten haben mein Wissen über Backend-Entwicklung erheblich erweitert.

Insgesamt hat dieses Projekt meine Fähigkeiten in der Softwareentwicklung gestärkt und mir wertvolle Einblicke in die Best Practices der Branche gegeben. Es war eine herausfordernde, aber ungemein bereichernde Erfahrung, die meine Begeisterung für die Entwicklung von Webanwendungen weiter gefestigt hat.

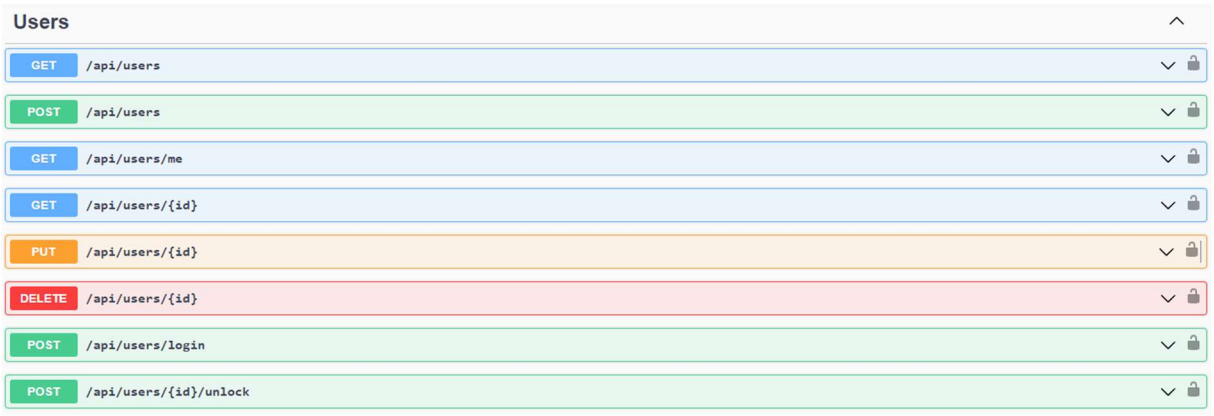
8 Anhänge

8.1 Verwendete NuGet-Pakete

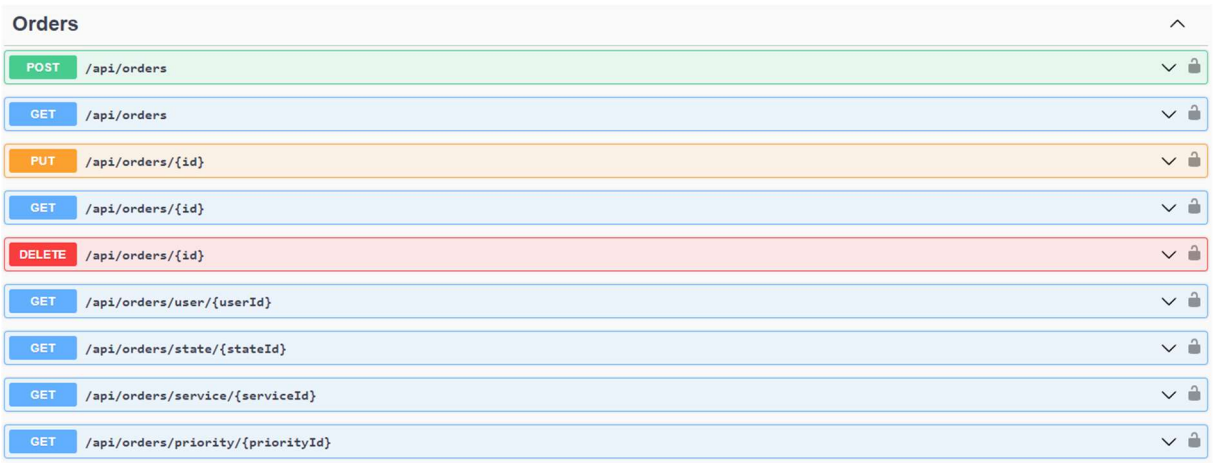
Paket	Version
AutoMapper	12.0.1
AutoMapper.Extensions.Microsoft.DependencyInjection	12.0.1
Microsoft.AspNetCore.Authentication.JwtBearer	7.0.13
Microsoft.AspNetCore.OpenApi	7.0.13
Microsoft.EntityFrameworkCore	7.0.13
Microsoft.EntityFrameworkCore.Design	7.0.13
Microsoft.EntityFrameworkCore.Proxies	7.0.13
Microsoft.EntityFrameworkCore.SqlServer	7.0.13
Microsoft.EntityFrameworkCore.Tools	7.0.13
Microsoft.IdentityModel.Tokens	7.0.3
Serilog.AspNetCore	7.0.0
Serilog.Settings.Configuration	7.0.1
Serilog.Sinks.File	5.0.0
Swashbuckle.AspNetCore	6.5.0
System.IdentityModel.Tokens.Jwt	7.0.3

3 - NuGet Paket Versionen

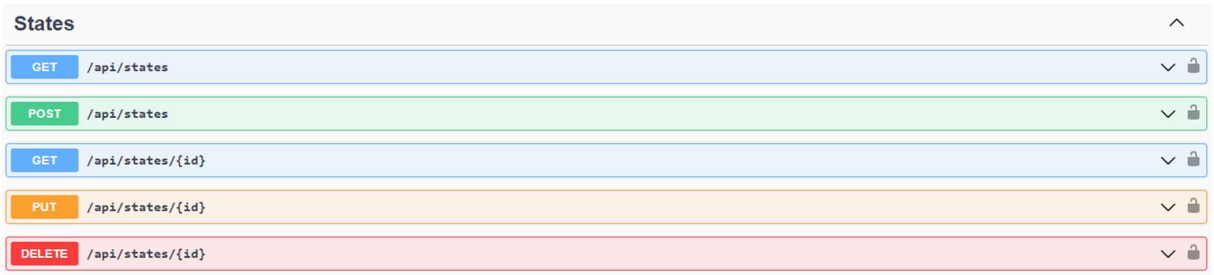
8.2 Swagger



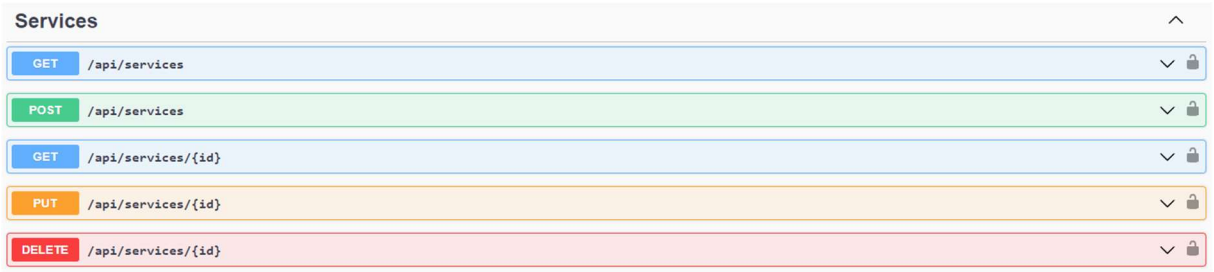
3 - Swagger UI - /api/users



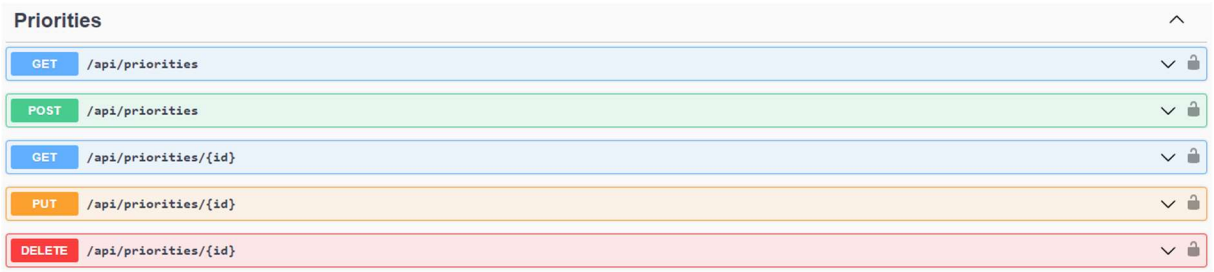
4 - Swagger UI - /api/orders



5 - Swagger UI - /api/states

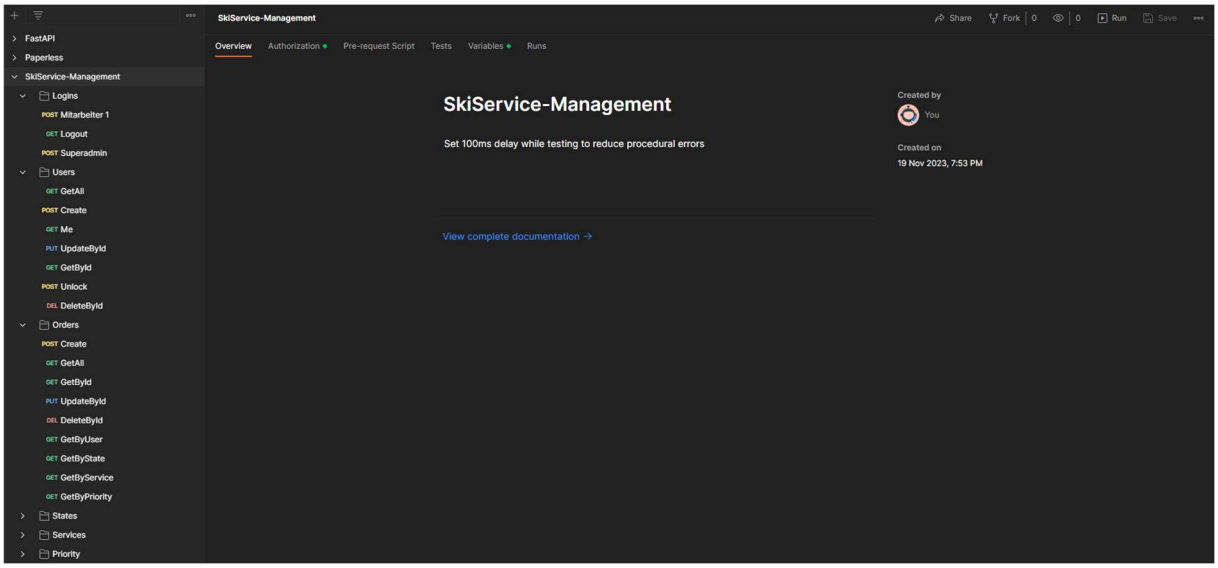


6 - Swagger UI - /api/services



7 - Swagger UI - /api/priorities

8.3 Postman



8 - Postman Collection - Übersicht

8.4 Glossar

Abkürzung	Ausgeschrieben	Erklärung
KMU	Kleine und mittlere Unternehmen	Bezieht sich auf Unternehmen, die aufgrund ihrer Größe bestimmte Merkmale aufweisen, wie begrenzte Ressourcen oder einen spezifischen Marktansatz.
API	Application Programming Interface	Eine Schnittstelle, die die Interaktion zwischen verschiedenen Softwareanwendungen ermöglicht.
MSSQL	Microsoft SQL Server	Ein relationales Datenbankverwaltungssystem von Microsoft.
MSSMS	Microsoft SQL Server Management Studio	Ein integriertes Umfeld zur Verwaltung von SQL-Datenbanken.
JWT	JSON Web Token	Ein kompaktes, URL-sicheres Mittel zur Darstellung von Ansprüchen, die zwischen zwei Parteien ausgetauscht werden.
DTO	Data Transfer Object	Ein Muster, das verwendet wird, um Daten zwischen Softwareanwendungsebenen zu übertragen.
HTTPS	Hypertext Transfer Protocol Secure	Eine erweiterte Version des HTTP-Protokolls, die eine sichere Kommunikation im Internet bietet.
JSON	JavaScript Object Notation	Ein leichtgewichtiges Daten-Austauschformat, das für Menschen leicht lesbar und für Maschinen einfach zu parsen ist.
UI	User Interface	Die Schnittstelle, über die der Benutzer mit einer Computeranwendung interagiert.
CRUD	Create, Read, Update, Delete	Ein Akronym für die vier grundlegenden Funktionen, die in Datenbankanwendungen angewendet werden.
SOLID	Single Responsibility, Open-Closed, Liskov Substitution, Interface Segregation, Dependency Inversion	Eine Gruppe von fünf Designprinzipien, die für die objektorientierte Programmierung verwendet werden.

9 Verweise

9.1 Quellen

SwaggerGen. (21. 11 2023). *Swagger*. Von Swagger Docs: https://localhost:7155 abgerufen

9.2 Tabellen

9.3 Abbildung